28th International Symposium on Algorithms and Computation

ISAAC 2017, December 9–12, 2017, Phuket, Thailand

Edited by

Yoshio Okamoto
Takeshi Tokuyama

LIPIcs – Vol. 92 – ISAAC 2017 www.dagstuhl.de/lipics

Editors

Yoshio Okamoto
Graduate School of Informatics and Engineering
The University of Electro-Communications
`okamotoy@uec.ac.jp`

Takeshi Tokuyama
Graduate School of Information Sciences
Tohoku University
`tokuyama@dais.is.tohoku.ac.jp`

ACM Classification 1998
E.1 Data Structures, F. Theory of Computation, G. Mathematics of Computing, I. Computing Methodologies

ISBN 978-3-95977-054-5

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/978-3-95977-054-5.

Publication date
December, 2017

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.ISAAC.2017.0

ISBN 978-3-95977-054-5 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

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

Invited Talks

Contributed Talks

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC 2017), held in Phuket, Thailand, December 9–12, 2017. ISAAC is an annual international symposium that covers the very wide range of topics in the field of algorithms and computation. The main purpose of the symposium is to provide a forum for researchers working in algorithms and theory of computation from all over the world. In response to our call for papers, we received 160 submissions from 36 countries, among which three submissions were withdrawn. Each submission was reviewed by at least three Program Committee members, possibly with the assistance of external reviewers. After a rigorous review process and extensive discussion, the Program Committee selected 65 papers. Two special issues of Algorithmica and International Journal of Computational Geometry and Applications will publish selected papers from ISAAC 2017. The best paper award was given to "Crossing Number for Graphs with Bounded Pathwidth" by Therese Biedl, Markus Chimani, Martin Derka and Petra Mutzel. Selected from submissions authored by students only, the best student paper award was given to "Fully Dynamic Connectivity Oracles under General Vertex Updates" by Kengo Nakamura. In addition to selected papers, the program also included plenary talks by two prominent invited speakers, Satoru Iwata, University of Tokyo, Japan, and Suresh Venkatasubramanian, University of Utah, USA. We thank all the Program Committee members and external reviewers for their professional service and volunteering their time to review the submissions under time constraints. We also thank all authors who submitted papers for consideration, thereby contributing to the high quality of the conference. We would like also to acknowledge our supporting organizations for their assistance and support, in particular Artificial Intelligence Association of Thailand, Sirindhorn International Institute of Technology, and Thammasat University. Finally, we are deeply indebted to the Organizing Committee members, Thanaruk Theeramunkung, Jittat Fakcharoenphol, Natsuda Kaothanthong, Chutima Beokhaimook, and Pokpong Songmuang, whose excellent effort and professional service to the community made the conference an unparalleled success.

December, 2017 Yoshio Okamoto and Takeshi Tokuyama

Program Committee

Isolde Adler	(University of Leeds, UK)
Patrizio Angelini	(University of Tübingen, Germany)
Markus Bläser	(Saarland University, Germany)
Yixin Cao	(Hong Kong Polytechnic University, Hong Kong)
Jean Cardinal	(Université Libre de Bruxelles, Belgium)
Parinya Chalermsook	(Aalto University, Finland)
Erin Wolf Chambers	(Saint Louis University, USA)
Kun-Mao Chao	(National Taiwan University, Taiwan)
Sevag Gharibian	(Virginia Commonwealth University, USA)
Keiko Imai	(Chuo University, Japan)
Taisuke Izumi	(Nagoya Institute of Technology, Japan)
Jesper Jansson	(Hong Kong Polytechnic University, Hong Kong)
Naoyuki Kamiyama	(Kyushu University, Japan)
Akinori Kawachi	(Osaka University, Japan)
Chung-Shou Liao	(National Tsing Hua University, Taiwan)
Yoshio Okamoto	(The University of Electro-Communications, Japan) Co-Chair
Dömötör Pálvölgyi	(Eötvös Loránd University, Hungary)
C. Pandu Rangan	(Indian Institute of Technology Madras, India)
Laura Sanità	(University of Waterloo, Canada)
Daniel Stefankovic	(University of Rochester, USA)
Takeshi Tokuyama	(Tohoku University, Japan) Co-Chair
Kei Uchizawa	(Yamagata University, Japan)
Marc van Kreveld	(Utrecht University, the Netherlands)
Haitao Wang	(Utah State University, USA)
Yajun Wang	(Microsoft, USA)
Wei Xu	(Tsinghua University, P.R. China)
Guochuan Zhang	(Zhejiang University, P.R. China)
Martin Ziegler	(KAIST, Republic of Korea)

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Mikkel Abrahamsen
Rupam Acharyya
Nieke Aerts
Thomas Dybdahl Ahle
Sara Ahmadian
Hee-Kap Ahn
Yoshinori Aono
Boris Aronov
Franz Aurenhammer
Sang Won Bae
Xiaohui Bei
Michael Bekos
Rémy Belmonte
Dietmar Berwanger
Thomas Bläsius
Marthe Bonamy
Flavia Bonomo
Cornelius Brand
Andreas Brandstädt
Nick Brettell
Gerth Stølting Brodal
Brian Brubach
Kevin Buchin
Maike Buchin
Tiziana Calamoneri
Timothy M. Chan
Hsien-Chih Chang
Hong-Bin Chen
Ho-Lin Chen
Li-Hsuan Chen
Lin Chen
Po-An Chen
Wenbin Chen
Zhihuai Chen
Eddie Cheng
Otfried Cheong
Ágnes Cseh
Fabio Cunial
Giordano Da Lozzo
Gábor Damasdi
Shantanu Das
Syamantak Das
Marc Demange
Hu Ding
Anne Driemel
Christoph Dürr
Thorsten Ehlers
Eduard Eiben
Kord Eickmeyer
Hicham El-Zein
Leah Epstein
Chenglin Fan
Lene Favrholdt
Andreas Emil Feldmann
Qilong Feng
Henry Förster
Nóra Frankl
Fabrizio Frati
Akihiro Fujiwara
Hiroshi Fujiwara
Radoslav Fulek
Jinxiang Gan
Robert Ganian
Michal Garlik
Ran Duan
Jérôme Durand-Lose
Dániel Gerbner
Archontia Giannopoulou
Omer Gold
Martin Grohe
Martin Gronemann
Martin Groß
Joachim Gudmundsson
Guru Guruganesh
Xin Han
Meng He
Pinar Heggernes
Niklas Heinsohn
Yuya Higashikawa
Wing-Kai Hon
Seok-Hee Hong
Tamás Hubai
Ling-Ju Hung
John Iacono
Takehiro Ito
Tomoko Izumi
Wanchote Jiamjitrak
Seungbum Jo
Daniel Kane
Michael Kaufmann
Balázs Keszegh
Marc Khoury
Takuya Kida
Shuji Kijima
Eun Jung Kim
Kei Kimura
Philipp Kindermann
Zoltán Király
Sándor Kisfaludi-Bak
Kim-Manuel Klein
Boris Klemz
Naoki Kobayashi
Atsushi Koike
Sudeshna Kolay
Balagopal Komarath
Christian Komusiewicz
Alexander Kononov
Christian Konrad
Matias Korman
Takeshi Koshiba
Adrian Kosowski
Nirman Kumar
Akitoshi Kawamura
Phillip Keldenich
Ritu Kundu
Jan Kynčl
Maria Kyropoulou
Bundit Laekhanukit
Pablo Pérez-Lantero
Francois Le Gall
Euiwoong Lee
Inbok Lee
Arnaud Lefebvre
Johannes Lengler
Jian Li
Yi Li
Vincent Limouzy
Hsiang-Hsuan Liu
Tengyu Ma
Enrico Malaguti
Bodo Manthey
Fábio Viduani Martinez
Kitty Meeks
Lili Mei
George Mertzios

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tillmann Miltzow
Pranabendu Misra
Shuichi Miyazaki
Matthias Mnich
Tobias Mömke
Fabrizio Montecchiani
Amer Mouawad
Haiko Müller
Wolfgang Mulzer
Elizabeth Munch
Sadagopan Narasimhan
Ilan Newman
André Nichterlein
Joanna Ochremiak
Hirotaka Ono
Fukuhito Ooshita
Sebastian Ordyniak
Yota Otachi
Rasmus Pagh
Linda Pagli
Peter Palfrader
Dénes Pálvölgyi
Evanthia Papadopoulou
Kunsoo Park
Sewon Park
Christophe Paul
Daniel Paulusma
David Peleg
Michael Pelsmajer
Claire Pennarun
Valentin Polishchuk
Chrysanthi Raftopoulou
Ashutosh Rai
Rajeev Raman
Venkatesh Raman
Raghavendra Rao B. V.
Gaurav Rattan
Marcel Roeloffzen
Vincenzo Roselli
Alfred Rossi
Marc Roth
Paweł Rzążewski
Toshiki Saitoh
R.B. Sandeep
Nitin Saurabh
Andreas Schmid
Melanie Schmidt
Santiago Segarra
Shinnosuke Seki
Xiaohan Shan
Weiran Shen
Tetsuo Shibuya
Ayumi Shinohara
Takeharu Shiraga
Stavros Sintos
Jouni Sirén
Alexander Skopalik
Michiel Smid
Shakhar Smorodinsky
Bettina Speckmann
Karteek Sreenivasaiah
Frank Staals
Fabian Stehn
Darren Strash
Ben Strasser
Damian Straszak
Yuichi Sudo
Noriyoshi Sukegawa
Xiaorui Sun
Akira Suzuki
Suguru Tamaki
Bo Tang
Liangde Tao
Jun Tarui
Nguyen Kim Thang
Dirk Oliver Theis
Srikanta Tirthapura
Lilla Tóthmérész
Torsten Ueckerdt
Sumedha Uniyal
Takeaki Uno
André van Renssen
Antoine Vigneron
Máté Vizer
Jan Vybíral
Koichi Wada
Hung-Lung Wang
Rolf Wanka
Zhewei Wei
Hao-Ting Wei
Carola Wenk
Andrew Winslow
Marcin Wrochna
Piotr Wygocki
Mingyu Xiao
Chenyang Xu
Yukiko Yamauchi
Kenji Yasunaga
Junjie Ye
Yitong Yin
Yusuke Yokosuka
Yuichi Yoshida
Ryo Yoshinaka
Hung-I Yu
Chenhao Zhang
Chihao Zhang
Jia Zhang
Jialin Zhang
Jingru Zhang
Peng Zhang
Zhao Zhang
Standa Živný

Weighted Linear Matroid Parity*

Satoru Iwata

Department of Mathematical Informatics, University of Tokyo, Tokyo, Japan
iwata@mist.i.u-tokyo.ac.jp

Abstract

The matroid parity (or matroid matching) problem, introduced as a common generalization of matching and matroid intersection problems, is so general that it requires an exponential number of oracle calls. Nevertheless, Lovasz (1978) showed that this problem admits a min-max formula and a polynomial algorithm for linearly represented matroids. Since then efficient algorithms have been developed for the linear matroid parity problem.

This talk presents a recently developed polynomial-time algorithm for the weighted linear matroid parity problem. The algorithm builds on a polynomial matrix formulation using Pfaffian and adopts a primal-dual approach based on the augmenting path algorithm of Gabow and Stallmann (1986) for the unweighted problem.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases Matroid, matching, Pfaffian, polynomial-time algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.1

Category Invited Talk

1 Introduction

The concept of matroids was introduced by Whitney [33] as a combinatorial abstraction of linear dependence. A matroid is a pair $(S, \mathcal{I})$ of a finite set E and its subset family $\mathcal{I}$ that satisfy the following axioms.

(I0) $\emptyset \in \mathcal{I}$.

(I1) $I \subseteq J \in \mathcal{I} \Rightarrow I \in \mathcal{I}$.

(I2) $I, J \in \mathcal{I}$, $|I| < |J| \Rightarrow \exists e \in J \setminus I, I \cup \{e\} \in \mathcal{I}$.

A primary example is a set S of vectors in a certain linear space, where $\mathcal{I}$ is the collection of vector subsets that are linearly independent. Such a matroid representable in this way is called a linear matroid.

The importance of matroids in the context of combinatorial otimization was established by Edmonds [6, 8]. In particular, the framework of matroid intersection generalizes bipartite matching and captures various combinatorial optimizations problems solvable in polynomial time.

As a common generalization of matroid intersection and nonbipartite matching, Lawler [19] introduced matroid parity. Suppose that the ground set S of a matroid $(S, \mathcal{I})$ is partitioned into pairs, called lines. The matroid parity problem asks for finding a maximum cardinality independent set that is a disjoint union of lines. It turned out, however, that this framework is too general to be solvable. In fact, it includes NP-hard problems and requires exponential number of independence oracle calls [17, 21]. A PTAS for this general framework has been developed only recently [20].

* This work was partially supported by JST CREST (JPMJCR14D2).

For linear matroids, however, Lovász [21, 23, 24] showed a min-max formula and presented a polynomial algorithm that is applicable if the linear representation is available. Since then, efficient combinatorial algorithms have been developed for this linear matroid parity problem [11, 29, 30]. Gabow and Stallmann [11] developed an augmenting path algorithm with the aid of a linear algebraic trick, which was later extended to the linear delta-matroid parity problem [13]. Orlin and Vande Vate [30] provided an algorithm that solves this problem by repeatedly solving matroid intersection problems coming from the min-max theorem. Later, Orlin [29] improved the running time bound of this algorithm. The current best deterministic running time bound due to [11, 29] is $O(nm^{\omega})$, where n is the cardinality of the ground set, m is the rank of the linear matroid, and ω is the matrix multiplication exponent, which is at most 2.38.

Since matching and matroid intersection algorithms [4, 7] have been successfully extended to their weighted version [5, 9, 15, 18], it is natural to expect polynomial algorithms for the weighted linear matroid parity problem. In fact, a recent work [16] has presented a combinatorial, deterministic, polynomial-time algorithm for the weighted linear matroid parity problem. The algorithm builds on a polynomial matrix formulation, which naturally extends the one discussed in [12] for the unweighted problem.

2 The Linear Matroid Parity Problem

Let A be a matrix of row-full rank over an arbitrary field $\mathbf{K}$ with row set U and column set V. Assume that $n = |V|$ are even. The column set V is partitioned into pairs, called *lines*. Each $v \in V$ has its *mate* $\bar{v}$ such that $\{v, \bar{v}\}$ is a line. We denote by L the set of lines.

The linear dependence of the column vectors naturally defines a matroid $\mathbf{M}(A)$ on V. The independent set family $\mathcal{I}$ is given by $\mathcal{I} = \{J \mid \operatorname{rank} A[U, J] = |J|\}$ A subset $X \subseteq V$ is called a *parity set* if it consists of lines. The linear matroid parity problem asks for finding an independent parity set of maximum cardinality. We denote the optimal value by $\nu(A, L)$ This problem generalizes finding a maximum matching in graphs and a maximum common independent set of a pair of linear matroids on the same ground set.

For a skew-symmetric matrix Φ whose rows and columns are indexed by W, the *support graph* of Φ is the graph $G = (W, E)$ with edge set $E = \{(u, v) \mid \Phi_{uv} \neq 0\}$. We denote by $\operatorname{Pf} \Phi$ the *Pfaffian* of Φ, which is defined as follows:

$$\operatorname{Pf} \Phi = \sum_{M} \sigma_M \prod_{(u,v) \in M} \Phi_{uv},$$

where the sum is taken over all perfect matchings M in G and σ_M takes ± 1 in a suitable manner, see [25]. It is well-known that $\det \Phi = (\operatorname{Pf} \Phi)^2$ and $\operatorname{Pf} (S \Phi S^{\top}) = \operatorname{Pf} \Phi \cdot \det S$ for any square matrix S.

Associated with the linear matroid parity problem, we consider a skew-symmetric matrix Φ_A defined by

$$\Phi_A = \begin{pmatrix} O & A \\ -A^{\top} & D \end{pmatrix},$$

where D is a block-diagonal matrix in which each block is a 2×2 skew-symmetric matrix $D_\ell = \begin{pmatrix} 0 & -\tau_\ell \\ \tau_\ell & 0 \end{pmatrix}$ corresponding to a line $\ell \in L$. Assume that the coefficients τ_ℓ are independent parameters (or indeterminates).

▶ **Lemma 1** ([12]). *The optimal value $\nu(A, L)$ of the linear matroid parity problem is given by*

$$\nu(A, L) = \operatorname{rank} \Phi_A - n.$$

This characterization leads to an efficient randomized algorithm for solving the linear matroid parity problem in high probability by substituting randomly generated numbers to the indeterminates. In fact, Lovász [22] introduced such an approach using another skew-symmetric matrix, and Cheung, Lau, and Leung [3] improved it to run in $O(nm^{\omega-1})$ time, extending the techniques of Harvey [14] developed for matching and matroid intersection.

3 The Minimum-Weight Parity Base Problem

In the same setting as the linear matroid parity problem, suppose that each line $\ell \in L$ has a weight $w_\ell \in \mathbb{R}$. Let $\mathcal{B}$ be the base family of $\mathbf{M}(A)$, i.e., $\mathcal{B} = \{B \mid \operatorname{rank} A[U, B] = |B| = |U|\}$. A base $B \in \mathcal{B}$ is called a *parity base* if it consists of lines. As a weighted version of the linear matroid parity problem, we will consider the problem of finding a parity base of minimum weight, where the weight of a parity base is the sum of the weights of lines in it. We denote the optimal value by $\zeta(A, L, w)$. This problem generalizes finding a minimum-weight perfect matching in graphs and a minimum-weight common base of a pair of linear matroids on the same ground set.

As another weighted version of the matroid parity problem, one can think of finding an independent parity set of maximum weight. This problem can be easily reduced to the minimum-weight parity base problem.

Associated with the minimum-weight parity base problem, we consider a skew-symmetric polynomial matrix $\Phi_A(\theta)$ in variable θ defined by

$$\Phi_A(\theta) = \begin{pmatrix} O & A \\ -A^\top & D(\theta) \end{pmatrix},$$

where $D(\theta)$ is a block-diagonal matrix in which each block is a 2×2 skew-symmetric polynomial matrix $D_\ell(\theta) = \begin{pmatrix} 0 & -\tau_\ell \theta^{w_\ell} \\ \tau_\ell \theta^{w_\ell} & 0 \end{pmatrix}$ corresponding to a line $\ell \in L$. Assume that the coefficients τ_ℓ are independent parameters (or indeterminates).

We have the following lemma that associates the optimal value of the minimum-weight parity base problem with $\operatorname{Pf} \Phi_A(\theta)$.

▶ **Lemma 2** ([16]). *The optimal value of the minimum-weight parity base problem is given by*

$$\zeta(A, L, w) = \sum_{\ell \in L} w_\ell - \deg_\theta \operatorname{Pf} \Phi_A(\theta).$$

In particular, if $\operatorname{Pf} \Phi_A(\theta) = 0$*, then there is no parity base.*

Note that Lemma 2 does not immediately lead to a polynomial-time algorithm for the minimum weight parity base problem. This is because computing the degree of the Pfaffian of a skew-symmetric polynomial matrix is not so easy. Indeed, randomized algorithms in [2, 3] for the weighted linear matroid parity problem compute the degree of the Pfaffian of another skew-symmetric polynomial matrix, which results in pseudopolynomial complexity.

Starting with the characterization in Lemma 2, we have developed a combinatorial, deterministic polynomial-time algorithm for the minimum-weight parity base problem [16].

The algorithm employs a modification of the augmenting path search procedure for the unweighted problem by Gabow and Stallmann [11]. The correctness proof for the optimality is based on the idea of combinatorial relaxation for polynomial matrices due to Murota [28].

▶ **Theorem 3** ([16]). *The minimum-weight parity base problem can be solved with* $O(mn^3)$ *arithmetic operations over* $\mathbf{K}$*, where* $m = |U|$ *and* $n = |V|$*.*

This leads to a strongly polynomial algorithm for linear matroids represented over a finite field. For linear matroids represented over the rational field, one can exploit that algorithm to solve the problem in polynomial time.

4 Applications

The linear matroid parity problem finds various applications: structural solvability analysis of passive electric networks [27], pinning down planar skeleton structures [25], and maximum genus cellular embedding of graphs [10]. We describe two interesting applications of the weighted matroid parity problem in combinatorial optimization.

A T-path in a graph is a path between two distinct vertices in the terminal set T. Mader [26] showed a min-max characterization of the maximum number of openly disjoint T-paths. The problem can be equivalently formulated in terms of $\mathcal{S}$-paths, where $\mathcal{S}$ is a partition of T and an $\mathcal{S}$-path is a T-path between two different components of $\mathcal{S}$. Lovász [24] formulated the problem as a matroid matching problem and showed that one can find a maximum number of disjoint $\mathcal{S}$-paths in polynomial time. Schrijver [32] has described a more direct reduction to the linear matroid parity problem.

As a weighted version of the disjoint $\mathcal{S}$-paths problem, it is quite natural to think of finding disjoint $\mathcal{S}$-paths of minimum total length. It is not immediately clear that this problem reduces to the weighted linear matroid parity problem. A recent paper of Yamaguchi [34] clarifies that this is indeed the case.

The weighted linear matroid parity has also been used in the design of approximation algorithms. Prömel and Steger [31] provided a 5/3-approximation algorithm for the Steiner tree problem with the aid of the weighted parity problem for graphic matroids. Even though the performance ratio is larger than the current best one for the Steiner tree problem [1], this suggests that there may be other combinatorial optimization problems that admit new approximation algorithms using weighted linear matroid parity.

References

1 J. Byrka, F. Grandoni, T. Rothvoss, L. Sanità: Steiner tree approximation via iterative randomized rounding, *J. ACM*, 60 (2013), 6: 1–33.

2 P. M. Camerini, G. Galbiati, and F. Maffioli: Random pseudo-polynomial algorithms for exact matroid problems, *J. Algorithms*, 13 (1992), 258–273.

3 H. Y. Cheung, L. C. Lau, and K. M. Leung: Algebraic algorithms for linear matroid parity problems, *ACM Trans. Algorithms*, 10 (2014), 10: 1–26.

4 J. Edmonds: Paths, trees, and flowers, *Canadian J. Math.* 17 (1965), 449–467.

5 J. Edmonds: Maximum matching and a polyhedron with 0, 1-vertices, *J. Research National Bureau of Standards*, Section B, 69 (1965), 125–130.

6 J. Edmonds: Minimum partition of a matroid into independent sets, *J. Research National Bureau of Standards*, Section B, 69 (1965), 67–72.

7 J. Edmonds: Matroid partition, *Mathematics of the Decision Sciences: Part 1* (G.B. Dantzig and A.F. Veinott, eds.), American Mathematical Society, 1968, 335–345.

8 J. Edmonds: Matroids and the greedy algorithm, *Math. Programming*

9 J. Edmonds: Matroid intersection, *Ann. Discrete Math.*, 4 (1979), 39–49.

10 M. L. Furst, J. L. Gross, and L. A. McGeoch: Finding a maximum-genus graph imbedding, *J. ACM*, 35 (1988), 523–534.

11 H. N. Gabow and M. Stallmann: An augmenting path algorithm for linear matroid parity, *Combinatorica*, 6 (1986), 123–150.

12 J. F. Geelen and S. Iwata: Matroid matching via mixed skew-symmetric matrices, *Combinatorica*, 25 (2005), 187–215.

13 J. F. Geelen, S. Iwata, and K. Murota: The linear delta-matroid parity problem, *J. Combinatorial Theory*, Ser. B, 88 (2003), 377–398.

14 N. J. A. Harvey: Algebraic algorithms for matching and matroid problems, *SIAM J. Comput.*, 39 (2009), 679–702.

15 M. Iri and N. Tomizawa: An algorithm for finding an optimal "independent assignment", *J. Oper. Res. Soc. Japan*, 19 (1976), 32–57.

16 S. Iwata and Y. Kobayashi: A weighted linear matroid parity algorithm, *Proceedings of the 49th ACM Annual Symposium on Theory of Computing*, 2017, 264–276.

17 P. M. Jensen and B. Korte: Complexity of matroid property algorithms, *SIAM J. Comput.*, 11 (1982), 184–190.

18 E.L. Lawler: Matroid intersection algorithms, *Math. Programming*, 9 (1975), 31–56.

19 E.L. Lawler: *Combinatorial Optimization — Networks and Matroids*, Holt, Rinehalt, and Winston, 1976.

20 J. Lee, M. Sviridenko, and J. Vondrák: Matroid matching: The power of local search, *SIAM J. Comput.*, 42 (2013), 357–379.

21 L. Lovász: The matroid matching problem, *Algebraic Methods in Graph Theory*, Colloq. Math. Soc. János Bolyai, 25 (1978), 495–517.

22 L. Lovász: On determinants, matchings, and random algorithms, *Fundamentals of Computation Theory*, L. Budach ed., Academie-Verlag, 1979, 565–574.

23 L. Lovász: Selecting independent lines from a family of lines in a space, *Acta Sci. Math.*, 42 (1980), 121–131.

24 L. Lovász: Matroid matching and some applications, *J. Combinatorial Theory*, Ser. B, 28 (1980), 208–236.

25 L. Lovász and M. D. Plummer: *Matching Theory*, North-Holland, Amsterdam, 1986.

26 W. Mader: Über die Maximalzahl krezungsfreier H-Wege, *Arch. Math.*, 31 (1978), 387–402, 1978.

27 M. M. Milić: General passive networks — Solvability, degeneracies, and order of complexity, *IEEE Trans. Circuits Syst.*, 21 (1974), 177–183.

28 K. Murota: Computing the degree of determinants via combinatorial relaxation, *SIAM J. Comput.*, 24 (1995), 765–796.

29 J. B. Orlin: A fast, simpler algorithm for the matroid parity problem, *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization*, LNCS 5035, Springer-Verlag, 2008, 240–258.

30 J. B. Orlin and J. H. Vande Vate: Solving the linear matroid parity problem as a sequence of matroid intersection problems, *Math. Programming*, 47 (1990), 81–106.

31 H. J. Prömel and A. Steger: A new approximation algorithm for the Steiner tree problem with performance ratio 5/3, *J. Algorithms*, 36 (2000), 89–101.

32 A. Schrijver: *Combinatorial Optimization — Polyhedra and Efficiency*, Springer-Verlag, Berlin, 2003.

33 H. Whitney: On the abstract properties of linear dependence, *Amer. J. Math*, 57 (1935), 509–533.

34 Y. Yamaguchi: Shortest disjoint $\mathcal{S}$-paths via weighted linear matroid parity, *Proceedings of the 27th International Symposium on Algorithms and Computation*, 2016, 63: 1–13.

Computational Philosophy: On Fairness in Automated Decision Making

Suresh Venkatasubramanian

School of Computing, University of Utah, Salt Lake City, USA
suresh@cs.utah.edu

Abstract

As more and more of our lives are taken over by automated decision making systems (whether it be for hiring, college admissions, criminal justice or loans), we have begun to ask whether these systems are making decisions that humans would consider fair, or non-discriminatory. The problem is that notions of fairness, discrimination, transparency and accountability are concepts in society and the law that have no obvious formal analog.

But our algorithms speak the language of mathematics. And so if we want to encode our beliefs into automated decision systems, we must formalize them precisely, while still capturing the natural imprecision and ambiguity in these ideas.

In this talk, I'll survey the new field of fairness, accountability and transparency in computer science. I'll focus on how we formalize these notions, how they connect to traditional notions in theoretical computer science, and even describe some impossibility results that arise from this formalization. I'll conclude with some open questions.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, J.4 Social and Behavioral Sciences

Keywords and phrases fairness, transparency, accountability, impossibility results

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.2

Category Invited Talk

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Faster Algorithms for Growing Prioritized Disks and Rectangles

Hee-Kap Ahn[*1], Sang Won Bae[†2], Jongmin Choi[‡3], Matias Korman[§4], Wolfgang Mulzer[¶5], Eunjin Oh[‖6], Ji-won Park[7], André van Renssen[††8], and Antoine Vigneron[‡‡9]**

1 **Dept. Computer Science and Engineering, POSTECH, Pohang, Republic of Korea**
heekap@postech.ac.kr

2 **Dept. Computer Science, Kyonggi University, Kionggi, Republic of Korea**
swbae@kgu.ac.kr

3 **Dept. Computer Science and Engineering, POSTECH, Pohang, Republic of Korea**
icothos@postech.ac.kr

4 **Tohoku University, Sendai, Japan**
mati@dais.is.tohoku.ac.jp

5 **Institut für Informatik, Freie Universität Berlin, Berlin, Germany**
mulzer@inf.fu-berlin.de

6 **Dept. Computer Science and Engineering, POSTECH, Pohang, Republic of Korea**
jin9082@postech.ac.kr

7 **School of Computing, KAIST, Daejeon, Republic of Korea**
wldnjs1727@kaist.ac.kr

8 **National Institute of Informatics (NII), Tokyo, and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
andre@nii.ac.jp

9 **School of Electrical and Computer Engineering, UNIST, Ulsan, Republic of Korea**
antoine@unist.ac.kr

Abstract

Motivated by map labeling, we study the problem in which we are given a collection of n disks in the plane that grow at possibly different speeds. Whenever two disks meet, the one with the

[*] The work was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP–2017–0–00905) supervised by the IITP (Institute for Information & communications Technology Promotion.).

[†] S.W. Bae was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057220).

[‡] The work was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP–2017–0–00905) supervised by the IITP (Institute for Information & communications Technology Promotion.).

[§] M. K. was supported in part by KAKENHI Nos. 15H02665 and 17K12635, Japan.

[¶] W. M. was supported in part by DFG Grants MU 3501/1 and MU 3501/2.

[‖] The work was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP–2017–0–00905) supervised by the IITP (Institute for Information & communications Technology Promotion.).

[**] J.-W. Park was supported by the NRF Grant 2011-0030044 (SRC-GAIA) funded by the Korea government (MSIP).

[††] A. v. R. was supported by JST ERATO Grant Number JPMJER1201, Japan.

[‡‡] A. Vigneron was supported by the 2016 Research Fund (1.160054.01) of UNIST (Ulsan National Institute of Science and Technology).

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 3; pp. 3:1–3:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

higher index disappears. This problem was introduced by Funke, Krumpe, and Storandt [IWOCA 2016]. We provide the first general subquadratic algorithm for computing the times and the order of disappearance. Our algorithm also works for other shapes (such as rectangles) and in any fixed dimension.

Using quadtrees, we provide an alternative algorithm that runs in near linear time, although this second algorithm has a logarithmic dependence on either the ratio of the fastest speed to the slowest speed of disks or the spread of the disk centers (the ratio of the maximum to the minimum distance between them). Our result improves the running times of previous algorithms by Funke, Krumpe, and Storandt [IWOCA 2016], Bahrdt *et al.* [ALENEX 2017], and Funke and Storandt [EWCG 2017]. Finally, we give an $\Omega(n \log n)$ lower bound on the problem, showing that our quadtree algorithms are almost tight.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems–Geometrical problems and computations

Keywords and phrases map labeling, growing disks, elimination order

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.3

1 Introduction

Suppose we are given a sequence $D_1, \ldots, D_n$ of n *growing disks*. At time $t = 0$, each disk D_i starts out as a point $p_i \in \mathbb{R}^2$, and as time passes, it grows linearly with *growth rate* $v_i > 0$. Thus, at any time $t \geq 0$, the disk D_i is centered at p_i and has radius tv_i. The position of a disk in the sequence corresponds to its *priority* (the smaller the index, the higher its priority). Whenever two disks meet, we eliminate the one with lower priority from the arrangement. More precisely, for any $1 \leq i < j \leq n$, let $t(i,j) > 0$ be the time when D_i and D_j touch, i.e., $t(i,j) = |p_i p_j|/(v_i + v_j)$. Then, if neither of the two disks D_i and D_j has been removed before time $t(i,j)$, we eliminate D_j at this time, while D_i keeps growing. Our goal is to determine the *elimination order*, that is, the instants of time and the order in which the disks are removed from the arrangement.

Motivated by map labeling, this problem was first considered by Funke, Krumpe, and Storandt [7]. As one zooms out from a labeled map, labels grow in size. Clearly, we do not want the labels to overlap, so whenever this happens, one of the two is removed. This creates the need to determine when and in which order the labels need to be discarded. Funke, Krumpe, and Storandt [7] observed that a straightforward simulation of the growth process with a priority queue solves the problem in time $O(n^2 \log n)$. They also gave an algorithm that runs in expected time $O(n(\log^6 n + \Delta^2 \log^2 n + \Delta^4 \log n))$, where $\Delta = \max_i v_i / \min_j v_j$ is the maximum ratio between two growth rates. Subsequently, Bahrdt *et al.* [2] improved this to an algorithm that runs in worst-case time $O(\Delta^2 n(\log n + \Delta^2))$. This generalizes to growing balls in arbitrary fixed dimension d, with running time $O(\Delta^d n(\log n + \Delta^d))$. Recently, Funke and Storandt [8] presented two further parameterized algorithms for the problem. The first algorithm runs in time $O(n \log \Delta(\log n + \Delta^{d-1}))$, for arbitrary dimension d, while the second algorithm is specialized for the plane and runs in time $O(Cn \log^{O(1)} n)$, where C denotes the number of distinct growth rates. If we are interested only in the first pair of touching disks, our problem is equivalent to the *weighted closest pair* of the disk centers. Formann showed how to compute it in optimal $O(n \log n)$ time [6].

Table 1 Summary of our results. The $O(dn^2)$-time algorithm in the first row works for growing objects of any shape in $\mathbb{R}^d$ such that the touching time of any pair of them can be computed in $O(d)$ steps. $\mathcal{SA}_k$ stands for any semialgebraic shape that is described with k parameters. Φ denotes the *spread* of the disk centers and $\Delta = \max_i v_i / \min_j v_j$ is the maximum ratio between two growth rates.

Shape	**Time**	**Method**	**Where**
Balls, Boxes in $\mathbb{R}^d$	$O(dn^2)$	Priority sort	Section 2
Disks in $\mathbb{R}^2$	expected $O(n^{5/3+\varepsilon})$	Bucketing	Section 3
Rectangles in $\mathbb{R}^2$	expected $O(n^{11/6+\varepsilon})$		
$\mathcal{SA}_k$, $k \geq 4$	expected $O(n^{2-1/(2k-2)+\varepsilon})$		
Cubes in $\mathbb{R}^d$	$O(n \log^{d+2} n)$	Linearity of queries	Section 4
Disks in $\mathbb{R}^2$	$O(n \log \Phi \min\{\log \Delta, \log \Phi\})$	Quadtree	Section 5.1
Disks in $\mathbb{R}^2$	$O(n(\log n + \min\{\log \Delta, \log \Phi\}))$	Compressed quadtree	Section 5.2

Our results. We first present a simple algorithm that runs in time $O(dn^2)$ in any fixed dimension d (Section 2). In Section 3, we combine it with an advanced data structure for querying lower envelopes of algebraic surfaces [1, 11] and with bucketing. In particular, the algorithm runs in $O(n^{5/3+\varepsilon})$ and $O(n^{11/6+\varepsilon})$ expected time for disks and rectangles in two dimensions, respectively. These are the first subquadratic-time algorithms for the problem. More generally, we show that the elimination sequence of a set of n growing objects of any semi-algebraic shape described with $k \geq 4$ parameters can be computed in subquadratic time for any fixed k. In Section 4, we consider the case of growing squares. These objects are much simpler, hence we can use ray shooting techniques and similar properties to reduce the running time to $O(n \log^{d+2} n)$.

In Section 5, we consider a completely different approach based on quadtrees. The running time of these algorithms also depends on the *spread* Φ of the disk centers (that is, the ratio of the maximum to the minimum distance between disk centers) and the ratio Δ between the fastest and slowest speed of the disks. Table 1 provides a summary of our results. Finally, we give an $\Omega(n \log n)$ lower bound using a simple reduction from sorting. Our algorithm using compressed quadtrees is thus nearly optimal as well as it is an improvement over Bahrdt *et al.*'s algorithm [2] that runs in $O(\Delta^2 n(\log n + \Delta^2))$ time.

Note. Parallel to our work, Castermans *et al.* [4] considered a variant of the problem for squares in the plane. Whenever two squares meet, they are replaced by a new one located at their weighted center. Like us, they are interested in the elimination/replacement sequence. Although our algorithms are slightly faster (by polylogarithmic factors) and more general (their algorithm can only handle square shapes), we emphasize that they are not comparable, since our techniques do not apply in their setting.

Notation. For any $1 \leq i \leq n$, we denote by t_i the time at which disk D_i is eliminated. Since D_1 will never be eliminated, we set $t_1 = \infty$. We denote by $t(i,j) = |p_i p_j|/(v_i + v_j)$ the time at which disks the D_i and D_j would touch, supposing that no other disk has interfered. We assume general position, meaning that all times $t(i,j)$, for $i \neq j$, are pairwise distinct.

2 A simple quadratic algorithm

We provide a simple iterative way to determine the elimination times t_i. This method will be used for small groups of disks afterwards. As noted above, we have $t_1 = \infty$. For $i \geq 2$, the next lemma shows how to find t_i, provided that $t_1, \ldots, t_{i-1}$ are known.

Algorithm 1 A quadratic time algorithm

```
function ELIMINATIONORDER($p_1, \dots, p_n$, $v_1, \dots, v_n$)
    $t_1 \leftarrow \infty$
    for $i \leftarrow 2, n$ do
        $t_i \leftarrow t(i, 1)$
        for $j \leftarrow 2, i-1$ do
            if $t_j \geq t(i, j)$ and $t_i \geq t(i, j)$ then
                $t_i \leftarrow t(i, j)$
    $S \leftarrow (D_1, \dots, D_n)$
    Sort $S$ using key $t_i$ for each disk $D_i$
    return $S$
```

▶ **Lemma 2.1.** *Let $i \in \{2, \dots, n\}$, and let*

$$j^* = \operatorname{argmin}_{j=1,\dots,i-1}\{t(i,j) \mid t(i,j) \leq t_j\}.$$

Then, $t_i = t(i, j^*)$, i.e., the disk D_i is eliminated by the disk D_{j^*}.

Proof. On the one hand, we have $t_i \leq t(i, j^*)$, because at time $t(i, j^*)$, the disk D_i would meet the disk D_{j^*} that has higher priority and that has not been eliminated yet. On the other hand, we have $t_i \geq t(i, j^*)$, because every disk that D_i could meet before time $t(i, j^*)$ either has lower priority or has been eliminated before the encounter. ◀

Lemma 2.1 leads to a straightforward iterative algorithm, see Algorithm 1.

▶ **Theorem 2.2.** *Algorithm 1 computes the elimination order of a set of prioritized disks in $O(n^2)$ time. It generalizes to growing objects of any shape in $\mathbb{R}^d$ such that the touching time of any pair of them can be computed in $O(d)$ steps, with running time $O(dn^2)$.*

Proof. The correctness follows directly from Lemma 2.1. The running time analysis is straightforward. Lemma 2.1 is purely combinatorial and requires only that the times $t(i, j)$ are well defined. Thus, Algorithm 1 can be generalized to balls and rectangles in $\mathbb{R}^d$ by using an appropriate subroutine for computing $t(i, j)$. This subroutine takes $O(d)$ steps. ◀

3 A subquadratic algorithm using bucketing

We now improve Algorithm 1 by using a bucketing approach and lifting the problem to higher dimensions. For this purpose, we will use a data structure for querying lower envelopes in $\mathbb{R}^4$, which allows us to compute t_i in increasing order of i.

Suppose that for a set $B \subset \{1, \dots, n\}$ of indices, we know the elimination time t_j of any D_j with $j \in B$. In an *elimination query*, we are given a query index $q > \max B$, and we ask for the disk D_{j^*} with $j^* \in B$, that eliminates the query disk D_q. The argument from Lemma 2.1 shows that we can find j^* as follows

$$j^* = \operatorname{argmin}_{j \in B}\{t(q,j) \mid t(q,j) \leq t_j\}.$$

This leads to a natural interpretation of elimination queries: a query disk D corresponds to a point $(x, y, v) \in \mathbb{R}^3$, where (x, y) is the center of D and v is the growth rate. For each $j \in B$, consider the function $f_j : \mathbb{R}^3 \to \mathbb{R}$ defined by

$$f_j(x, y, v) = \begin{cases} t(j, D(x, y, v)), & \text{if } t(j, D(x, y, v)) < t_j, \\ \infty, & \text{otherwise,} \end{cases}$$

where $t(j, D(x, y, v))$ denotes the time when D_j and the growing disk given by (x, y, v) touch. For $q > \max B$, let $(x_q, y_q, v_q) \in \mathbb{R}^3$ be the point that represents D_q. Then, the elimination query q corresponds to finding the point vertically above (x_q, y_q, v_q) in the lower envelope of the graphs of the functions f_j for all $j \in B$. The following lemma is a direct consequence of a result by Agarwal et al. [1].

▶ **Lemma 3.1.** *Let $B \subset \{1, \dots, n\}$ with $|B| = m$. Then, for any fixed $\varepsilon > 0$, elimination queries for B can be answered in $O(\log^2 m)$ time, after randomized expected preprocessing time $O(m^{3+\varepsilon})$.*

We describe our subquadratic algorithm. Set $m = \lfloor n^{1/3} \rfloor$. We group the disks into $\lceil n/m \rceil$ *buckets* $B_1, \dots, B_{\lceil n/m \rceil}$ such that the kth bucket B_k contains the disks $D_{(k-1)m+1}, \dots, D_{km}$. There are $O(n^{2/3})$ buckets, each of which contains at most m disks. As before, we compute the elimination times $t_1, \dots, t_n$ in this order. As soon as the elimination times of all the disks in a bucket B_k have been determined, we construct the elimination query data structure for B_k. For each bucket, this takes $O(n^{1+\varepsilon})$ expected time, for a total time of $O(n^{5/3+\varepsilon})$.

Now, in order to determine the elimination time t_i of a disk D_i, note that we must check the previous buckets (as well as the bucket containing D_i). We first perform elimination queries for the previous buckets, that is, buckets B_k with $1 \le k \le \lfloor (i-1)/m \rfloor$. There are $O(n^{2/3})$ such queries, so this takes $O(n^{2/3} \log^2 n)$ time. Then, we handle the disks that are in the same bucket as D_i by brute force, which takes $O(n^{1/3})$ time. Overall, the running time is dominated by the time spent in preprocessing the buckets for elimination queries, which takes $O(n^{5/3+\varepsilon})$ expected time.

▶ **Theorem 3.2.** *The elimination sequence of a set of n growing disks can be computed in $O(n^{5/3+\varepsilon})$ expected time for any fixed $\varepsilon > 0$.*

As before, our algorithm generalizes to other types of shapes. Consider for example the problem of growing rectangles in $\mathbb{R}^2$. Each rectangle is given by 4 parameters: the x- and y-coordinates of two opposite corners after one unit of time (these values allow us to also obtain the center and the speed of the rectangle). Thus, the data structure for elimination queries is obtained by computing a lower envelope in $\mathbb{R}^5$. Given m growing rectangles, such a data structure with query time $O(\log m)$ can be constructed in $O(m^{6+\varepsilon})$ expected time for any fixed $\varepsilon > 0$ [11]. We now apply the same approach as for growing disks, but using buckets of size $m = \lfloor n^{1/6} \rfloor$.

▶ **Theorem 3.3.** *The elimination sequence of a set of n growing rectangles can be computed in $O(n^{11/6+\varepsilon})$ expected time for any $\varepsilon > 0$.*

More generally, we can use regions defined by any semi-algebraic shape of constant complexity. If the shape of the object is described with $k \ge 4$ parameters, we need to construct the lower envelope of n surfaces in $\mathbb{R}^{k+1}$ to answer elimination queries. After $O(n^{2k-2+\varepsilon})$-time preprocessing, we can answer queries in logarithmic time [11] (again, for any fixed $\varepsilon > 0$). The optimal size of the buckets is $n^{1/(2k-2)}$, which gives an overall running time of $O\left(n^{\frac{4k-5}{2k-2}+\varepsilon}\right)$, which is subquadratic for any fixed $k \ge 4$.

▶ **Theorem 3.4.** *The elimination sequence of a set of n growing objects of any semi-algebraic shape, each described with $k \ge 4$ parameters can be computed in $O\left(n^{2-\frac{1}{2k-2}+\varepsilon}\right)$ expected time for any $\varepsilon > 0$.*

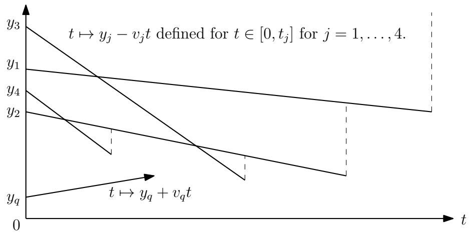

$t \mapsto y_j - v_j t$ defined for $t \in [0, t_j]$ for $j = 1, \ldots, 4$.
y_3
y_1
y_4
y_2
y_q
$t \mapsto y_q + v_q t$
0
t

Figure 1 The lower envelope of four line segments. An elimination query for a square D_q with center (x_q, y_q) and growth rate v_q consists of shooting a ray $t \mapsto y_q + v_q t$ from below.

4 Growing cubes

Axis-aligned cubes in $\mathbb{R}^d$ are described with $d+1$ parameters. Thus, the approach from the previous section applies. However, elimination queries become much easier, since they are linear functions on the input. In this section, we combine the bucketing approach with ray shooting techniques for lines to reduce the running time to an almost linear bound.

To simplify the presentation, we first assume that $d = 2$. Now, a sequence of n growing squares is given by the centers $p_1, \ldots, p_n$ and the growth rates $v_1, \ldots, v_n$. At time $t \geq 0$, each square D_i has edge length $2v_i t$. We consider the four *quadrants* around each center $p_i = (x_i, y_i)$. The *north*, *east*, *south*, and *west* quadrants are, respectively, $\{(x, y) \in \mathbb{R}^2 \mid y - y_i \geq |x - x_i|\}$, $\{(x, y) \in \mathbb{R}^2 \mid x - x_i \geq |y - y_i|\}$, $\{(x, y) \in \mathbb{R}^2 \mid -(y - y_i) \geq |x - x_i|\}$, and $\{(x, y) \in \mathbb{R}^2 \mid -(x - x_i) \geq |y - y_i|\}$.

Suppose that p_j is in the north quadrant of p_i. Then, the possible elimination time of D_i and D_j is $t(i, j) = (y_j - y_i)/(v_i + v_j)$. Thus, suppose we have a set $B \subset \{1, \ldots, n\}$ of m growing cubes, and let $q > \max B$ such that all centers p_j with $j \in B$ lie in the north quadrant of p_q. Then, an elimination query for q in B is essentially a two-dimensional problem: the x-coordinates do not matter any more. We can solve it using ray-shooting for the lower envelope of a set of line segments in $\mathbb{R}^2$.

▶ **Lemma 4.1.** *Let $B \subset \{1, \ldots, n\}$, $|B| = m$. We can preprocess B in $O(m \log m)$ time, so that elimination queries can be answered in $O(\log m)$ time, given that the centers of the squares in B lie in the north quadrant of the query square D_q.*

Proof. For each $j \in B$, consider the line segment $t \mapsto y_j - v_j t$, defined for $t \in [0, t_j]$. See Figure 1. All these line segments intersect the line $t = 0$, so their lower envelope has at most $\lambda_2(m) = 2m - 1$ edges, where $\lambda_2(m)$ denotes the maximum length of a Davenport-Schinzel sequence of order 2 with alphabet size m [12]. An elimination query for a square D_q with center (x_q, y_q) and growth rate v_q consists of shooting a ray $t \mapsto y_q + v_q t$ from below. Thus, we first compute the lower envelope in $O(m \log m)$ time [10]. Then we build a ray-shooting data structure for this lower envelope, which takes $O(m)$ preprocessing time with $O(\log m)$ query time [5]. ◀

We now give a slightly less efficient data structure that does not require B to be in the north quadrant of D_i.

▶ **Lemma 4.2.** *Let $B \subset \{1, \ldots, n\}$, $|B| = m$. We can preprocess B in time $O(m \log^3 m)$ so that elimination queries can be answered in $O(\log^3 m)$ time.*

Proof. Our aim is to build a data structure for each quadrant that answers which square (if any) of B in the quadrant will be the first to eliminate the query square. To answer a query D_q, we query the data structure for each quadrant, and we return the minimum value.

For each quadrant, the data structure is a two-dimensional range tree [3], where the coordinate axes have been rotated by an angle of $\pi/4$, so that the new coordinate axes are the bisectors of the original ones. For each canonical subset of each range tree, we construct the data structure of Lemma 4.1.

Now, given the query disk D_q and a quadrant, the centers of the disks of B in this quadrant are in the union of $O(\log^2 m)$ canonical subsets. So we query the $O(\log^2 m)$ corresponding data structures in $O(\log m)$ time each, and we return the result with the smallest timestamp. All these data structures can be built in $O(m \log^3 m)$ time. ◀

Once we have the data structure for elimination queries, we can apply the bucketing technique from Section 3. This time, we will use varying bucket sizes as points are processed. More precisely, we construct a balanced binary tree T whose leaves represent the squares $D_1, \ldots, D_n$, from left to right. As usual, a node $\nu \in T$ represents the subset that consists of the leaves in the subtree that is rooted in ν.

As soon as the elimination times of all the disks associated with a node of T have been determined, we compute the elimination query structure from Lemma 4.2. Thus, after we have determined t_j for all $j < i$, we can find t_i in $O(\log^4 n)$ time by querying the data structures recorded at $O(\log n)$ nodes of T (at most one node per level in the tree will be queried). The running time is bounded by the time needed to preprocess the points for elimination queries ($O(n \log^3 n)$ per level). So overall, this algorithm runs in $O(n \log^4 n)$ time. In higher dimensions, this bound increases by a factor $O(\log n)$ per dimension, as we need one more level in the range tree.

▶ **Theorem 4.3.** *The elimination sequence of a set of n axis-aligned cubes in fixed dimension $d = O(1)$ can be computed in $O(n \log^{d+2} n)$ time.*

5 Quadtree-based approach

Let Φ denote the *spread* of the disk centers and Δ denote the ratio of the growth rates, i.e., $\Phi = \max_{1 \le i < j \le n} |p_i p_j| / \min_{1 \le i < j \le n} |p_i p_j|$ and $\Delta = \max_{i \in \{1,\ldots,n\}} v_i / \min_{j \in \{1,\ldots,n\}} v_j$. We first present an algorithm that runs in $O(n \log \Phi \min\{\log \Phi, \log \Delta\})$ time using a quadtree. Then, we present an improved algorithm that runs in $O(n(\log n + \min\{\log \Phi, \log \Delta\}))$ time using a compressed quadtree. To simplify the notation, we set $\alpha = \min\{\log \Phi, \log \Delta\}$.

5.1 Using an (uncompressed) quadtree

Without loss of generality, all disk centers lie in the unit square $[0,1]^2$, and their diameter is 1. We construct a *quadtree* $\mathcal{Q}$ for the disk centers. It is a rooted tree in which every internal node has four children. Each node ν of $\mathcal{Q}$ has an associated square *cell* $b(\nu)$. To obtain $\mathcal{Q}$, we recursively split the unit square. In each step, the current node is partitioned into four congruent quadrants (cells) if its corresponding cell contains one or more disk centers. We stop when each cell at the bottom level contains at most one disk center and the diameter of the cell becomes smaller than a quarter of the smallest distance between disk centers. This takes $O(n \log \Phi)$ time as the depth of the quadtree is $O(\log \Phi)$. See Figure 2 (left) for an illustration.

For a node $\nu \in \mathcal{Q}$, we let $p(\nu)$ be the parent node of ν. We denote by $|\nu|$ the diameter of the cell $b(\nu)$. For two nodes $\nu, \nu' \in \mathcal{Q}$, we write $d(\nu, \nu')$ for the smallest distance between a point in $b(\nu)$ and a point in $b(\nu')$. For a point q and a node $\nu \in \mathcal{Q}$, we write $d(q, \nu)$ for the smallest distance between q and a point in $b(\nu)$. For $t \ge 0$, we let D_i^t be the disk D_i at time t. We say that D_i^t *occupies* a node ν if (i) $p_i \in b(\nu)$; (ii) ν is a leaf or $b(\nu) \subseteq D_i^t$; and (iii) D_i^t

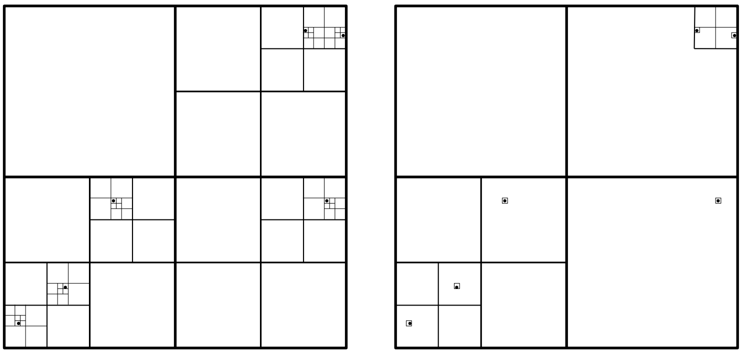


Figure 2 Obtaining a quadtree and its compressed quadtree: (left) a quadtree for 6 disk centers, where the subdivision process stops once a cell contains at most one disk center and the diameter of the cell becomes smaller than a quarter of the smallest distance between disk centers.; (right) the compressed quadtree obtained after eliminating the maximal singular paths.

has not been eliminated before time t. At each moment, each node ν is occupied by at most one disk, and we denote by $D(\nu)$ the index of the disk that occupies ν. If there is no such disk, we set $D(\nu) = \perp$. We denote by $\nu(i, t)$ the node of the largest cell of $\mathcal{Q}$ that is occupied by D_i^t.

▶ **Lemma 5.1.** *Let $i \in \{2, \ldots, n\}$, and let $D_j (j \in \{1, \ldots, i-1\})$ be the disk that eliminates D_i, i.e., $t_i = t(i, j)$. Then,*

$$d\left(\nu(i,t_i), \nu(j,t_i)\right) \le 2\left(|\nu(i,t_i)| + |\nu(j,t_i)|\right),$$

and

$$1/(4\Delta) \le |\nu(i,t_i)| \,/\, |\nu(j,t_i)| \le 4\Delta.$$

Proof. We note three simple facts from the construction of $\mathcal{Q}$ and the definition of $\nu(\cdot,\cdot)$: (i) all non-empty leaf cells have the same diameter; (ii) for any $k \in \{1, \ldots, n\}$ and $t > 0$, if $\nu(k,t)$ is not a leaf, then $|\nu(k,t)| \le 2v_k t$; and (iii) for any $k \in \{1, \ldots, n\}$ and $t \ge 0$, we have $|\nu(k,t)| \ge v_k t/2$.

For the first claim, let $q = \partial D_i^{t_i} \cap \partial D_j^{t_i}$. By fact (iii), we have $v_i t_i \le 2\,|\nu(i,t_i)|$ and $v_j t_i \le 2\,|\nu(j,t_i)|$. Hence, it follows that $d\left(\nu(i,t_i), \nu(j,t_i)\right) \le d\left(q, \nu(i,t_i)\right) + d\left(q, \nu(j,t_i)\right) \le v_i t_i + v_j t_i \le 2\,|\nu(i,t_i)| + 2\,|\nu(j,t_i)|$.

Now we prove the second claim. Suppose first that $v_i \ge v_j$. If $\nu(j,t_i)$ is a leaf, $|\nu(i,t_i)| \,/\, |\nu(j,t_i)| \ge 1$, by fact (i). If $\nu(j,t_i)$ is not a leaf, it follows from facts (ii) and (iii) that

$$\frac{|\nu(i,t_i)|}{|\nu(j,t_i)|} \ge \frac{v_i t_i/2}{2v_j t_i} \ge \frac{1}{4} \ge \frac{1}{4\Delta}.$$

By construction, the leaf cell that contains p_i has diameter smaller than a quarter of the smallest distance between disk centers. Hence, the node $\nu(i,t_i)$ is not a leaf. Thus, by facts (ii) and (iii),

$$\frac{|\nu(i,t_i)|}{|\nu(j,t_i)|} \le \frac{2v_i t_i}{v_j t_i/2} \le \frac{4 \max_i v_i}{\min_j v_j} \le 4\Delta.$$

The argument for $v_j > v_i$ is analogous: if $\nu(i,t_i)$ is a leaf, then $|\nu(j,t_i)| \,/\, |\nu(i,t_i)| \ge 1$, by fact (i). If not, then

$$\frac{|\nu(j,t_i)|}{|\nu(i,t_i)|} \ge \frac{v_j t_i/2}{2v_i t_i} > \frac{1}{4} \ge \frac{1}{4\Delta},$$

Algorithm 2 Quadtree based algorithm

```
function ELIMINATIONORDER($p_1, \dots, p_n, v_1, \dots, v_n$)
    $\mathcal{Q} \leftarrow$ ConstuctQuadTree($p_1, \dots, p_n$)
    CandidatePairs($\mathcal{Q}$)
    $D(\nu) \leftarrow \perp$ for every node $\nu$ of $\mathcal{Q}$
    $D(\text{root}) \leftarrow 1$
    for $i \leftarrow 1, n$ do
        $\nu \leftarrow$ getLeaf($p_i$)
        $t_i \leftarrow \infty$
        while $\nu \neq$ root and $t_i \geq \tau(\nu, i)$ do
            $D(\nu) \leftarrow i$
            for $(\nu, \nu')$ in CNP($\nu$) do
                if $D(\nu') \neq \perp$ and $t_{D(\nu')}, t_i \geq t(i, D(\nu'))$ then
                    $t_i \leftarrow t(i, D(\nu'))$
            $\nu \leftarrow p(\nu)$
    $S \leftarrow (D_1, \dots, D_n)$
    Sort $S$ using key $t_i$ for each disk $D_i$
    return $S$
```

by facts (ii) and (iii). Now, the node $\nu(j, t_i)$ cannot be a leaf, so by facts (ii) and (iii)

$$\frac{|\nu(j,t_i)|}{|\nu(i,t_i)|} \leq \frac{2v_j t_i}{v_i t_i / 2} \leq \frac{4 \max_j v_j}{\min_i v_i} \leq 4\Delta.$$

The lemma follows. ◀

Lemma 5.1 implies that instead of checking all disk pairs for elimination events, we can restrict ourselves to the nodes given by $\mathcal{Q}$. We say that two unrelated[1] nodes $\nu, \nu' \in \mathcal{Q}$ form a *candidate pair* if (i) $|\nu|/4\Delta \leq |\nu'| \leq 4\Delta|\nu|$ and (ii) $d(\nu, \nu') \leq 2(|\nu| + |\nu'|)$. In this case, we say that ν *forms the candidate pair* (ν, ν') with ν'. We denote by $\mathsf{CNP}(\nu)$ the set of candidate pairs formed by ν.

▶ **Lemma 5.2.** *Let $\nu \in \mathcal{Q}$. Then, $\mathsf{CNP}(\nu)$ has $O(\alpha)$ candidate pairs (ν, ν') with $|\nu| \leq |\nu'|$. All the sets $\mathsf{CNP}(\nu)$ over $\nu \in \mathcal{Q}$ can be computed in $O(n\alpha \log \Phi)$ time.*

Proof. Using a packing argument we can show that each level of $\mathcal{Q}$ contains at most $O(1)$ candidate pairs (ν, ν') that satisfy $|\nu| \leq |\nu'|$. Furthermore, by definition of Φ and of candidate pair, $|\nu'| = O(\min\{\Phi, \Delta\})|\nu|$, which implies that the levels of ν and ν' in $\mathcal{Q}$ differ by $O(\alpha)$. This implies that globally $\mathsf{CNP}(\nu)$ contains $O(\alpha)$ candidate pairs (ν, ν') with $|\nu| \leq |\nu'|$. Since $\mathcal{Q}$ has $O(n \log \Phi)$ nodes, and since $(\nu, \nu') \in \mathsf{CNP}(\nu)$ if and only if $(\nu', \nu) \in \mathsf{CNP}(\nu')$, there are $O(n\alpha \log \Phi)$ candidate pairs overall. While building $\mathcal{Q}$, we can find all sets $\mathsf{CNP}(\nu)$ in $O(n\alpha \log \Phi)$ time by maintaining pointers between nodes whose cells are neighboring and by traversing the cells, using these pointers when needed. ◀

Our algorithm for computing the elimination sequence of the input disks is given as Algorithm 2. We use $\tau(\nu, i)$ for the first time at which $b(\nu)$ is covered by disk D_i.

[1] That is, no node is an ancestor or descendant of the other node.

▶ **Theorem 5.3.** *The elimination sequence of n growing disks can be computed in $O(n\alpha\log\Phi)$ time, where $\alpha = \min\{\log\Phi, \log\Delta\}$.*

Proof. We can compute in $O(n\log\Phi)$ time the quadtree $\mathcal{Q}$ with $O(n\log\Phi)$ nodes. By Lemma 5.2, there are $O(n\alpha\log\Phi)$ candidate pairs, which can be found in $O(n\alpha\log\Phi)$ time.

The outer **for**-loop iterates over the input disks in decreasing order of priority. In the **while**-loop, the algorithm traverses each node $\nu \in \mathcal{Q}$ from the leaf-node containing p_i to the root. It updates $D(\nu)$ if necessary until it encounters a node ν with $t_i < \tau(\nu, i)$. The inner **for**-loop iterates over every candidate pair (ν, ν') in $\mathsf{CNP}(\nu)$. It checks if disk $i = D_\nu$ and D'_ν have the possibility to touch by computing the time $t(i, D(\nu'))$; if so, it updates the elimination time for D_i. Thus, the algorithm takes $O(n\alpha\log\Phi)$ time. Since $\Phi = \Omega(\sqrt{n})$, this subsumes the time for the sorting step.[2] ◀

5.2 Using a compressed quadtree

Now we show how to improve the running time by using a *compressed quadtree*. Let $\mathcal{Q}$ be the (usual) quadtree for the n disk centers. The tree $\mathcal{Q}$ is obtained as in the previous section. We describe how to obtain the compressed quadtree $\mathcal{Q}_C$ from $\mathcal{Q}$. A node ν in $\mathcal{Q}$ is *empty* if $b(\nu)$ does not contain a disk-center, and *non-empty* otherwise. A *singular path* σ in $\mathcal{Q}$ is a path $\nu_1, \nu_2, \dots, \nu_k$ of nodes such that (i) ν_k is a non-empty leaf or has at least two non-empty children; and (ii) for $i = 1, \dots, k-1$, the node ν_{i+1} is the only non-empty child of ν_i. We call σ *maximal* if it cannot be extended by the parent of ν_1 (either because ν_1 is the root or because $p(\nu_1)$ has two non-empty children). For each maximal singular path $\sigma = \nu_1, \dots, \nu_k$ in $\mathcal{Q}$, we remove from $\mathcal{Q}$ all proper descendants of ν_1 that are not descendants of ν_k, together with their incident edges. Then, we add a new *compressed edge* between ν_1 and ν_k. The resulting tree $\mathcal{Q}_C$ has $O(n)$ nodes. Each internal node has 1 or 4 children. There are algorithms that can compute $\mathcal{Q}_C$ in $O(n\log n)$ time [9]. A node ν from $\mathcal{Q}$ may appear as a node in $\mathcal{Q}_C$ or not. We let $\pi(\nu)$ be the lowest ancestor node and $\sigma(\nu)$ the highest descendant node (in both cases including ν) of ν in $\mathcal{Q}$ that appears also in $\mathcal{Q}_C$. See Figure 2 (right) for an illustration. For a node ν in $\mathcal{Q}_C$, we define the set of *compressed candidate pairs* $\mathsf{CNP}_C(\nu)$ for ν as

$$\mathsf{CNP}_C(\nu) = \{(\nu, \pi(\nu')) \mid (\nu, \nu') \in \mathsf{CNP}(\nu), |\nu| \le |\pi(\nu')|\}.$$

For a pair $(\nu, \nu') \in \mathsf{CNP}_C(\nu)$, we say ν *forms the candidate pair* with ν' in $\mathcal{Q}_C$. The following lemmas will be handy for the rest of the section.

▶ **Lemma 5.4.** *Let $(\nu, \nu') \in \mathsf{CNP}(\nu)$, such that $p(\nu) \ne p(\nu')$. Then, (i) we have $(p(\nu), p(\nu')) \in \mathsf{CNP}(p(\nu))$. Moreover, (ii) if $|\nu| \le |\nu'|$, then $(\nu'', \nu') \in \mathsf{CNP}(\nu'')$ for any ancestor ν'' of ν with $|\nu''| \le |\nu'|$.*

Proof. For the first part (i), we have $d(p(\nu), p(\nu')) \le d(\nu, \nu') \le 2(|\nu|+|\nu'|) \le 2(|p(\nu)|+p(|\nu'|))$ and $|p(\nu')|/|p(\nu)| = |\nu'|/|\nu|$ lies between $1/4\Delta$ and 4Δ.

For the second part (ii), we have $d(\nu'', \nu') \le d(\nu, \nu') \le 2(|\nu| + |\nu'|) \le 2(|\nu''| + |\nu'|)$ and $1 \le |\nu'|/|\nu''| \le |\nu'|/|\nu| \le 4\Delta$. ◀

▶ **Lemma 5.5.** *Let ν be a node of $\mathcal{Q}$. Then, for every $(\nu, \nu') \in \mathsf{CNP}(\nu)$, we have that $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$ or $(\pi(\nu'), \pi(\nu)) \in \mathsf{CNP}_C(\pi(\nu'))$.*

[2] A packing argument shows that the spread of any d-dimensionsonal n-point set is $\Omega(n^{1/d})$: if any two points have distance at least 1, the point set must cover at least $\Omega(n)$ units of volume and hence must have diameter $\Omega(n^{1/d})$.

Proof. First, we note that $\pi(\nu)$ and $\pi(\nu')$ are distinct, since ν and ν' are unrelated nodes in $\mathcal{Q}$, so their least common ancestor in $\mathcal{Q}$ must have two non-empty children. Since the lemma is symmetric in ν and ν', we may assume without loss of generality that $|\pi(\nu)| \le |\pi(\nu')|$. We apply Lemma 5.4(i) repeatedly until we meet $\pi(\nu)$ or $\pi(\nu')$, whichever happens first. If we meet $\pi(\nu)$, we have $(\pi(\nu), \nu'') \in \mathsf{CNP}(\pi(\nu))$ for some ancestor ν'' of ν' in $\mathcal{Q}$. Since $\pi(\nu)$ is encountered first, we have $\pi(\nu'') = \pi(\nu')$, so it follows that $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$. If we meet $\pi(\nu')$, we have $(\nu'', \pi(\nu')) \in \mathsf{CNP}(\nu'')$ for some ancestor ν'' of ν. Since $|\pi(\nu)| \le |\pi(\nu')|$ and again $\pi(\nu'') = \pi(\nu)$, it follows that $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}(\pi(\nu))$ by Lemma 5.4(ii), and thus $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$. ◀

As with Lemma 5.2, we argue that $\mathsf{CNP}_C(\nu)$ has $O(\alpha)$ candidate pairs. To that end, we charge each pair $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$ to a pair $(\nu, \nu'') \in \mathsf{CNP}(\nu)$ with $|\nu| \le |\nu''|$, such that each such pair in $\mathsf{CNP}(\nu)$ is charged at most once. First, if $|\nu| \le |\nu'|$, we can charge $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$ directly to $(\nu, \nu') \in \mathsf{CNP}(\nu)$ (in this way, we may even charge several such pairs in $\mathsf{CNP}(\nu)$ for $(\nu, \pi(\nu'))$). Second, if $|\nu'| < |\nu|$, by Lemma 5.4(ii) there is an ancestor ν'' of ν' with $|\nu| = |\nu''|$ and $(\nu, \nu'') \in \mathsf{CNP}(\nu)$. Furthermore, since by definition of $\mathsf{CNP}_C(\nu)$ we have $|\nu| \le |\pi(\nu')|$, it follows that $\pi(\nu'') = \pi(\nu')$, so we can charge the pair $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$ to the pair $(\nu, \nu'') \in \mathsf{CNP}(\nu)$. It follows that there are $O(n\alpha)$ compressed candidate pairs in total. The following lemma shows how to compute $\mathsf{CNP}_C(\nu)$ for all nodes ν in $\mathcal{Q}_C$.

▶ **Lemma 5.6.** *We can compute all the sets* $\mathsf{CNP}_C(\nu)$ *over* $\nu \in \mathcal{Q}_C$ *in* $O(n\alpha)$ *total time.*

Proof. We traverse the nodes in $\mathcal{Q}_C$ from the root in BFS-fashion, ordered by decreasing diameter. We compute $\mathsf{CNP}_C(\nu)$ for each node ν in order. For a node ν in $\mathcal{Q}_C$, we put into $\mathsf{CNP}_C(\nu)$ all pairs $(\nu, \nu') \in \mathsf{CNP}(\nu)$ with $\nu' \in \mathcal{Q}_C$ and $|\nu| = |\nu'|$. Furthermore, we check all pairs (ν, ν') with $|\nu| < |\nu'|$ and (a) $(p(\nu), \nu') \in \mathsf{CNP}_C(p(\nu))$ or (b) $(\nu', p(\nu)) \in \mathsf{CNP}_C(\nu')$. We add (ν, ν') to $\mathsf{CNP}_C(\nu)$ if (ν, ν') fulfills the requirements of a compressed candidate pair. This can be checked in $O(1)$ time. By our BFS-traversal, we already know the sets $\mathsf{CNP}_C(p(\nu))$ and $\mathsf{CNP}_C(\nu')$ for $|\nu| < |\nu'|$.

For $|\nu| = |\nu'|$, there are $O(1)$ pairs to check, and they can be found at the same time using appropriate pointers in $\mathcal{Q}_C$. For $|\nu| < |\nu'|$, since $|\mathsf{CNP}_C(p(\nu))| = O(\alpha)$, there are $O(\alpha)$ pairs to check for case (a). There can be $\omega(\alpha)$ pairs for case (b), but obviously there are $O(n\alpha)$ such pairs in total for all $\nu \in \mathcal{Q}_C$.

Now we show that the algorithm correctly computes all the compressed candidate pairs in $\mathsf{CNP}_C(\nu)$. Consider a pair $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$, where $(\nu, \nu') \in \mathsf{CNP}(\nu)$ and $|\nu| \le |\pi(\nu')|$. If $|\nu| = |\pi(\nu')|$, we have $(\nu, \pi(\nu')) \in \mathsf{CNP}(\nu)$ so the algorithm will find it. If $|\nu| < |\pi(\nu')|$, let η be the parent of ν *in* $\mathcal{Q}$. If $\pi(\nu') = \nu'$, we have $(\eta, \pi(\nu')) \in \mathsf{CNP}(\eta)$ by Lemma 5.4(ii), since $|\eta| \le |\pi(\nu')|$. If $|\pi(\nu')| > |\nu'|$, let η' be the parent of ν' *in* $\mathcal{Q}$. Lemma 5.4(i) implies $(\eta, \eta') \in \mathsf{CNP}(\eta)$. Since $\pi(\eta) = p(\nu)$ (as a node in $\mathcal{Q}_C$ this time) and $\pi(\eta') = \pi(\nu')$, we conclude with Lemma 5.5 that $(p(\nu), \pi(\nu')) \in \mathsf{CNP}_C(p(\nu))$ or $(\pi(\nu'), p(\nu)) \in \mathsf{CNP}_C(\pi(\nu'))$. ◀

Recall that, in the uncompressed quadtree approach each candidate pair (of nodes) leads to a pair of disks that may touch at some time. We will call such a pair a *candidate pair of disks*. Note that two distinct candidate pairs may be associated to the same candidate pair of disks. Let $\mathcal{D}$ be the set of all candidate pairs of disks obtained using the uncompressed quadtree approach.

We set $D_C(\nu)$ to $D(\nu)$, if $D(\nu) \ne \perp$. If $D(\nu) = \perp$ and ν has a single child ν' connected by a compressed edge, we set $D_C(\nu) = D(\nu')$. In all other cases, we set $D_C(\nu) = \perp$. A compressed candidate pair (ν, ν') for $\nu, \nu' \in \mathcal{Q}_C$ defines a candidate pair of disks $(D_C(\nu), D_C(\nu'))$ if both $D_C(\nu), D_C(\nu') \neq \perp$. We let $\mathcal{D}_C$ denote the set of all candidate pairs of disks defined by compressed candidate pairs. We claim that $\mathcal{D} \subseteq \mathcal{D}_C$. That is, even though the compressed quadtree has fewer candidate pairs of nodes, we discard only candidates that are already in $\mathcal{D}_C$. We first introduce a helpful lemma.

▶ **Lemma 5.7.** *Let $\nu \in \mathcal{Q}$, and consider the nodes $\sigma(\nu)$ and $\pi(\nu)$ in $\mathcal{Q}_C$. If $\pi(\nu)\sigma(\nu)$ is a compressed edge, then for any node $\nu' \in \mathcal{Q}$ on the singular path for $\pi(\nu)\sigma(\nu)$, we have $D(\nu') \in \{D(\sigma(\nu)), \perp\}$.*

Proof. Recall that, for any node $\eta \in \mathcal{Q}$, we have $D(\eta) = i$ if and only if D_i occupies η and $b(\eta)$ contains p_i. Since each node ν' on the singular path has only one non-empty child, the only disk that can occupy ν' is $D(\sigma(\nu))$. ◀

▶ **Lemma 5.8.** $\mathcal{D} \subseteq \mathcal{D}_C$.

Proof. Let $(D(\nu), D(\nu')) \in \mathcal{D}$. If $\nu \in \mathcal{Q}_C$, $\pi(\nu) = \nu$ and $D_C(\pi(\nu)) = D(\nu)$. If $\nu \notin \mathcal{Q}_C$, then if $D(\pi(\nu)) \neq \perp$, by Lemma 5.7, $D(\pi(\nu)) = D(\sigma(\nu))$ and hence $D_C(\pi(\nu)) = D(\sigma(\nu))$. If $D(\pi(\nu)) = \perp$, then the child node of $\pi(\nu)$ in $\mathcal{Q}_C$ is $\sigma(\nu)$, and therefore $D_C(\pi(\nu)) = D(\sigma(\nu))$. Thus, in both cases, we have $D_C(\pi(\nu)) = D(\sigma(\nu))$. Since $D(\nu) \neq \perp$, we have $D(\nu) = D(\sigma(\nu))$ by Lemma 5.7, so $D_C(\pi(\nu)) = D(\nu)$. The same holds for ν'. Finally, $(\nu, \nu') \in \mathsf{CNP}(\nu)$ implies tha $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$ or $(\pi(\nu'), \pi(\nu)) \in \mathsf{CNP}_C(\pi(\nu'))$ by Lemma 5.5. We conclude that $(D(\nu), D(\nu')) = (D_C(\pi(\nu)), D_C(\pi(\nu')) \in \mathcal{D}_C$. ◀

▶ **Theorem 5.9.** *The elimination sequence of n disks can be computed in $O(n \log n + n\alpha)$ time, where $\alpha = \min\{\log \Phi, \log \Delta\}$.*

Proof. We compute the compressed quadtree for the disk centers, and we find the compressed candidate pairs. As described above, this takes $O(n \log n + n\alpha)$ time. After that, we make the candidate pairs symmetric so that for all pairs ν, ν', we have $(\nu, \nu') \in \mathsf{CNP}_C(\nu)$ if and only if $(\nu', \nu) \in \mathsf{CNP}_C(\nu')$. This takes $O(n\alpha)$ time. Finally, we proceed as in Algorithm 2, but using $\mathcal{Q}_C$ instead of $\mathcal{Q}$ and the compressed candidate pairs instead of the (regular) candidate pairs. By Lemma 5.8, this algorithm still considers all the relevant candidate pairs of disks. The running time for the last step is proportional to the number of nodes in $\mathcal{Q}_C$ and the number of compressed candidates, i.e., $O(n\alpha)$. The total running time of the algorithm is $O(n \log n + n\alpha)$. ◀

6 Lower bound

We show that the elimination order can be used to sort n numbers $v_{n+1}, \ldots, v_{2n}$ larger than 1 and smaller than 2, which implies an $\Omega(n \log n)$ lower bound. Place n growing disks $D_1, \ldots, D_n$ centered at points $(2,0), (4,0), \ldots (2n, 0)$, all with growth rate $v_i = 1$. Also, place n disks $D_{n+1}, \ldots, D_{2n}$ centered at points $(2,1), (4,1), \ldots (2n, 1)$ with growth rates $v_{n+1}, \ldots, v_{2n}$. Observe that disk D_{n+i} will be eliminated by disk D_i at $t_{n+i} = t(n+i, i) = 1/(1 + v_{n+i}) < 1/2$ since $t_i = 1/2$ for $1 \leq i \leq n$. Then the elimination order of this set of growing disks gives the input growth rates $\{v_{n+1}, \ldots, v_{2n}\}$ in reversed sorted order. The same argument holds for squares.

▶ **Theorem 6.1.** *It takes at least $\Omega(n \log n)$ time to find the elimination order of a set of n growing disks or squares in the plane under the algebraic decision tree model.*

Acknowledgments. This work was initiated during the 20th Korean Workshop on Computational Geometry. The authors would like to thank the other participants for motivating discussions.

References

1 Pankaj K. Agarwal, Boris Aronov, and Micha Sharir. Computing envelopes in four dimensions with applications. *SIAM J. Comput.*, 26(6):1714–1732, 1997.

2 Daniel Bahrdt, Michael Becher, Stefan Funke, Filip Krumpe, André Nusser, Martin Seybold, and Sabine Storandt. Growing balls in $\mathbb{R}^d$. In *Proc. 19th Workshop Algorithm Eng. Exp. (ALENEX)*, pages 247–258, 2017.

3 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 2008.

4 Thom Castermans, Bettina Speckmann, Frank Staals, and Kevin Verbeek. Agglomerative clustering of growing squares. *CoRR*, abs/1706.10195, 2017. URL: `http://arxiv.org/abs/1706.10195`.

5 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

6 Michael Formann. Weighted closest pairs. In *Proc. 10th Sympos. Theoret. Aspects Comput. Sci. (STACS)*, pages 270–281, 1993.

7 Stefan Funke, Filip Krumpe, and Sabine Storandt. Crushing disks efficiently. In *Proc. 27th Int. Workshop Comb. Alg. (IWOCA)*, pages 43–54, 2016.

8 Stefan Funke and Sabine Storandt. Parametrized runtimes for ball tournaments. In *Proc. 33rd European Workshop Comput. Geom. (EWCG)*, pages 221–224, 2017.

9 Sariel Har-Peled. *Geometric Approximation Algorithms.* American Mathematical Society, Boston, MA, USA, 2011.

10 John Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33(4):169–174, 1989.

11 Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004.

12 Jiří Matoušek. *Lectures on Discrete Geometry.* Springer-Verlag, 2002.

Placing your Coins on a Shelf[*]

Helmut Alt[1], Kevin Buchin[2], Steven Chaplick[3], Otfried Cheong[†4], Philipp Kindermann[5], Christian Knauer[6], and Fabian Stehn[7]

1 **Freie Universität Berlin, Germany**
`alt@fu-berlin.de`

2 **Technische Universiteit Eindhoven, Netherlands**
`k.a.buchin@tue.nl`

3 **Universität Würzburg, Germany**
`steven.chaplick@uni-wuerzburg.de`

4 **KAIST, Daejeon, Republic of Korea**
`otfried@kaist.airpost.net`

5 **FernUniversität in Hagen, Germany**
`philipp.kindermann@fernuni-hagen.de`

6 **Universität Bayreuth, Germany**
`christian.knauer@uni-bayreuth.de`

7 **Universität Bayreuth, Germany**
`fabian.stehn@uni-bayreuth.de`

Abstract

We consider the problem of packing a family of disks "on a shelf," that is, such that each disk touches the x-axis from above and such that no two disks overlap. We prove that the problem of minimizing the distance between the leftmost point and the rightmost point of any disk is NP-hard. On the positive side, we show how to approximate this problem within a factor of $4/3$ in $O(n \log n)$ time, and provide an $O(n \log n)$-time exact algorithm for a special case, in particular when the ratio between the largest and smallest radius is at most four.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases packing problems, approximation algorithms, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.4

1 Introduction

Packing problems have a long history and abundant literature. Circular disks and spherical balls, because of their symmetry and simplicity, are of particular interest from a theoretical point of view. Historically, Johannes Kepler conjectured that an optimal packing of unit spheres into the Euclidean three-space cannot have greater density than the face-centered cubic packing [8]. The conjecture was first proven to be correct by Hales and Ferguson [7]. A more recent treatment of the proof is given by Hales et al. [6]. The proof of the 2-dimensional version of Kepler's conjecture, that is, packing unit disks into the Euclidean two-space, is elementary and attributed to Lagrange (1773).

Packing unit disks into 2-dimensional shapes in the plane is a well studied problem in recreational mathematics. Croft et al. [2] give an overview of packing geometrical objects in finite-sized containers, for instance finding the smallest square (circle, isosceles triangle, etc.)

[*] A full version of this result is available on the ArXiv [1], `https://arxiv.org/abs/1707.01239`.

[†] OC is supported by NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 4; pp. 4:1–4:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

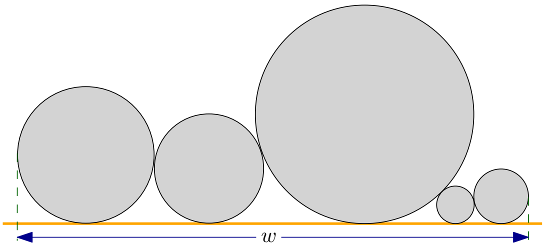


Figure 1 Illustration of the span w of a valid (but not optimal) placement of five discs.

such that a given number of n unit disks can be packed into it. Specht [10] presents the best known packings of up to $10,000$ disks into various containers.

Algorithmically, many packing problems are NP-hard, some are not even known to be in NP. Demaine, Fekete, and Lang showed that the problems whether a given set of circular disks of arbitrary radii can be packed into a given square, rectangle, or triangle are all NP-hard problems [3].

We will discuss a particular "nearly" one-dimensional packing problem for disks from an algorithmic aspective. We are given a family of disks that we wish to arrange "on a shelf," that is, such that each disk touches the x-axis from above and such that no two disks overlap; see Figure 1. The goal is to minimize the *span* of the resulting configuration, that is, to minimize the horizontal distance between the leftmost point and the rightmost point of any disk. In other words, we want to minimize the required width of the shelf. Obviously, this problem is trivial for unit disks, so we allow the disks to have different sizes.

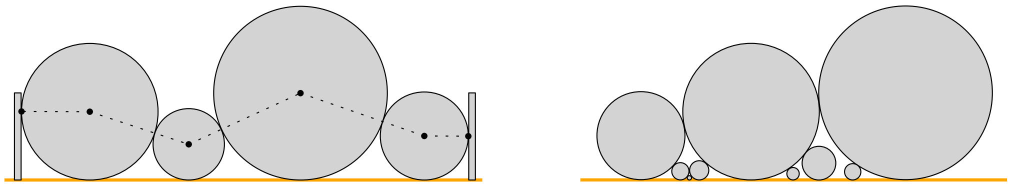
Related work. Independently from us, Dürr et al. [4] have studied the same problem, but for an isosceles, right-angled triangle. Given n sizes of this triangle, they ask for the shortest horizontal span in which the triangles can be arranged so that their lowest point lies on the x-axis, while the triangles do not overlap. Their entirely independent results are quite similar to ours: an NP-hardness proof by reduction from 3-Partition, a fast algorithm for a special case, and a $^3/_2$-approximation algorithm.

Klemz et al. [9] show that it is NP-hard to decide if n given disks fit around a large center disk, such that each disk is in contact with the center disk while all disks are disjoint. Their proof is by reduction from 3-Partition as well.

Stoyan and Yaskov [11] introduce the problem of packing disks of unequal sizes into a strip of given height and minimizing the required width which is known as the *circular open dimension problem.*

Our results. We first give some useful definitions and properties for touching disks in Section 2. The hardness of the problem arises from the fact that disks can sometimes "hide" in the holes formed by larger disks, as in Figure 2b. For this reason, in Section 3, we consider the special case where, for any ordering of the disks, each disk can touch only its left and its right neighbor (where the two walls bounding the span count as neighbors as well). In particular, this implies that no disk will ever fit in a gap between two other disks. We call this the *linear case*, see Figure 2a. It turns out that for this (linear) case the optimal configuration depends *only* on the relative order of the disk sizes,[1] so it suffices to sort the disks in $O(n \log n)$ time to determine the optimal sequence.

[1] The median disk for an odd number of disks is the only exception, it can be on either end, depending on its actual size.



(a) The linear case.

(b) Small disks can "hide" between larger disks.

Figure 2 Illustration of different instances of the problem.

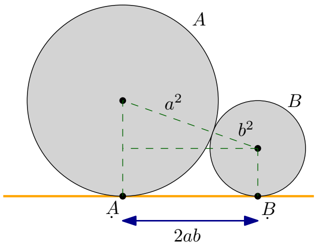


Figure 3 The footpoint distance of two touching disks.

In Section 4, we show that in its general form, the problem is NP-hard. More precisely, we show that given n disk sizes and a number $\delta > 0$, it is NP-hard to decide if a non-overlapping arrangement of the disks with horizontal span at most δ exists. Our NP-hardness proof is by a reduction from 3-Partition, and exploits the fact that disks can "hide" in the holes formed by larger disks.

Finally, in Section 5, we give an approximation algorithm that runs in $O(n \log n)$ time and guarantees a span at most $4/3$ times the optimal span.

2 Preliminaries

For reasons that will become obvious shortly, it will be convenient to define the *size* of a disk as the *square root* of its radius. We will denote disks by capital letters, and their size by the corresponding lower-case letter. Namely, disk A has size a, radius a^2, and diameter $2a^2$.

In a valid placement, each disk A touches the x-axis in its lowest point. We will call this point the *footpoint* of the disk and denote it $\underset{\cdot}{A}$. All of our arguments are based on calculations involving the distances between footpoints, so we start with the following lemma.

▶ **Lemma 1.** *If A and B touch, then their footpoint distance $\underset{\cdot}{A}\underset{\cdot}{B}$ is $2ab$.*

Proof. The statement holds for $a = b$, so we assume $a > b$ and consider the right-angled triangle with edge lengths $\underset{\cdot}{A}\underset{\cdot}{B}$, $a^2 + b^2$, and $a^2 - b^2$, see Figure 3. We obtain $(\underset{\cdot}{A}\underset{\cdot}{B})^2 = (a^2 + b^2)^2 - (a^2 - b^2)^2 = 4a^2b^2$. ◀

▶ **Lemma 2.** *Let G be the largest disk that fits in the gap formed by two touching disks A and B. Then $1/g = 1/a + 1/b$.*

Proof. Since G is the largest disk that fits in the gap, it must touch both A and B. By Lemma 1 we have $2ab = \underset{\cdot}{A}\underset{\cdot}{B} = \underset{\cdot}{A}\underset{\cdot}{G} + \underset{\cdot}{G}\underset{\cdot}{B} = 2ag + 2gb$, proving the lemma. ◀

▶ **Lemma 3.** *Let G be the largest disk that fits in the gap between a disk A and the vertical wall through A's rightmost point. Then $g = (\sqrt{2} - 1) \cdot a$.*

Proof. Again, G must touch both A and the wall, so we have $a^2 = \underset{\cdot}{A}\underset{\cdot}{G} + g^2 = 2ag + g^2$. The positive solution to $g^2 + 2ag - a^2 = 0$ is $(\sqrt{2} - 1) \cdot a$. ◀

In any valid placement of the disks, their footpoints are distinct. Thus, the footpoints induce a linear left-to-right order on the disks. We refer to this linear order as the *footpoint sequence* of a valid placement. Further, disks are called *consecutive* or *neighbors* when their footpoints are consecutive in the footpoint sequence.

3 The Linear Case

In this section, we consider *linear case instances*, that is, instances where in any valid placement only consecutive pairs of disks can touch, only the first disk (with the leftmost footpoint) touches the left wall, and only the last disk touches the right wall.

By Lemmas 2 and 3, this is true if and only if the following condition holds: Let A be the largest disk, B the second largest, and Z the smallest disk in the collection. Then $1/z < 1/a + 1/b$, and $z > (\sqrt{2} - 1)a$. The condition holds in particular if the ratio between the largest and smallest disk size is less than two (that is, if the ratio of diameters is less than four), since then we have $1/z < 2/a \leqslant 1/a + 1/b$ and $z > a/2 > (\sqrt{2} - 1)a$.

In an optimal placement of a linear case instance, each disk must touch both its neighbors. Thus, the ordering of the disks uniquely determines the exact placement of every disk in any layout of minimal span. From now on, we represent placements by the *ordering* of the disks, with the understanding that the placement minimizes the span for this ordering. It remains to determine the optimal ordering. We will first give a lemma that allows us to improve a given ordering.

▶ **Lemma 4.** *Let $\mathcal{D}$ be a left-to-right or right-to-left ordering of the disks in a linear case instance. Let A, B, Z be three disks that appear in this order in $\mathcal{D}$ such that AB is a consecutive pair. Let $\mathcal{D}'$ be the ordering obtained from $\mathcal{D}$ by reversing the subsequence from B to Z. Then $\mathcal{D}'$ has smaller span than $\mathcal{D}$ if one of the following is true:*

1. *Z is the last disk and $a > b > z$;*
2. *Z is the last disk and $a < b < z$;*
3. *$a > y$ and $b > z$, where Y is the disk after Z in $\mathcal{D}$;*
4. *$a < y$ and $b < z$, where Y is the disk after Z in $\mathcal{D}$.*

Proof. First, suppose that Z is the last disk in $\mathcal{D}$. Then, except for $\underset{\cdot}{A}\underset{\cdot}{B}$ being replaced by $\underset{\cdot}{A}\underset{\cdot}{Z}$, each consecutive footpoint distance in $\mathcal{D}'$ is the same as in $\mathcal{D}$. So, since the last disk in $\mathcal{D}'$ is B, the change in span is $\underset{\cdot}{A}\underset{\cdot}{Z} + b^2 - \underset{\cdot}{A}\underset{\cdot}{B} - z^2 = 2az + b^2 - 2ab - z^2 = (b + z - 2a)(b - z)$. For both $a < b < z$ and $a > b > z$, this is negative, and so $\mathcal{D}'$ has smaller span than $\mathcal{D}$.

Now suppose Z is not the last disk, and let Y be the disk after Z. Here, except for $\underset{\cdot}{A}\underset{\cdot}{B}$ being replaced by $\underset{\cdot}{A}\underset{\cdot}{Z}$ and $\underset{\cdot}{Z}\underset{\cdot}{Y}$ being replaced by $\underset{\cdot}{B}\underset{\cdot}{Y}$, each consecutive footpoint distance in $\mathcal{D}'$ is the same as in $\mathcal{D}$. Thus, the change in span is $\underset{\cdot}{A}\underset{\cdot}{Z} + \underset{\cdot}{B}\underset{\cdot}{Y} - \underset{\cdot}{A}\underset{\cdot}{B} - \underset{\cdot}{Z}\underset{\cdot}{Y} = 2(az + by - ab - zy) = 2(a - y)(z - b)$. For $a > y$ and $b > z$ or $a < y$ and $b < z$, this is negative. So, again $\mathcal{D}'$ has smaller span than $\mathcal{D}$. ◀

We label a given family of n disks in order of decreasing size as $D_1, D_2, D_3, \dots, D_n$, and in order of increasing size as $S_1, S_2, S_3, \dots, S_n$. In other words, $d_1 \geqslant d_2 \geqslant d_3 \geqslant \cdots \geqslant d_n$ and $s_1 \leqslant s_2 \leqslant s_3 \leqslant \cdots \leqslant s_n$. Thus, each disk has two names, and we have $D_1 = S_n$, $D_2 = S_{n-1}$, and so on until $D_n = S_1$.

We now prove our claim about the structure of the optimal ordering (see also Figure 4):

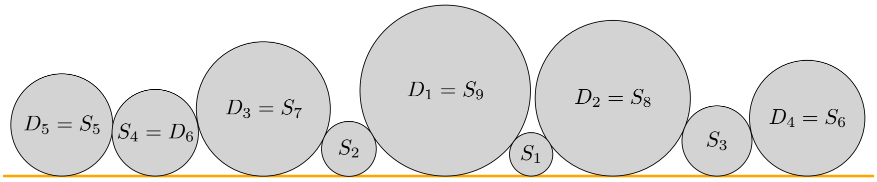


Figure 4 An optimal placement in the linear case. For instance for $k = 2$, the disks in $\{S_1, S_2, D_1, D_2\}$ form the consecutive subsequence starting with S_2 and ending with D_2.

▶ **Lemma 5.** *Let k be an integer with $1 \leqslant k \leqslant n/2$. In any optimal placement of n disks with distinct sizes in a linear case instance, the k largest disks $D_1, \dots, D_k$ and the k smallest disks $S_1, \dots, S_k$ appear as a consecutive subsequence terminated by the disks S_k and D_k. If $k > 1$, then $D_k S_{k-1}$ and $S_k D_{k-1}$ are consecutive pairs.*

Proof. We use induction over k. For $k = 1$, it suffices to prove that S_1 and D_1 are consecutive, so assume for a contradiction that this is not the case. Let $A = D_1$, $Z = S_1$, assume A is to the left of Z, and let B be the right neighbor of A. By Lemma 4 (Case 1 or 3), the sequence can now be improved by reversing the subsequence from B up to Z.

Assume now that $k > 1$ and that the statement holds for $k - 1$. This means that there is a consecutive subsequence of the disks $\{S_1, \dots, S_{k-1}, D_1, \dots, D_{k-1}\}$, terminated by disk S_{k-1} at the, say, right end and disk D_{k-1} at the left end, as in the example of Figure 4.

We first show that the right neighbor of S_{k-1} is D_k. Assume this is not the case. We distinguish four cases:

1. If D_k appears to the right of S_{k-1} (but not immediately adjacent), then we apply Lemma 4 (Case 2 or 4) with $A = S_{k-1}$, B the right neighbor of S_{k-1}, and $Z = D_k$.
2. If D_k appears to the left of S_{k-1}, then it must appear to the left of D_{k-1}. If D_k is not the left neighbor of D_{k-1}, then apply Lemma 4 (Case 1 or 3) with $A = D_k$, B the right neighbor of D_k, and $Z = S_{k-1}$.
3. If D_k is the left neighbor of D_{k-1} and S_{k-1} is not the rightmost disk, then apply Lemma 4 (Case 3) with $A = D_k$, $B = D_{k-1}$, and $Z = S_{k-1}$.
4. If D_k is the left neighbor of D_{k-1} and S_{k-1} is the rightmost disk, then S_k appears somewhere to the left of D_k. We apply Lemma 4 (Case 1 or 3) with $A = D_{k-1}$, $B = D_k$, and $Z = S_k$.

We next show that the left neighbor of D_{k-1} is S_k. Assume this is not the case. If S_k appears somewhere to the left of D_{k-1}, apply Lemma 4 (Case 1 or 3) with $A = D_{k-1}$, B the left neighbor of D_{k-1}, and $Z = S_k$. If, on the other hand, S_k appears to the right of D_k, apply Lemma 4 (Case 2 or 4) with $A = S_k$, B the left neighbor of S_k, and $Z = D_{k-1}$. (Note that in this case B might be D_k.) ◀

▶ **Theorem 6.** *Let $\mathcal{D}$ be a linear case instance of n disks $D_1, \dots, D_n$ of sizes $d_1 \geqslant d_2 \geqslant \dots \geqslant d_n$. If n is even, then the following ordering is optimal:*

$$\dots, D_{n-5}, D_5, D_{n-3}, D_3, D_{n-1}, D_1, D_n, D_2, D_{n-2}, D_4, D_{n-4}, D_6, D_{n-6} \dots$$

For odd n, the median disk needs to be appended at the end of the sequence with the larger size difference.

Proof. Let $\mathcal{D}$ be in the given ordering, and assume a better ordering $\mathcal{D}'$ exists. We can modify the disk sizes slightly so as to make them unique while keeping $\mathcal{D}'$ better than $\mathcal{D}$. But then we have a contradiction to Lemma 5. If n is odd, then the only possible placements of the median disk are the left end and the right end, so choosing the end with the larger size difference gives the optimal solution. ◀

4 NP-Hardness of the General Case

Let us denote the decision version of our problem as CoinsOnAShelf. Its input is a set of disks with rational radii and a rational number $\delta > 0$, the question is whether there is a feasible placement of the disks with span at most δ.

▶ **Theorem 7.** *CoinsOnAShelf is* NP*-hard, even when the ratio of the largest and smallest disk size is bounded by six and when all numbers are given in unary notation.*

Our proof is by reduction from 3-Partition [5, Problem SP15]. An instance of 3-Partition consists of $3m$ integers $\mathcal{A} = a_1, \ldots, a_{3m}$ and another integer B, with $\sum_{i=1}^{3m} a_i = mB$ and $B/4 < a_i < B/2$ for all i. 3-Partition decides if there is a partition of $\mathcal{A}$ into m three-element groups $A_1, \ldots, A_m$ such that $\sum_{a \in A_i} a = B$ for each group A_i.

Given a 3-Partition instance $(\mathcal{A}, B)$, we construct a family $\mathcal{D}$ of $12m + 11$ disks, as follows:

- $m + 1$ disks of size 1, we will refer to these disks as *outer frame disks*;
- $4(m + 1)$ disks of size $s_0 = 33/100 = 0.33$, we will refer to these disks as *inner frame disks*;
- $2(m + 1)$ disks of size $s_1 = s_0/1+s_0 = 33/133$ (≈ 0.24812), we will refer to these disks as *large filler disks*;
- $2(m + 1)$ disks of size $s_2 = s_1/1+s_1 = 33/166$ (≈ 0.198795), we will refer to these disks as *small filler disks*;
- 2 disks of size $s_3 = \frac{1-s_0^2-2s_0}{4s_0} = 2311/13200$ (≈ 0.175076), referred to as *end disks*;
- $3m$ disks $D_1, \ldots, D_{3m}$, referred to as *partition disks*, where $d_i = \frac{17}{99}\left(\frac{3}{100}\frac{a_i}{B} + \frac{99}{100}\right)$.

In the following, we will identify disks by their size or type. We observe that all disk sizes are rational, where numerator and denominator can be computed in time polynomial in the input size. The radius of a disk is obtained by squaring its size. Note that, if we multiply all radii by the product of the denominators, then we obtain in polynomial time an instance of our problem with integer radii.

▶ **Lemma 8.** *Each end disk and partition disk has size at least* $s_4 = 2261/13200 > 0.17128$.

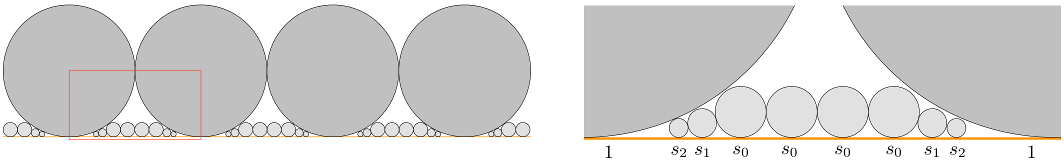
Proof. Since $s_3 > s_4$, the statement is trivial for end disks. Let $a_i \in \mathcal{A}$. From $a_i > B/4$ follows that the size d_i of the corresponding partition disk is $d_i \geqslant 17/99(3/400 + 99/100) = 17/99 \cdot 399/400 = 2261/13200$. ◀

Equivalence of the problem instances. We show that $\mathcal{D}$ has a placement with span $2(m+1)$ if and only if $(\mathcal{A}, B)$ is a Yes-instance of 3-Partition, implying the NP-hardness of CoinsOnAShelf.

The $m+1$ outer frame disks alone already require a span of $2(m+1)$, so no better span is possible. A placement of all disks of $\mathcal{D}$ with span $2(m+1)$ therefore implies that consecutive outer frame disks touch, and that all remaining disks fit into the space under these outer frame disks.

Let's call the m spaces between two consecutive (and touching) outer frame disks *gaps*. The space to the left of the leftmost outer frame disk is called the *left end*, the *right end* is defined symmetrically.

▶ **Lemma 9.** *There is only one pattern of frame and filler disks (ignoring end disks and partition disks) that has span* $2(m + 1)$.



(a) The overall picture for $m = 3$.

(b) The frame and filler disks inside a gap.

Figure 5 The unique pattern of span $2(m + 1)$ in Lemma 9.

The proof can be found in the full paper [1], here we only show the pattern in Figure 5a. Each gap contains eight disks of sizes s_2, s_1, s_0, s_0, s_0, s_0, s_1, s_2; see Figure 5b. The left end contains four disks of sizes s_0, s_0, s_1, s_2, the right end contains disks of sizes s_2, s_1, s_0, s_0.

▶ **Lemma 10.** *Three end/partition disks X, Y, and Z fit in the three gaps formed by the three pairs of consecutive inner frame disks in a common gap if and only if $x + y + z \leqslant {}^{17}/_{33}$.*

Proof. By Lemma 2, the largest disk that fits in the space between two touching disks of size s_0 has size $^{s_0}/_2$. By Lemma 8, an end/partition disk has size at least $s_4 > {}^{s_0}/_2$, so it does not fit entirely in this space. It follows that the total footpoint distance of the sequence $1, s_0, x, s_0, y, s_0, z, s_0, 1$ is at least $4s_0 + 4s_0x + 4s_0y + 4s_0z = 4s_0(x + y + z + 1)$. X, Y, and Z fit in the prescribed manner if and only if this total footpoint distance is at most two, proving the lemma. ◀

▶ **Lemma 11.** *Placing a disk X in the space between the two consecutive inner frame disks in the left end or the right end causes the total span to increase if and only if $x > s_3$.*

Proof. If $x \leqslant {}^{s_0}/_2 < s_3$, the statement follows from Lemma 2, so assume $x > {}^{s_0}/_2$. Then the total width of the sequence $1, s_0, x, s_0$ is $2s_0 + 4s_0x + s_0^2$. The span increases if and only if this is larger than one, proving the lemma. ◀

A 3-partition implies small span. Assume that $\mathcal{A}$ can be partitioned into m groups A_i such that $\sum_{a \in A_i} a = B$. Consider a group $A_i = (a_{i1}, a_{i2}, a_{i3})$ and let X, Y, and Z be the partition disks corresponding to a_{i1}, a_{i2}, a_{i3}. Then we have

$$x + y + z = \frac{17}{99}\left(\frac{3}{100}\,\frac{a_{i1} + a_{i2} + a_{i3}}{B} + 3 \cdot \frac{99}{100}\right) = \frac{17}{33}.$$

By Lemma 10 this implies that X, Y, and Z can be placed in a common gap in the pattern of Figure 5 without increasing the total span. Since there are m gaps, we can place all partition disks into the m gaps. Finally, by Lemma 11, we can place the two end disks inside the left end and the right end.

Small span implies a 3-partition. We assume now that a placement of the disks $\mathcal{D}$ with span $2(m + 1)$ exists. By Lemma 9, the frame and filler disks must be placed in the pattern of Figure 5. It remains to discuss the possible locations of the end disks and the partition disks. We need a number of observations about a placement of span $2(m + 1)$:

1. The left end and right end can contain at most one end disk or partition disk, and only between the two inner frame disks or between the outer frame disk and the small filler disk, see top of Table 1.
2. A gap can contain at most three partition disks or end disks. If a gap contains three such disks, each has to appear between two inner frame disks, see bottom of Table 1.

Table 1 Impossible placements of end/partition disks...

...in the right end

sequence	width	
1 s_0 s_0 s_4	$2s_0 + 2s_0^2 + 2s_0s_4 + s_4^2$	> 1.0201
1 s_1 s_4 s_0 s_0	$2s_1 + 2s_1s_4 + 2s_0s_4 + 3s_0^2$	> 1.0209
1 s_2 s_4 s_1 s_0 s_0	$2s_2 + 2s_2s_4 + 2s_1s_4 + 2s_1s_0 + 3s_0^2$	> 1.0411
1 s_0 s_4 s_4 s_0	$2s_0 + 4s_0s_4 + 2s_4^2 + s_0^2$	> 1.0536
1 s_4 s_4 s_2 s_1 s_0 s_0	$2s_4 + 2s_4^2 + 2s_2s_4 + 2s_1s_2 + 2s_1s_0 + 3s_0^2$	> 1.0584
1 s_4 s_2 s_1 s_0 s_4 s_0	$2s_4 + 2s_2s_4 + 2s_1s_2 + 2s_1s_0 + 4s_0s_4 + s_0^2$	> 1.0080

...in a gap

sequence	total footpoint distance	
1 s_1 s_4 s_0 s_0 s_0 s_0 1	$2s_1 + 2s_1s_4 + 2s_0s_4 + 6s_0^2 + 2s_0$	> 2.0076
1 s_2 s_4 s_1 s_0 s_0 s_0 s_0 1	$2s_2 + 2s_2s_4 + 2s_4s_1 + 2s_1s_0 + 6s_0^2 + 2s_0$	> 2.0278
1 s_0 s_4 s_4 s_0 s_0 s_0 1	$4s_0 + 4s_0s_4 + 2s_4^2 + 4s_0^2$	> 2.0403
1 s_4 s_4 s_2 s_1 s_0 s_0 s_0 s_0 1	$2s_4 + 2s_4^2 + 2s_4s_2 + 2s_2s_1 + 2s_1s_0 + 6s_0^2 + 2s_0$	> 2.0451
1 s_4 s_2 s_1 s_0 s_4 s_0 s_4 s_0 s_0 1	$2s_4 + 2s_4s_2 + 2s_2s_1 + 2s_1s_0 + 8s_0s_4 + 2s_0^2 + 2s_0$	> 2.0030
1 s_4 s_2 s_1 s_0 s_4 s_0 s_0 s_0 s_1 s_2 s_4 1	$4s_4 + 4s_4s_2 + 4s_2s_1 + 4s_1s_0 + 4s_0s_4 + 4s_0^2$	> 2.0078

3. Since there are $3m + 2$ end and partition disks, (1) and (2) imply that each gap contains three such disks, while the left end and right end each contain one.
4. By (1) and Lemma 11, the left end and the right end can contain only disks of size at most s_3. We can assume that these are the two end disks (otherwise, swap them with an end disk).
5. Consider a gap. It contains exactly three partition disks X, Y, and Z. By Lemma 10, we have $x + y + z \leqslant 17/33$. Let a, b, c be the elements of $\mathcal{A}$ corresponding to X, Y, and Z. Then we have

$$x + y + z = \frac{17}{99}\left(\frac{3}{100}\frac{a+b+c}{B} + 3 \cdot \frac{99}{100}\right) \leqslant \frac{17}{33},$$

which implies $a + b + c \leqslant B$. It follows that we have partitioned the elements of $\mathcal{A}$ into m groups $A_1, A_2, \ldots, A_m$ with $\sum_{a \in A_i} a \leqslant B$. Since $\sum_{a \in \mathcal{A}} a = mB$, we must have $\sum_{a \in A_i} a = B$ for each i, so $(\mathcal{A}, B)$ is a Yes-instance of 3-Partition.

This concludes the proof of Theorem 7, noting that by Lemma 8 all disks have size at least $s_4 > 1/6$.

5 A 4/3-Approximation

In this section, we give a *greedy algorithm* and prove that it computes a 4/3-approximation to the problem.

Our algorithm starts by sorting the disks $D_1, D_2, \ldots, D_n$ by decreasing size, such that $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n$. It then considers the disks one by one, in this order, maintaining a placement of the disks considered so far. Each disk D is placed as follows:

1. If there is a gap between two consecutive disks A and B in the current placement that is large enough to contain D, then we place D in this gap, touching the *smaller* one of the two disks A and B.
2. Otherwise, let A be the leftmost disk in the current placement (that is, the disk with the leftmost footpoint—this is not necessarily the disk defining the left end of the current

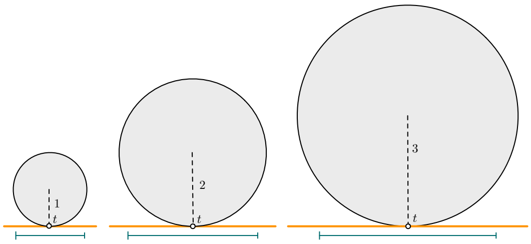


Figure 6 Support of three disks of radius 1, 2 and 3 respectively.

span), and let Z be the rightmost disk. Since $d \leqslant a$, we can place D so that it touches A from the left (candidate placement D_A), and since $d \leqslant z$, we can place D so that it touches Z from the right (candidate placement D_Z).

3. If one of the candidate placements D_A or D_Z does not increase the span, we place D in this way.
4. Otherwise, we place D at D_A if $a > z$ and at D_Z otherwise.

The algorithm can be implemented to run in time $O(n \log n)$ as follows: We maintain a priority queue that stores, for each pair of consecutive disks, the size of the largest disk that will fit between them. Since we are placing disks in order of decreasing size, a newly placed disk can only touch its two neighbors, and so it will fit into the gap if and only if its size is at most the stored gap size.

For the analysis of the approximation factor, we will assume, without loss of generality, that the final disk D_n is placed using the last rule (as otherwise it does not contribute to the final span and can be ignored in the analysis). We also assume that $d_n = 1$.

Next, let's call a disk D *large* if $d \geqslant 2$, and *small* otherwise. We have the following:

▶ **Lemma 12.** *Any two consecutive small disks placed by the algorithm touch.*

Proof. Assume, for a contradiction, that D is the *first* small disk whose placement causes two small disks to be consecutive but non-touching.

If D was placed by the third or fourth rule (at the left or right end of the sequence), it is touching its only neighbor. Therefore, D must have been placed in a gap between two disks A and B. If both A and B are small, they must be touching (since D is the first small disk that will not touch a neighboring small disk). But by Lemma 2 that implies that the gap between A and B is too small to contain a disk of size $d \geqslant 1$. It follows that at most one of A and B is small, say B. But then the algorithm will place D such that it touches B, a contradiction. ◀

We now associate with each disk a *support interval*. The support interval of a disk A is the interval $[\underset{\cdot}{A} - 2a + 1, \underset{\cdot}{A} + 2a - 1]$. Since $0 \leqslant (a-1)^2 = a^2 - 2a + 1$, we have $2a - 1 \leqslant a^2$, and so the support interval of a disk lies within the disk's span, see Figure 6.

▶ **Lemma 13.** *In any feasible placement of disks of size at least one, the open support intervals of the disks are disjoint.*

Proof. Consider the function $f(a,b) = (a+b-1)/ab$ for $a, b \geqslant 1$. Since $f(1,\cdot) = f(\cdot,1) = 1$ and the partial derivatives of f are negative for $a, b > 1$, we have $f(a,b) \leqslant 1$.

Consider two consecutive touching disks of size a and b. Their footpoints are at distance $2ab$. The support intervals cover $2a + 2b - 2$ of this distance. From $f(a,b) \leqslant 1$ it follows that $2a + 2b - 2 \leqslant 2ab$, and so the support intervals do not overlap. ◀

Lemma 13 implies that the total length of the support intervals is a lower bound for the span of a family of disks. We will show that our greedy algorithm computes a solution where the support intervals cover at least $3/4$ of the span, implying approximation factor $4/3$.

Consider a pair of two consecutive disks A and B placed by the algorithm, and let G be the (imaginary) largest disk that can be placed in the gap between A and B. Since D_n was not placed in this gap, we have $g < 1$. By Lemma 1, we have $\underset{\cdot}{A}\underset{\cdot}{B} = \underset{\cdot}{A}\underset{\cdot}{G} + \underset{\cdot}{G}\underset{\cdot}{B} = 2ag + 2gb = 2g(a+b)$.

Consider first the case where A and B touch. Lemma 2 gives $1/g = 1/a + 1/b$ or $g = ab/(a+b)$. The support intervals cover $2a + 2b - 2$ of the footpoint distance $2ab$, so the ratio is $1/a + 1/b - 1/ab$. For $1 \leqslant a, b$ under the constraint $1/a + 1/b > 1$ this is minimized at $a = b = 2$ and we have $1/a + 1/b - 1/ab \geqslant 3/4$, so the claim holds for this interval.

Now suppose that A and B do not touch. By Lemma 12, this means at least one of the disks is large, say A, that is $a \geqslant 2$. The footpoint distance $\underset{\cdot}{A}\underset{\cdot}{B}$ is $2g(a+b) \leqslant 2(a+b)$, and the support intervals cover $2a + 2b - 2$ of this distance, so the ratio is

$$\frac{2a+2b-2}{2g(a+b)} \geqslant \frac{a+b-1}{a+b} = 1 - \frac{1}{a+b}.$$

If $a \geqslant 3$ or $b \geqslant 2$, we already have $1 - 1/(a+b) \geqslant 3/4$, and this bound is good enough.

It remains to consider the situation when $2 \leqslant a < 3$ and $1 \leqslant b \leqslant 2$. Without loss of generality, we assume that B is to the right of A. We denote the first disk to the right of A that is touching A as D. By the nature of our algorithm, when B was placed, it was placed inside the space between A and D (possibly, other disks were already present in this space at that time). Since B does not touch A, the disk D must be smaller than A, that is $1 \leqslant d \leqslant a < 3$.

We analyze the entire interval $[\underset{\cdot}{A}, \underset{\cdot}{D}]$ as a whole. Since A and D touch, the length of this interval is $2ad$. In between A and D, some $k \geqslant 1$ disks have been placed, with B being the leftmost of these.

We first consider the case $k \geqslant 2$. If two disks X and Y of size one fit between A and D, then we have

$$2ad = \underset{\cdot}{A}\underset{\cdot}{D} = \underset{\cdot}{A}\underset{\cdot}{X} + \underset{\cdot}{X}\underset{\cdot}{Y} + \underset{\cdot}{Y}\underset{\cdot}{D} \geqslant 2a + 2 + 2d,$$

and from $a < 3$ follows

$$d \geqslant \frac{a+1}{a-1} = 1 + \frac{2}{a-1} > 2.$$

The total length of the support intervals in the interval $\underset{\cdot}{A}\underset{\cdot}{D}$ is at least $2a - 1 + 2d - 1 + 2k \geqslant 2(a + d + 1)$. The distance $\underset{\cdot}{A}\underset{\cdot}{D}$ is $2ad$. For $2 \leqslant a, d \leqslant 3$, the ratio $(a+d+1)/ad$ is at least $7/9 > 3/4$, implying the claim.

In the second case, B is the only disk between A and D. This means that B touches D. The total support interval length in the interval $\underset{\cdot}{A}\underset{\cdot}{D}$ is

$$2a - 1 + 4b - 2 + 2d - 1 = 2a + 4b + 2d - 4.$$

Let G be the largest disk that fits in the gap between A and B. Its size is determined by the equality $2ag + 2gb + 2bd = 2ad$, so $g = (a-b)d/(a+b)$. Since D_n was not placed in this gap, we have $g < 1$, and so $(a-b)d < a + b$. Minimizing the expression

$$\frac{a+2b+d-2}{ad}$$

under the constraints $2 \leqslant a \leqslant 3$, $1 \leqslant d \leqslant a$, $1 \leqslant b \leqslant 2$, and $(a-b)d < a+b$ leads to the minimum $7/9 > 3/4$ for $a = d = 3$ and $b = 3/2$.

To complete the proof, we need to argue about the part of the span that does not lie between two footpoints, in other words, the two intervals between the left wall (defined by the leftmost point on any disk) and the leftmost footpoint, and between the rightmost footpoint and the right wall. Recall that we assumed that placing D_n increased the total span. This implies that D_n was placed using the algorithm's last rule and therefore touches one of the two walls, let's say the right wall. Let A and B be the leftmost two disks (in footpoint order), and let Y and Z be the rightmost two disks (in footpoint order). By assumption, $Z = D_n$ and so $z = 1$. Since D_n was placed using the last rule, we have $y \geqslant a$, and Z touches Y. Let us call G the (imaginary) largest disk that would fit into the space between the left wall and A. Since D_n was not placed in this position, we have $g < 1$. Note that the left wall is at coordinate $\underset{.}{G} - g^2$, the right wall at coordinate $\underset{.}{Z} + 1$. We now distinguish two cases.

We first consider the case where $a \geqslant 3/2$. We then analyze the two intervals $[\underset{.}{G} - g^2, \underset{.}{A}]$ and $[\underset{.}{Y}, \underset{.}{Z} + 1]$ together. Their total length is $g^2 + 2ga + 2y + 1 < 2y + 2a + 2$, and the support intervals of A, Y, and Z cover $2a - 1 + 2y - 1 + 2 = 2y + 2a$ of this. The ratio is

$$\frac{2y+2a}{2y+2a+2} = 1 - \frac{1}{y+a+1} \geqslant 1 - \frac{1}{4} = \frac{3}{4} \qquad \text{since} \quad y \geqslant a \geqslant 3/2.$$

In the second case we have $a < 3/2$. Then B must be touching A. This is true if $b \geqslant a$, because then A was placed later than B using the third rule. When $b < a$, then it follows from Lemma 12. The distance between $\underset{.}{G} - g^2$ and $\underset{.}{B}$ is then $g^2 + 2ag + 2ab \leqslant 2ab + 2a + 1 \leqslant 3b + 4$. Since B fits inside the span, we must have $b^2 \leqslant 3b + 4$, which solves to $-1 \leqslant b \leqslant 4$.

We now analyze the intervals $[\underset{.}{G} - g^2, \underset{.}{B}]$ and $[\underset{.}{Y}, \underset{.}{Z} + 1]$ together. Their total length is

$$g^2 + 2ga + 2ab + 2y + 1 < 2y + 2a + 2ab + 2,$$

while the support intervals of A, B, Y, and Z cover

$$4a - 2 + 2b - 1 + 2y - 1 + 2 = 2y + 4a + 2b - 2.$$

Since $y \geqslant a$, we can lower-bound the ratio

$$\frac{2y + 4a + 2b - 2}{2y + 2a + 2ab + 2} \geqslant \frac{6a + 2b - 2}{4a + 2ab + 2} = \frac{3a + b - 1}{2a + ab + 1}.$$

Consider the function $h(a, b) = 3a + 2b - 3ab/2$ over the domain $1 \leqslant a \leqslant 3/2$ and $1 \leqslant b \leqslant 4$. For fixed b, the function $h(a, b)$ is linear in a, so $h(a, b) \geqslant \min\bigl\{h(1, b), h(3/2, b)\bigr\}$. We have $h(1, b) = 3 + 2b - 3b/2 = 3 + b/2 \geqslant 7/2$ and $h(3/2, b) = 9/2 + 2b - 9b/4 = 9/2 - b/4 \geqslant 7/2$.

It follows that $\frac{3}{2}a + b - \frac{3}{4}ab \geqslant \frac{7}{4}$, and so

$$3a + b - 1 \geqslant \frac{3}{2}a + \frac{3}{4}ab + \frac{3}{4} = \frac{3}{4}\bigl(2a + ab + 1\bigr).$$

Note that in this second case we have used the interval $[\underset{.}{A}, \underset{.}{B}]$ to help bound the coverage of the two end intervals. This could be a problem if the same interval was also needed to help bound a larger interval of the form $[\underset{.}{A}, \underset{.}{C}]$, where A and C touch and B was inserted into this interval later. But note that we needed to analyze $[\underset{.}{A}, \underset{.}{C}]$ as a whole only if $c < 3$. Since $a < 3/2$, no disk of size one would then fit into the gap between A and C, so this situtation cannot occur.

This completes the proof of the following theorem.

▶ **Theorem 14.** *The greedy algorithm computes a $4/3$-approximation in time $O(n \log n)$.*

6 Conclusions

Our best approximation algorithm achieves an approximation factor of $4/3$. We were unable to find a polynomial time approximation scheme, so it would be natural to try to prove that the problem is APX-hard. This, however, seems unlikely to be true, for the same reasons as outlined by Dürr et al. [4]: The ideas they present appear to transfer to our problem, and would lead to an $2^{O(\log^{O(1)} n)}$ algorithm with approximation factor $(1+\varepsilon)$. APX-hardness, on the other hand, would imply that for some $\varepsilon > 0$ this approximation problem is NP-hard, implying subexponential algorithms for NP.

Acknowledgments. We thank Peyman Afshani and Ingo van Duin for helpful discussions during O.C.'s visit to Madalgo in 2016. We also thank all participants at the Korean Workshop on Computational Geometry in Würzburg 2016.

References

1 Helmut Alt, Kevin Buchin, Steven Chaplick, Otfried Cheong, Philipp Kindermann, Christian Knauer, and Fabian Stehn. Placing your coins on a shelf, 2017. arXiv:1707.01239.

2 Hallard T. Croft, Kenneth J. Falconer, and Richard K. Guy. *Unsolved Problems in Geometry.* Springer-Verlag, 1991. pp. 108–110.

3 Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard. In A. K. Peters, editor, *Origami^5: Proceedings of the 5th International Conference on Origami in Science, Mathematics and Education (OSME 2010)*, pages 609–626, 2010.

4 Christoph Dürr, Zdeněk Hanzálek, Christian Konrad, Yasmina Seddik, René Sitters, Óscar C. Vásquez, and Gerhard Woeginger. The triangle scheduling problem. *Journal of Scheduling*, pages 1–8, 2017. doi:10.1007/s10951-017-0533-1.

5 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

6 Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5(e2):1–29, 2017. doi:10.1017/fmp.2017.1.

7 Thomas C. Hales and Samuel P. Ferguson. A formulation of the Kepler conjecture. *Discrete & Computational Geometry*, 36(1):21–69, 2006.

8 Johannes Kepler. *Strena seu de nive sexangula (The six-cornered snowflake).* 1611.

9 Boris Klemz, Martin Nöllenburg, and Roman Prutkin. Recognizing weighted disk contact graphs. In *Graph Drawing and Network Visualization*, volume 9411 of *LNCS*, pages 433–446. Springer, 2015.

10 Eckehard Specht. www.packomania.com. Accessed 2017-06-28. URL: http://www.packomania.com/.

11 Yu. G. Stoyan and Georgiy Yaskov. A mathematical model and a solution method for the problem of placing various-sized circles into a strip. *European Journal of Operational Research*, 156(3):590–600, 2004.

On the Number of p4-Tilings by an n-Omino*

Kazuyuki Amano†[1] **and Yoshinobu Haruyama**[2]

1 **Department of Computer Science, Gunma University, Japan**
amano@gunma-u.ac.jp

2 **Department of Computer Science, Gunma University, Japan**
haruyama@amano-lab.cs.gunma-u.ac.jp

Abstract

A plane tiling by the copies of a polyomino is called isohedral if every pair of copies in the tiling has a symmetry of the tiling that maps one copy to the other. We show that, for every n-omino (i.e., polyomino consisting of n cells), the number of non-equivalent isohedral tilings generated by 90 degree rotations, so called p4-tilings or quarter-turn tilings, is bounded by a constant (independent of n). The proof relies on the analysis of the factorization of the boundary word of a polyomino.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases polyomino, plane tiling, isohedral tiling, word factorization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.5

1 Introduction

The investigation of plane tilings by polyominoes have attracted many researchers for a long time. In this paper, we focus on the following type of problem: what is the maximum number of *isohedral* tilings that a single polyomino can have? A plane tiling by a polyomino is called isohedral if every pair of copies in the tiling has a symmetry of the tiling that maps one copy to the other. Two tilings are said to be *equivalent* if they are congruent, i.e., they can be mapped onto each other by a combination of rotations, translations and reflections.

A polyomino having an isohedral tiling can be classified into seven types according to its boundary word. See the recent work by Langerman and Winslow [14, Section 3] for a clear description of the classification based on earlier works (e.g., [11]). In this paper, we focus on the isohedral tiling called *p4-tiling* (or *quarter-turn* tiling) among these seven types. A polyomino is said to have a p4-tiling if it covers the plane by only 90 degree rotations around two designated points called *rotation centers*. See Figure 1.

Some polyominoes have multiple p4-tilings. Figure 2 shows an example of a pentomino (i.e., 5-omino) having two non-equivalent p4-tilings. One can see that each pentomino is adjacent to four (five, respectively) pentominoes in the left (right, respectively) tiling.

It is known that (see e.g., [6, 7]), if an n-omino has a p4-tiling, then the relative distance (x, y) of two rotation centers satisfies

$$n = \frac{x^2 + y^2}{2}. \qquad (1)$$

This says that an n-omino can have a p4-tiling only if

$$n = 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, \ldots.$$

[*] This work was partially supported by JSPS Kakenhi No. 15K00006 and 24106006.

[†] Corresponding author.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 5; pp. 5:1–5:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

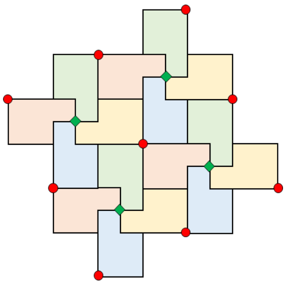


Figure 1 An example of a p4-tiling by a pentomino. The rotation centers are represented by red and green circles.

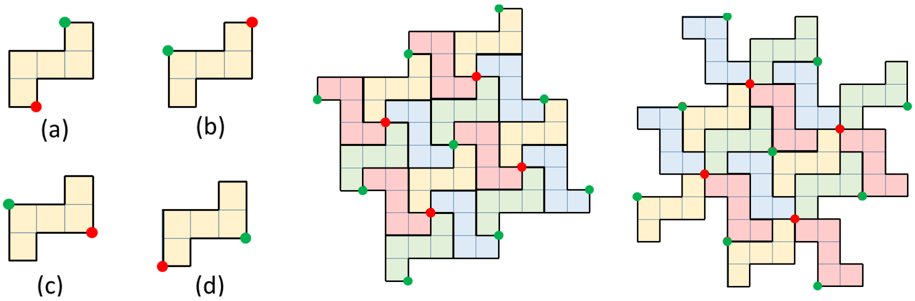


Figure 2 A pentomino having two non-equivalent p4-tilings. The rotation centers shown in (a) gives the left tiling, and (b) gives the right tiling. The rotation centers shown in (c) gives the tiling equivalent to the left tiling, and (d) is considered to be same as (b) since these are overlapped by the 180 degree rotation.

For example, every pentomino (like the one in Figure 2) has rotation centers with relative distance $(1, 3)$ (ignoring the order of x and y). We call such a tiling as a p4-tiling *with center* $(1, 3)$.

Eq. (1) not only restricts the values of n, but also arises another multiplicity of p4-tilings since Eq. (1) may have many solutions. For examples, $(x, y) = (1, 7)$ and $(5, 5)$ satisfies Eq. (1) for $n = 25$. A computer experiment shows that among 2,557,227,044,764 25-ominoes (Sequence A000105 in OEIS [15]), 3,076,890 and 1,526,416 have a p4-tiling with center $(1, 7)$ and $(5, 5)$, respectively. The size of their intersection is 10,824. See Figure 3 for one of such 25-ominoes.

The number of solutions to Eq. (1) can be unbounded as n goes to infinity (see e.g., [10, 17]). Indeed, if n is factored as $n = 2^{a_0} p_1^{2a_1} \cdots p_r^{2a_r} q_1^{b_1} \cdots q_r^{b_r}$, where the p_is are primes of the form $4k + 3$ and the q_is are primes of the form $4k + 1$, then the number of solutions $R(n)$ to Eq. (1) (allowing zeros and ignoring order and signs) is given by[1]

$$R(n) \quad = \quad \begin{cases} 0, & \text{if any } a_i \text{ is a half-integer,} \\ \lceil \frac{(b_1+1)(b_2+1)\cdots(b_r+1)}{2} \rceil, & \text{if all } a_i \text{ are integers.} \end{cases} \tag{2}$$

[1] In [17], the formula for the number of solutions of $n = x^2 + y^2$ not allowing zeros and ignoring order and signs is given. Eq. (2) is essentially the same to this by observing $n = x^2 + y^2$ iff $2n = (x - y)^2 + (x + y)^2$.

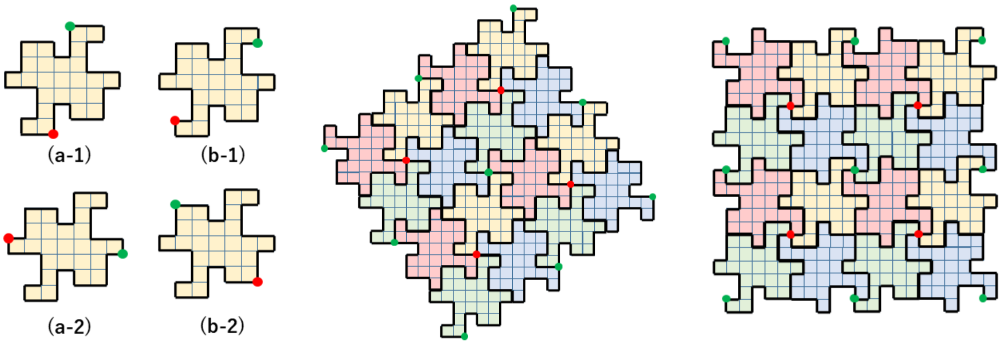


Figure 3 A 25-omino having p4-tilings with centers $(1, 7)$ and $(5, 5)$. It has four pairs of rotation centers. The rotation centers (a-1) and (a-2) admit the left tiling, and (b-1) and (b-2) admit the right tiling. Note that this is the only 25-omino having (at least) four pairs of rotation centers found through our experiments.

Hence, for example, $n = 5^{2k-1}$ has $k = \Theta(\log n)$ solutions.

Figure 4 shows a 1300-omino that has three p4-tilings with centers $(x, y) = (10, 50), (22, 46)$ and $(34, 38)$. Note that $325 = (5^2 + 25^2)/2 = (11^2 + 23^2)/2 = (17^2 + 19^2)/2$ is the smallest integer having three solutions to Eq. (1) (up to the order of x and y), but we have not succeeded to find a 325-omino that has p4-tilings for these three distances. Note also that we found the 1300-omino shown in Figure 4 by using a SAT solver [12].

Now the following questions become interesting: what is the maximum number of p4-tilings that a single polyomino can have? Is it bounded, or is there a polyomino having an unbounded number of p4-tilings?

1.1 Our Contributions

The contribution of this paper is to show that the number of p4-tilings by an n-omino is bounded by a constant (Theorem 1). This is true even for n having an unbounded number of solutions to Eq. (1). It is in sharp contrast to the tiling by *translations*; a $1 \times n$ rectangle has $\Theta(n)$ translation tilings.

In order to show the upper bound on the number of p4-tilings, we use an equivalence between p4-tilings and factorizations of the boundary word of a polyomino into some specific form. Then, we show that the number of such factorizations is bounded for every polyomino based on the analysis of words having such factorizations.

1.2 Related Works

There are plenty of works dealing with polyomino tilings since Golomb [9] initiated the work in 60s. We listed here only those closely related to our work.

Fukuda et al. [6, 7] enumerated n-ominoes that have a p4-tiling for $n \leq 10$. Horiyama and Samejima [13] gave an algorithm for generating many n-ominoes for p4-tilings.

Winslow [18] gave a linear time algorithm for deciding whether a given polyomino has an isohedral tiling by *translations*. This improved the earlier works of Beauquier and Nivat [1], Gambini and Vuillon [8] and Provençal [16] in terms of its running time, and generalized the works of Brlek et al. [2, 3] that dealt with some restricted cases. In [18], Winslow also

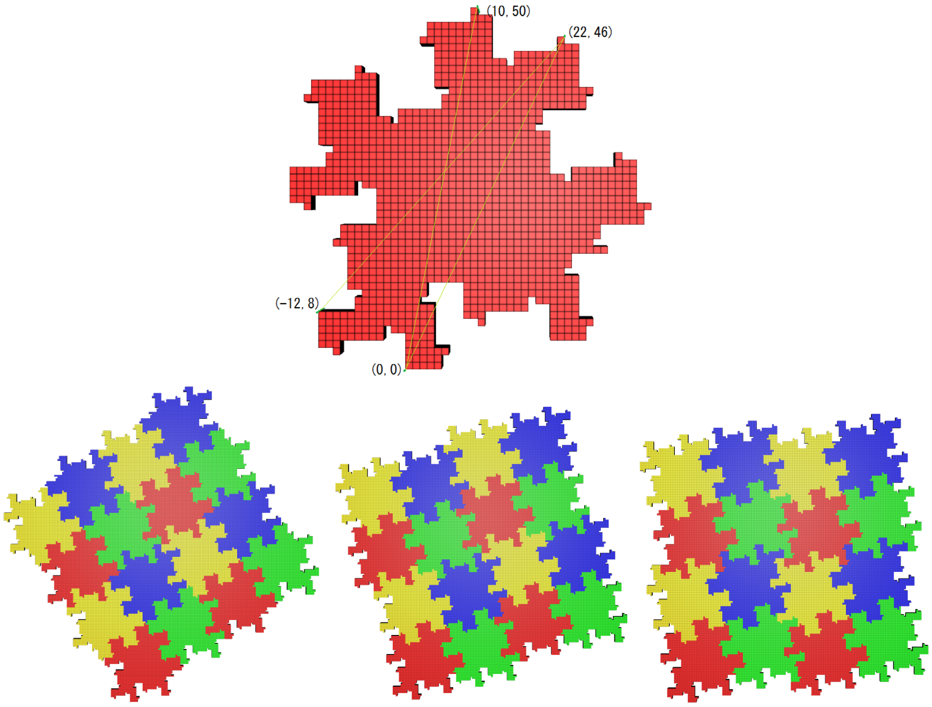


Figure 4 Top: A 1300-omino having p4-tilings for three pairs of rotation centers with distinct relative distances. Bottom: A p4-tiling by rotation centers $(0,0)$ and $(10,50)$ (left), $(0,0)$ and $(22,46)$ (center), and $(-12,8)$ and $(22,46)$ (right).

showed that every n-omino has $O(n)$ translation tilings. Recently, Langerman and Winslow [14] extended this work to give quasi-linear time algorithms for all of seven types of isohedral tilings. Their results include a linear time algorithm for deciding whether a polyomino has a p4-tiling. We note here that our proof borrows many useful analyses on words that appeared in their work [14].

2 Notations and Definitions

The notations and definitions used in this paper are similar to those of Langerman and Winslow [14].

A *polyomino* is a two-dimensional shape formed by connecting one or more unit squares edge to edge. A polyomino consisting of n unit squares is called *n-omino*. A polyomino having a p4-tiling never includes a hole, and so its boundary is naturally represented by a four-letter word. A *letter* is a symbol $x \in \Sigma = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$, where $\mathbf{0}, \mathbf{1}, \mathbf{2}$ and $\mathbf{3}$ represent the directions up, right, down and left, respectively. If no confusion arises, we identify the letters $\mathbf{0}, \mathbf{1}, \mathbf{2}$ and $\mathbf{3}$ with the integers 0, 1, 2 and 3, respectively. This is convenient when we consider a rotation. For $\Theta \in \{0, 90, 180, 270\}$, the *$\Theta°$-rotation* of a letter x, denoted $t_\Theta(x)$, is the letter obtained by rotating x clockwise by $\Theta°$, i.e., $t_{90k}(x) = x + k \pmod 4$ for $k \in \{0, 1, 2, 3\}$[2].

[2] Remark that our definition of the function $t_\Theta(\cdot)$ is different from the one used in [14].

A *word* is a sequence of letters and the *length* of a word W, denoted by $|W|$, is the number of letters in W. For an integer $1 \le i \le |W|$, the i-th letter of a word W is denoted by $W[i]$, and the i-th from the last letter of W is denoted by $W[-i]$.

A *factor* (or *subword*) of W is a contiguous sequence X of letters in W, written $X \preceq W$. A factor $X \preceq W$ is a *prefix* if X starts at $W[1]$, written $X \preceq_{\mathrm{pre}} W$. Similarly, $X \preceq W$ is a *suffix* if X ends at $W[-1]$, written $X \preceq_{\mathrm{suf}} W$. For $1 \le i \le j \le |W|$, a factor of W that starts at $W[i]$ and ends at $W[j]$ is denoted by $W[i : j]$. We say that a word W has a *period* p if $W[i] = W[i+p]$ for every $1 \le i \le |W| - p$.

The *reverse* of a word W, denoted by $\overleftarrow{W}$, is the word obtained by reading W in reverse order. The Θ-*rotation* of a word W, denoted by $t_\Theta(W)$, is the word obtained by replacing each letter of W by its Θ-rotation. A word W is called a *palindrome* if $W = \overleftarrow{W}$, and is called a Θ-*drome* if $W = Xt_\Theta(\overleftarrow{X})$ for some word X. For example, $W = \mathbf{010121}$ is a 90-drome with $X = \mathbf{010}$. Note that, in this paper, we only deal with a palindrome of even length.

The boundary of a polyomino can naturally be represented by a *circular* word on Σ, in which a first letter is not fixed. The *boundary word* of a polyomino P, denoted by $\mathcal{B}(P)$, is the circular word on Σ coding the boundary of P in a clockwise manner. For example, the boundary word of the pentomino P_5 shown in Figure 2 is $\mathcal{B}(P_5) = \mathbf{300110122332}$. A Θ-drome (or a palindrome for which $\Theta := 0$) factor X of a circular word W is said to be *admissible* if $W = XU$ satisfies $U[-1] \neq t_\Theta(U)[1]$, which intuitively says that X is maximal in a natural sense.

3 p4-tilings with Multiple Rotation Centers

This section is devoted to prove the main theorem of this paper.

▶ **Theorem 1.** *Every polyomino has* $O(1)$ *p4-tilings.*

In Section 3.1, we describe a number of lemmas which will be used in the proof of Theorem 1. Many of them are taken from Langerman and Winslow [14]. Here and hereafter, when we refer [14] the numbering of the lemmas or theorems is according to its full version (appeared in `arXiv:1507.02762v2`). The main body of the proof of Theorem 1 is described in Section 3.2.

The characterization of a polyomino that admits a p4-tiling in terms of its boundary word is formalized as follows (See e.g., [14, Section 3].)

▶ **Theorem 2.** *A polyomino* P *has a p4-tiling if and only if its circular boundary word* $\mathcal{B}(P)$ *can be factorized as* $\mathcal{B}(P) = ABC$ *where* A *is a palindrome and* B, C *are 90-dromes. In addition, all of* A*,* B *and* C *are admissible, and* A *can be empty but* B *and* C *are non-empty.* ◀

The admissibility is shown in [14, Lemma 5.8], and the non-emptiness of 90-dromes is verified as follows: Suppose to the contrary that a polyomino P with $\mathcal{B}(P) = AB$, where $|B| \neq 0$, has a p4-tiling. One of the rotation centers should be just before $B[1]$ or just after $B[-1]$ (the other center is between $B[|B|/2]$ and $B[|B|/2+1]$). Suppose without loss of generality that $B[1] = \mathbf{0}$ and $B[-1] = \mathbf{1}$. Then, $A[1] (= A[-1]) \in \{\mathbf{2}, \mathbf{3}\}$ is prohibited since it contradicts that $\mathcal{B}(P)$ is the boundary word of a polyomino, and $A[1] (= A[-1]) \in \{\mathbf{0}, \mathbf{1}\}$ is prohibited since it contradicts that exactly one of four cells surrounding a rotation center should be occupied by P.

Recall that two tilings are said to be equivalent if they can be mapped onto each other by combination of rotations, translations and reflections. Obviously, the number of non-equivalent p4-tilings by a polyomino P is upper bounded by the number of factorizations of the boundary word $\mathcal{B}(P)$ into a form given by Theorem 2. For example, the boundary word of the pentomino P_5 shown in Figure 2 has four factorizations;

$$\begin{aligned}\mathcal{B}(P_5) &= \mathbf{300})(\mathbf{110122})(\mathbf{332} && (3)\\ &= (\mathbf{300110})(\mathbf{12})(\mathbf{2332}) && (4)\\ &= (\mathbf{300110})(\mathbf{122332}) && (5)\\ &= (\mathbf{30})(\mathbf{0110})(\mathbf{122332}) && (6)\end{aligned}$$

where the factorizations (3), (4), (5) and (6) correspond to (a), (b), (c) and (d) in Figure 2, respectively.

In the factorizations (3) and (5), a palindrome A is empty. The rotation centers are designated by the centers of each of two 90-dromes. As shown in Figure 2, the factorizations (3) and (5) (or (4) and (6)) admit an equivalent p4-tiling, and hence the number of factorizations and the number of non-equivalent p4-tilings are not necessarily identical.

3.1 Miscellaneous Lemmas

We start with the following lemmas on words.

▶ **Lemma 3.** *([14, Lemma 4.1]) Let $W = PX$ with P, W palindromes and $0 < |P| < |W|$. Then W has a period $|X|$.* ◀

▶ **Lemma 4.** *(Fine and Wilf's theorem [5], or see [14, Lemma 4.2]) Let W be a word with periods p and q. If $p + q \leq |W|$, then W also has a period $GCD(p, q)$.* ◀

▶ **Lemma 5.** *Let k be an integer such that $k \geq 2$. Let $X_1, \ldots, X_k$ be palindromes such that $X_{i+1} \preceq_{\text{pre}} X_i$ for every $1 \leq i \leq k-1$ and $|X_1| > |X_2| > \cdots > |X_k| \geq (2/3)|X_1|$. Then X_1 has a period p such that $p \leq (|X_1| - |X_k|)/(k-1)$.*

Proof. For each $2 \leq i \leq k$, let $a_k := |X_{k-1}| - |X_k|$. Let $p := \text{GCD}(a_2, \ldots, a_k)$. We will show that X_1 has a period p, which implies Lemma 5 by $\sum_{i=2}^{k} a_k = |X_1| - |X_k|$.

By Lemma 3, X_1 has a period a_2. This implies that X_k also has a period a_2 since $X_k \preceq_{\text{pre}} X_1$. Similarly, for every $2 \leq j \leq k-1$, X_j has a period a_{j+1}. These imply that X_k has periods $a_2, \ldots, a_k$. Since $|X_k| \geq \sum_{i=2}^{k} a_k$, Lemma 4 implies that X_k has a period p. Since $X_k \preceq_{\text{pre}} X_1$, $X_1[i] = X_1[i+p]$ for every $1 \leq i \leq |X_k| - p$. Since X_1 is a palindrome and $p \leq |X_1|/3$, we have $X_1[(2/3)|X_1| - p + \alpha] = X_1[|X_1|/3 + p - \alpha + 1] = X_1[|X_1|/3 - \alpha + 1] = X_1[(2/3)|X_1| + \alpha]$ for every $1 \leq \alpha \leq |X_1|/3$, which completes the proof. ◀

We will use the following lemma extensively in Section 3.2.

▶ **Lemma 6.** *([14, Lemma 5.3]) Let P, Q, W be 90-dromes with $P, Q \preceq_{\text{pre}} W$ and $|P| < |Q| < |W|$. Then $|P| < (2/3)|W|$, or equivalently $|W| > (3/2)|P|$.* ◀

An analogous argument to the proof of Lemma 6 gives a suffix version of the lemma.

▶ **Corollary 7.** *Let P, Q, W be 90-dromes with $P, Q \preceq_{\text{suf}} W$ and $|P| < |Q| < |W|$. Then $|P| < (2/3)|W|$, or equivalently $|W| > (3/2)|P|$.* ◀

The following lemma, which will also be used in Section 3.2, can easily be derived from Lemma 6 (and Corollary 7).

▶ **Lemma 8.** *There is no word that can be factorized into two 90-dromes in more than 8 ways.*

Proof. Suppose to the contrary that, W can be factorized into X_iY_i where X_i and Y_i are 90-dromes for $1 \leq i \leq 9$. Without loss of generality, we assume that $|X_1| < |X_2| < \cdots < |X_9|$, and also that $|X_5| \geq |W|/2$ (by interchanging X's and Y's and will use Corollary 7 instead of Lemma 6 if necessary). Since $X_i \preceq_{\mathrm{pre}} X_{i+1}$ for every $1 \leq i \leq 8$, Lemma 6 implies $|X_9| > (3/2)|X_7| > (9/4)|X_5| > |W|$, which is a contradiction. ◀

▶ **Lemma 9.** *([14, Lemma 5.4]) Let W be a boundary word. There exist $O(1)$ admissible 90-drome factors of W with length at least $|W|/3$.* ◀

The following lemma, whose proof is elementary but a bit long, is also useful in the proof of Theorem 1.

▶ **Lemma 10.** *Let W be a word. The number of factorizations of W in such a way that $W = XY$ with X a non-empty 90-drome and Y a palindrome is at most two. The same holds for $W = YX$ with X a non-empty 90-drome and Y a palindrome.*

Proof. We only show the first case of the lemma (the proof of the second case is completely analogous.). Suppose to the contrary that $W = X_1Y_1 = X_2Y_2 = X_3Y_3$ where X_1, X_2, X_3 are 90-dromes with $0 < |X_1| < |X_2| < |X_3|$ and Y_1, Y_2, Y_3 are palindromes. Put $p_i := |X_i|/2$ for $1 \leq i \leq 3$. Note that $0 < p_1 < p_2 < p_3$. For $1 \leq i \leq 3$, let L_i and R_i denote the first and second half of X_i, respectively.

We divide the proof into several cases depending on the values of p_1, p_2 and p_3.

Case 1: $p_1 + p_2 > p_3$.

Note that, in this case, we have $p_1 \geq 2$ since $p_3 - p_2 \geq 1$. Let $v_0 := p_1 + p_2 - p_3$ and suppose without loss of generality that $W[v_0] = \mathbf{0}$. Since $1 \leq v_0 \leq p_1 - 1$, $W[v_0] \in L_1$. So the reflection of $W[v_0]$ w.r.t. the center of X_1 is the $v_1 := (p_1 - p_2 + p_3 + 1)$-th letter in W and $W[v_1] = W[v_0] + 1 = \mathbf{1}$ since X_1 is a 90-drome. Alternatively, we can write this as

$$W[v_1] \quad = \quad X_1[-v_0] = X_1[v_0] + 1 = W[v_0] + 1 = \mathbf{1}.$$

Since $v_1 \leq p_3$, we have $W[v_1] \in L_3$. Hence the reflection of $W[v_1]$ w.r.t. the center of X_3 is the $v_2 := (-p_1 + p_2 + p_3)$-th letter in W and $W[v_2] = W[v_1] + 1 = \mathbf{2}$. By continuing this argument to X_2, X_1, X_3, X_2 in this order, we have the chain of implications:

$$\begin{aligned} W[v_0] = \mathbf{0} \quad &\Rightarrow \quad W[v_1 := p_1 - p_2 + p_3 + 1] = \mathbf{1} \qquad (\because W[v_0] \in L_1) \\ &\Rightarrow \quad W[v_2 := -p_1 + p_2 + p_3] = \mathbf{2} \qquad (\because W[v_1] \in L_3) \\ &\Rightarrow \quad W[v_3 := p_1 + p_2 - p_3 + 1] = \mathbf{1} \qquad (\because W[v_2] \in R_2) \\ &\Rightarrow \quad W[v_4 := p_1 - p_2 + p_3] = \mathbf{2} \qquad (\because W[v_3] \in L_1) \\ &\Rightarrow \quad W[v_5 := -p_1 + p_2 + p_3 + 1] = \mathbf{3} \qquad (\because W[v_4] \in L_3) \\ &\Rightarrow \quad W[v_0] = \mathbf{2} \qquad (\because W[v_5] \in R_2), \end{aligned}$$

which is a contradiction.

Case 2: $p_1 + p_2 \leq p_3$.

Case 2.1: $p_2 \geq 2p_1$.

Suppose $W[1] = \mathbf{0}$. By considering the reflection of $W[1]$ w.r.t. the center of X_1, we have $W[2p_1] = X_1[-1] = X_1[1] + 1 = \mathbf{1}$. Then, by considering the reflection of these two letters w.r.t. the center of X_2, we have $W[2p_2] = X_2[-1] = X_2[1] + 1 = W[1] + 1 = \mathbf{1}$, and $W[2p_2 - 2p_1 + 1] = X_2[-2p_1] = X_2[2p_1] + 1 = W[2p_1] + 1 = \mathbf{2}$.

We further divide this case into two subcases.

Case 2.1.1: $p_3 \geq 2p_2 - 2p_1 + 1$.

By considering the reflection of $W[1]$ and $W[2p_2 - 2p_1 + 1]$ w.r.t. the center of X_3, we have $W[2p_3] = X_3[-1] = X_3[1] + 1 = W[1] + 1 = \mathbf{1}$, and $W[2p_3 - 2p_2 + 2p_1] =$

$X_3[-(2p_2 - 2p_1 + 1)] = X_3[2p_2 - 2p_1 + 1] + 1 = W[2p_2 - 2p_1 + 1] + 1 = \mathbf{3}$ (*). Here we use the condition $p_3 \geq 2p_2 - 2p_1 + 1$ to derive the second equality. However, we have

$$\begin{aligned} \mathbf{1} &= W[2p_3] = Y_2[2p_3 - 2p_2] = Y_2[-(2p_3 - 2p_2)] && (\because Y_2 \text{ is a palindrome.}) \\ &= W[-(2p_3 - 2p_2)] = Y_1[-(2p_3 - 2p_2)] = Y_1[(2p_3 - 2p_2)] && (\because Y_1 \text{ is a palindrome.}) \\ &= W[2p_3 - 2p_2 + 2p_1], \end{aligned}$$

which contradicts (*).

We can show the remaining three cases (Cases 2.1.2, 2.2.1 and 2.2.2) similarly.

Case 2.1.2: $p_3 \leq 2p_2 - 2p_1$.

By considering the reflection of $W[2p_1](= \mathbf{1})$ and $W[2p_2](= \mathbf{1})$ w.r.t. the center of X_3, we have $W[2p_3 - 2p_1 + 1] = X_3[-2p_1] = X_3[2p_1] + 1 = W[2p_1] + 1 = \mathbf{2}$, and $W[2p_3 - 2p_2 + 1] = X_3[-2p_2] = X_3[2p_2] - 1 = W[2p_2] - 1 = \mathbf{0}$ (**), where the second equality follows from $2p_2 > p_3$ in this case. On the other hand, we have

$$\begin{aligned} \mathbf{2} &= W[2p_3 - 2p_1 + 1] \\ &= Y_2[2p_3 - 2p_2 - 2p_1 + 1] && (\because 2p_3 - 2p_1 + 1 \geq p_2) \\ &= Y_2[-(2p_3 - 2p_2 - 2p_1 + 1)] && (\because Y_2 \text{ is a palindrome.}) \\ &= W[-(2p_3 - 2p_2 - 2p_1 + 1)] \\ &= Y_1[-(2p_3 - 2p_2 - 2p_1 + 1)] \\ &= Y_1[2p_3 - 2p_2 - 2p_1 + 1] && (\because Y_1 \text{ is a palindrome.}) \\ &= W[2p_3 - 2p_2 + 1], \end{aligned}$$

which contradicts (**).

Case 2.2: $p_2 < 2p_1$.

Case 2.2.1: $p_3 < 2p_2$.

Suppose that $W[1] = \mathbf{0}$. By considering the reflection of $W[1]$ w.r.t. the center of X_1 and X_2, we get $W[2p_1] = \mathbf{1}$ and $W[2p_2] = \mathbf{1}$, respectively. Then, by considering the reflection of $W[2p_2]$ w.r.t. the center of X_3, we have $W[2p_3 - 2p_2 + 1] = X_3[-2p_2] = X_3[2p_2] - 1 = W[2p_2] - 1 = \mathbf{0}$, and $W[2p_3 - 2p_1 + 1] = X_3[-2p_1] = X_3[2p_1] + 1 = W[2p_1] + 1 = \mathbf{2}$ (*3). Here we use $2p_2 > p_3$ and $2p_1 \leq p_3$. However, we have

$$\begin{aligned} \mathbf{0} &= W[2p_3 - 2p_2 + 1] \\ &= Y_1[2p_3 - 2p_2 - 2p_1 + 1] && (\because 2p_3 - 2p_2 + 1 > 2p_1 = |X_1|) \\ &= Y_1[-(2p_3 - 2p_2 - 2p_1 + 1)] && (\because Y_1 \text{ is a palindrome.}) \\ &= W[-(2p_3 - 2p_2 - 2p_1 + 1)] \\ &= Y_2[-(2p_3 - 2p_2 - 2p_1 + 1)] && (\because (2p_3 - 2p_2 - 2p_1 + 1) + |X_2| \leq 2p_3 \leq |W|) \\ &= Y_2[2p_3 - 2p_2 - 2p_1 + 1] && (\because Y_2 \text{ is a palindrome.}) \\ &= W[2p_3 - 2p_1 + 1], \end{aligned}$$

which contradicts (*3).

Case 2.2.2: $p_3 \geq 2p_2$.

In this case, we first suppose that $W[p_1] = X_1[p_1] = \mathbf{0}$. We have $W[p_1 + 1] = X_1[p_1 + 1] = X_1[p_1] + 1 = \mathbf{1}$ since X_1 is a 90-drome, and this implies $W[2p_2 - p_1] = X_2[-(p_1 + 1)] = X_2[p_1 + 1] + 1 = W[p_1 + 1] + 1 = \mathbf{2}$ since X_2 is a 90-drome. Since X_3 is a 90-drome, we have $W[2p_3 - p_1 + 1] = X_3[-p_1] = X_3[p_1] + 1 = W[p_1] + 1 = \mathbf{1}$, and $W[2p_3 - 2p_2 + p_1 + 1] = X_3[-(2p_2 - p_1)] = X_3[2p_2 - p_1] + 1 = W[2p_2 - p_1] + 1 = \mathbf{3}$ (*4).

Here we use $2p_2 - p_1 \le p_3$ to derive the second equality. However, we have

$$\begin{aligned}
\mathbf{1} &= W[2p_3 - p_1 + 1] \\
&= Y_2[2p_3 - 2p_2 - p_1 + 1] \qquad (\because 2p_3 - p_1 + 1 > 2p_2 = |Y_2|) \\
&= Y_2[-(2p_3 - 2p_2 - p_1 + 1)] \qquad (\because Y_2 \text{ is a palindrome.}) \\
&= W[-(2p_3 - 2p_2 - p_1 + 1)] \\
&= Y_1[-(2p_3 - 2p_2 - p_1 + 1)] \\
&= Y_1[2p_3 - 2p_2 - p_1 + 1] \qquad (\because Y_1 \text{ is a palindrome.}) \\
&= W[2p_3 - 2p_2 + p_1 + 1],
\end{aligned}$$

which contradicts (*4). This completes the proof of Lemma 10. ◀

The following two lemmas are also from [14].

▶ **Lemma 11.** *([14, Lemma 5.1]) Let W be a word with a period p, and X a 90-drome subword of W. Then $|X| \le p$.* ◀

▶ **Lemma 12.** *([14, Lemma 5.7]) Let W be a word. There exists an $O(1)$-sized set $\mathcal{F}$ of factors W such that every admissible palindrome factor with length at least $|W|/3$ is an affix (i.e., a prefix or suffix) factor of an element of $\mathcal{F}$.* ◀

The final statement in this subsection is a famous theorem on integer sequences due to Erdős and Szekeres.

▶ **Theorem 13.** *([4]) Any sequence of $n^2 + 1$ distinct integers has either an increasing or a decreasing subsequence of length $n + 1$.* ◀

3.2 Proof of Theorem 1

Proof of Theorem 1. Given an arbitrary polyomino P, we will show that the number of ways such that $\mathcal{B}(P)$ is factorized into ABC with A a palindrome and B, C 90-dromes as described in Theorem 2 is $O(1)$, which is sufficient to prove Theorem 1. Let $n := |\mathcal{B}(P)|$.

Below we show this for each of two cases (i) $|B|$ or $|C|$ is at least $n/3$, and (ii) $|A|$ is at least $n/3$. The first case follows easily from the lemmas shown in Section 3.1.

Case 1: $|B| \ge n/3$ or $|C| \ge n/3$.

Suppose that $|B| \ge n/3$. (The case $|C| \ge n/3$ is analogous.) Lemma 9 says that there are $O(1)$ possibilities of B.

Fix a 90-drome B. Lemma 10 implies that the number of factorizations of $\mathcal{B}(P)$ including B as an admissible 90-drome factor is at most two. This completes the proof of Case 1.

Case 2: $|A| \ge n/3$.

This case covers the complement of Case 1. Lemma 12 implies that there are $O(1)$ possibilities of the position of the first or the last letter of A in $\mathcal{B}(P)$.

Fix a position of the first letter of A in $\mathcal{B}(P)$. The case for fixing the last letter of A is analogous. Let W be a non-circular word obtained from $\mathcal{B}(P)$ by applying a circular shift if necessary such that A starts at $W[1]$. Let c be a sufficiently large constant whose value will be determined at the end of the proof. Suppose to the contrary that W has at least c factorizations ABC with A a palindrome and B and C 90-dromes as described in Theorem 2. By Lemma 8, for every fixed A, there are at most eight such factorizations. Hence, we can assume that there are at least $c_0 := c/8$ factorizations $W = A_i B_i C_i$ $(1 \le i \le c_0)$ such that A_i starts at $W[1]$ for every $1 \le i \le c_0$, and that all the $|A_i|$s are distinct. Moreover, all the B_is and the C_is are non-empty.

We label these factorizations so that $|A_1| > |A_2| > \cdots > |A_{c_0}|$. Let S_0, S_1 and S_2 be the partition of the indices $\{1, \ldots, c_0\}$ such that $S_0 := \{i : (2/3)n \leq |A_i| \leq n\}$, $S_1 := \{i : (4/9)n \leq |A_i| < (2/3)n\}$ and $S_2 := \{i : n/3 \leq |A_i| < (4/9)n\}$. Choose the largest set among S_0, S_1 and S_2, and relabel the indices in the chosen set as $1, \ldots, c_1$, where $c_1 \geq c_0/3$. Note that this ensures that $|A_{c_1}| \geq (2/3)|A_1|$, which will be needed when we apply Lemma 5.

Let $d_i := |A_i| + |B_i|/2$ which represents the position of the center of B_i in W. We now focus on the d_is. We can assume that the d_is are all different. This is because $d_i = d_j$ with $i < j$ implies that B_i is a subword of B_j having the same center, and hence B_i is not admissible. Now the theorem of Erdős-Szekeres (Theorem 13) guarantees that we can pick a sequence of indices $1 \leq i_1 < i_2 < \cdots < i_{c_2} \leq c_1$ with $c_2 := \lceil \sqrt{c_1} \rceil$ such that $d_{i_1}, d_{i_2}, \ldots, d_{i_{c_2}}$ are sorted in increasing or decreasing order.

Now we divide Case 2 into two subcases depending on the order of $d_{i_1}, d_{i_2}, \ldots, d_{i_{c_2}}$. In what follows, we write $A_{i_k}B_{i_k}C_{i_k}$ as $A_kB_kC_k$ for simplicity (to avoid a double subscript). For $1 \leq k \leq c_2$, let $b_k := |A_k| + |B_k| - |A_1|$, which represents the position of $B_k[-1]$ in W where we count the position of $B_1[1]$ in W as 1.

Case 2.1: $d_1 < d_2 < \cdots < d_{c_2}$.

Notice that $|B_1| = b_1 > b_2 > \cdots > b_{c_2}$. For each $1 \leq k \leq c_2$, let X_k be a 90-drome subword of B_k obtained from B_k by truncating a same number of letters from both sides of B_k so that X_k starts at $W[|A_1|+1]$. Equivalently, $X_k := W[|A_1|+1 : 2|A_k|+|B_k|-|A_1|]$. Note that $X_1 = B_1$ and $X_k \preceq_{\text{pre}} X_{k+1}$ for every $1 \leq k \leq c_2 - 1$.

Suppose that $c_2 \geq 17$. Put $m := |B_1C_1|(= n - |A_1|)$. By Lemma 6, we have $|X_{k+2}| > (3/2)|X_k|$ for every $k \geq 1$. Thus, we can assume that $|B_1| = |X_1| < m/10$. This is because otherwise $|X_{17}| > (3/2)^8(1/10)m > m$ which is impossible. Hence, the center of C_1 is located at the position between $|A_1| + m/2$ and $|A_1| + (m/2 + m/20)$ in W (*5). By noticing that $C_{k+1} \preceq_{\text{suf}} C_k$ for every k, we can apply Corollary 7 to get $|C_9| < (2/3)^4|C_1| \leq (2/3)^4 m < m/5$ which implies $b_9 > (4/5)m$. We also have $|X_9| < m/5$ because otherwise $|X_{17}| > (1/5)(3/2)^4 m > m$ by Lemma 6.

Notice that $A_{i+1} \preceq_{\text{pre}} A_i$ for every $1 \leq i \leq 8$ and $|A_9| \geq (2/3)|A_1|$. By Lemma 5, A_1 has a period at most $(|A_1| - |A_9|)/8$. We have $|A_1| - |A_9| \leq m$ since otherwise $d_9 > d_1 > |A_1|$ implies $b_9 > m$ which is impossible. Hence, A_1 has a period at most $m/8$, and this implies $W[|A_1| + |X_9| + 1 : |A_9| + |B_9|]$ (i.e., a suffix of B_9 succeeding X_9) also has a period at most $m/8$. However, we can pick a 90-drome factor of length greater than $m/8$ inside this interval as a subword of C_1 (and hence a subword of W) by truncating a same number of letters from both side of C_1 since (*5) holds. This contradicts Lemma 11 which completes the proof of Case 2.1.

Case 2.2: $d_1 > d_2 > \cdots > d_{c_2}$.

Notice that, in this case, $d_i > d_j$ *does not* imply $b_i > b_j$. So we first apply Lemma 10 and Theorem 13 to the sequence $b_1, \ldots, b_{c_2}$ to obtain a subsequence of length $c_3 := \lceil \sqrt{c_2/2} \rceil$ such that the selected b_is are sorted in increasing or decreasing order. (Here we first pick $c_2/2$ indices such that the selected b_is are all distinct (the existence of such a set is guaranteed by Lemma 10), and then apply Theorem 13 to get a desired subsequence.) For simplicity, we write the indices of the selected b_is as $1, 2, \ldots, c_3$. That is, $W = A_kB_kC_k$ with A_k a palindrome and B_k and C_k non-empty admissible 90-dromes for $1 \leq k \leq c_3$. Moreover, $b_1, b_2, \ldots, b_{c_3}$ are increasing or decreasing. As to Case 2.1, we put $m := |B_1C_1|(= n - |A_1|)$, and put $b_k := |A_k| + |B_k| - |A_1|$ for $1 \leq k \leq c_3$.

Case 2.2.1: $d_1 > d_2 > \cdots > d_{c_3}$ and $b_1 < b_2 < \cdots < b_{c_3}$

Suppose that $c_3 \geq 13$. Note that $(3/2)^6 > 10$. By Corollary 7 and $|C_1| \leq m$, we have $|C_{13}| < (2/3)^6 m < m/10$, and equivalently $b_{13} > (9/10)m$. As to Case 2.1, for

$1 \le k \le c_3$, let $X_k := W[|A_1|+1 : 2|A_k|+|B_k|-|A_1|]$. In other words, X_k is a 90-drome subword of B_k sharing the center with B_k that starts at $W[|A_1|+1]$. If $d_k \le |A_1|$, $X_k := \emptyset$. We have $|X_{13}| < m/10$ because otherwise Lemma 6 implies $|B_1| = |X_1| > (3/2)^6(1/10)m > m$, which is impossible.

Now we focus on the A_is. Recall that $A_{k+1} \preceq_{\rm pre} A_k$ for every $1 \le k < c_3 - 1$. Then by Lemma 5, A_1 has a period at most $p := (|A_1| - |A_{13}|)/12$. We have $d_{13} \ge |A_1| - p/2$ since $d_{13} < |A_1| - p/2$ implies that we can pick a 90-drome factor of length greater than p which shares the center with B_{13} within $W[1 : |A_1|]$ and this contradicts Lemma 11. Then, we have

$$\frac{|B_{13}|}{2} = d_{13} - |A_{13}| \ge -\frac{p}{2} + (|A_1| - |A_{13}|) = -\frac{p}{2} + 12p \ge \frac{23}{2}p,$$

and hence $m \ge b_{13} = d_{13} + |B_{13}|/2 - |A_1| \ge 11p$, which implies $p \le m/11$. Since B_k is a 90-drome, $W[|A_1| + |X_{13}| + 1 : |A_{13}| + |B_{13}|]$ has a period at most $m/11$.

By recalling that $b_{13} \ge (9/10)m$, we have $|X_{13}| < m/10$. Assume that $|B_1| \le m/2$. Then the center of C_1 is located left to the $|A_1| + (3/4)m$-th letter in W. This guarantees that we can pick a 90-drome factor of length greater than $m/10$ whose center is common to C_1 within $W[|A_1| + |X_{13}| + 1 : |A_{13}| + |B_{13}|]$, which contradicts Lemma 11.

Now we can assume that $|B_1| > m/2$. Then the position of the center of B_1 is located right to the $|A_1| + (1/4)m$-th letter in W. However, in this case, we can pick a 90-drome factor of W sharing the center with B_1 whose length is greater than $m/10$. This contradicts Lemma 11, and thus completes the proof of Case 2.2.1.

Case 2.2.2: $d_1 > d_2 > \cdots > d_{c_3}$ and $b_1 > b_2 > \cdots > b_{c_3}$.

Suppose that $c_3 \ge 19$. By the same argument to the second paragraph of Case 2.2.1, we can show that $|A_1|$ and also $W[|A_1| + |X_{13}| + 1 : |A_{13}| + |B_{13}|]$ has a period at most $m/11$ and $b_{13} \ge 0$. We also have $b_{19} \ge 0$ by a similar argument to this. We have $|B_1| > (9/10)m$, or equivalently $|C_1| < m/10$, because $|C_1| \ge m/10$ implies $|C_{13}| \ge (1/10)(3/2)^6 m > m$ by Corollary 7, which contradicts $b_{13} \ge 0$. Hence the center of B_1 is located at a position between $|A_1| + (9/20)m$ and $|A_1| + m/2$ in W.

Define X_k as to Case 2.2.1. We have $|X_{13}| < m/10$, since otherwise $|B_1| = |X_1| > (1/10)(3/2)^6 m > m$ by Lemma 6. If $|A_{13}| + |B_{13}| \ge |A_1| + (7/10)m$, then there is a 90-drome factor of length greater than $m/10$ within $W[|A_1| + |X_{13}| + 1 : |A_{13}| + |B_{13}|]$ which shares the center with B_1. This contradicts Lemma 11. Hence we can assume $|A_{13}| + |B_{13}| < |A_1| + (7/10)m$ which means that $|C_{13}| > (3/10)m$. However, this implies $|C_{19}| > (3/10)(3/2)^3 m > m$ (by Corollary 7), which contradicts $b_{19} \ge 0$. This completes the proof of Case 2.2.2.

By putting $c \gg 3 \cdot 8 \cdot (2 \cdot 20^2)^2$, we can satisfy all the conditions on c_0, c_1, c_2 and c_3 in the proof, which completes the proof of Theorem 1. ◀

4 Concluding Remarks

In this work, we did not make any effort to optimize the constant in Theorem 1. Currently, the upper bound is dominated by Case 2, which is roughly (the value of c in the proof) × (the size of $\mathcal{F}$ in Lemma 12) × (2 · (representing the choice of $A[1]$ or $A[-1]$ at the beginning of Case 2)). An inspection of the proof of Lemma 12 (in [14, Lemma 5.7]) shows that $O(1)$ in Lemma 12 can be replaced by 34, and hence our upper bound is $\sim 10^9$. Currently, we do not know a polyomino having more than three non-equivalent p4-tilings (see Figure 4 for the one having three p4-tilings). Closing the gap to determine the true value is an interesting future work.

Acknowledgements. The authors would like to thank anonymous referees for their valuable comments, and Yuta Suzuki for creating a computer software to draw the picture of tilings.

References

1 D. Beauquier and M. Nivat. On translating one polyomino to tile the plane, Discrete & Computational Geometry, 6, 575—592 (1991)

2 S. Brlek and X. Provençal. An optimal algorithm for detecting pseudo-squares, In Proc. of DGCI 2006, LNCS 4245, 403–412 (2006)

3 S. Brlek, X. Provençal, and J.M. Fédou. On the tiling by translation problem, Discrete Applied Mathematics, 157, 464—475 (2009)

4 P. Erdős and G. Szekeres, A combinatorial problem in geometry, Compositio Mathematics, 2, 463–470 (1935)

5 N.J. Fine and H.S. Wilf, Uniqueness theorems for periodic functions, Proc. of the American Mathematical Society, 16, 109–114 (1965)

6 H. Fukuda, N. Mutoh, G. Nakamura, and D. Schattschneider, A method to generate polyominoes, polyiamonds and polyhexs for isohedral tilings with rotational symmetry, Graphs and Combinatorics, 23, 259–267 (2007)

7 H. Fukuda, N. Mutoh, G. Nakamura, and D. Schattschneider, Enumeration of polyominoes, polyiamonds and polyhexs for isohedral tilings with rotational symmetry, Symmetry 2011, 3(4), 828–851 (2011)

8 I. Gambini and L. Vuillon, An algorithm for deciding if a polyomino tiles the plane, Theoret. Inform. Appl. 41, 147–155 (2007)

9 S.W. Golomb, Polyominoes, Scribner's (1965)

10 E. Grosswald, Representations of integer as sums of squares, New York: Springer-Verlag (1985)

11 B. Grünbaum and G. C. Shephard, The eighty-one types of isohedral tilings in the plane, Mathematical Proceedings of the Cambridge Philosophical Society, 82(2), 177–196 (1977)

12 Y. Haruyama, The p4-tilings with multiple rotation centers (in Japanese), Master's thesis, Gunma University, Japan (2017)

13 T. Horiyama and M. Samejima, Enumeration of polyominoes for p4 tiling, Proc. of 21st Canadian Conference on Computational Geometry (CCCG 2009), 29–32 (2009)

14 S. Langerman and A. Winslow, A quasilinear-time algorithm for tiling the plane isohedrally with a polyomino, Proc. of 32nd International Symposium on Computational Geometry (SoCG 2016), 50:1–50:15 (2016) (full version at `arXiv:1507.02762`)

15 The on-line encyclopedia of integer sequences, published electronically at `https://oeis.org` (2017)

16 X. Provençal, Combinatoire des mots, géeomémtrie discrèete et pavages, PhD thesis, Université du Québec à Montréal (2008)

17 E.W. Weisstein, "Sum of squares function", from WathWorld–A Wolfram Web Resource, `http://mathworld.wolfram.com/SumofSquaresFunction.html` (2017.9.28 retreived)

18 A. Winslow, An optimal algorithm for tiling the plane with a translated polyomino, Proc. of 26th International Symposium on Algorithms and Computation (ISAAC 2015), LNCS 9472, 3–13 (2015)

Network Optimization on Partitioned Pairs of Points*†

Esther M. Arkin[‡1], Aritra Banik[2], Paz Carmi[3], Gui Citovsky[4], Su Jia[5], Matthew J. Katz[§6], Tyler Mayer[¶7], and Joseph S. B. Mitchell[‖8]

1 **Dept. of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, USA**
`esther.arkin@stonybrook.edu`

2 **Dept. of Computer Science, Ben-Gurion University, Beersheba, Israel**
`aritrabanik@gmail.com`

3 **Dept. of Computer Science, Ben-Gurion University, Beersheba, Israel**
`carmip@gmail.com`

4 **Google, Manhattan, USA**
`gcitovsky@gmail.com`

5 **Dept. of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, USA**
`su.jia@stonybrook.edu`

6 **Dept. of Computer Science, Ben-Gurion University, Beersheba, Israel.**
`matya@cs.bgu.ac.il`

7 **Dept. of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, USA**
`tyler.mayer@stonybrook.edu`

8 **Dept. of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, USA.**
`joseph.mitchell@stonybrook.edu`

Abstract

Given n pairs of points, $\mathcal{S} = \{\{p_1, q_1\}, \{p_2, q_2\}, \ldots, \{p_n, q_n\}\}$, in some metric space, we study the problem of two-coloring the points within each pair, red and blue, to optimize the cost of a pair of node-disjoint networks, one over the red points and one over the blue points. In this paper we consider our network structures to be spanning trees, traveling salesman tours or matchings. We consider several different weight functions computed over the network structures induced, as well as several different objective functions. We show that some of these problems are NP-hard, and provide constant factor approximation algorithms in all cases.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Spanning Tree, Traveling Salesman Tour, Matching, Pairs, NP-hard

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.6

[*] We acknowledge support from the US-Israel Binational Science Foundation (Projects 2010074, 2016116).

[†] A full version of the paper is available at `https://arxiv.org/abs/1710.00876`.

[‡] E. Arkin acknowledges support from the National Science Foundation (CCF-1526406).

[§] M. Katz was partially supported by grant 1884/16 from the Israel Science Foundation.

[¶] T. Mayer's research is partially supported by the NSF (IIS-1247726, CNS-1408695, CNS-1755615, CCF-1439084, CCF-1617618, CCF-BSF-1716252, CCF 1725543, CCF-1526406, CCF-1737939) and by Sandia National Laboratories and NetApp.

[‖] J. Mitchell acknowledges support from the National Science Foundation (CCF-1526406).

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 6; pp. 6:1–6:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Table of results: α is the Steiner ratio and β the best approximation factor of the TSP in the underlying metric space. Unless specified otherwise, all other results in this table apply to general metric spaces.

	min $\lvert f(R)\rvert + \lvert f(B)\rvert$	min-max$\{\lvert\lambda(R)\rvert, \lvert\lambda(B)\rvert\}$	min-max$\{\lvert f(R)\rvert, \lvert f(B)\rvert\}$
Spanning tree	3α	9 3 for $\mathbb{R}$	4α
Matching	2	3	3
TSP tour	3β	18	6β

1 Introduction

We study a class of network optimization problems on *pairs* of sites in a metric space. Our goal is to determine how to split each pair, into a "red" site and a "blue" site, in order to optimize *both* a network on the red sites and a network on the blue sites. In more detail, given n pairs of points, $\mathcal{S} = \{\{p_1, q_1\}, \{p_2, q_2\}, \ldots, \{p_n, q_n\}\}$, in the Euclidean plane or in a general metric space, we define a *feasible coloring* of the points in $S = \bigcup_{i=1}^{n}\{p_i, q_i\}$ to be a coloring, $S = R \cup B$, such that $p_i \in R$ if and only if $q_i \in B$. Among all feasible colorings of $\mathcal{S}$, we seek one which optimizes the cost function over a *pair* of network structures, spanning trees, traveling salesman tours (TSP tours) or matchings, one on the red set and one on the blue set. Let $f(X)$ be a certain structure computed on point set X and let $\lambda(X)$ be the longest edge of a bottleneck structure, $f(X)$, computed on point set X. For each of the aforementioned structures we consider the objective of (over all feasible colorings $S = R \cup B$) minimizing $|f(R)| + |f(B)|$, minimizing $\max\{|f(R)|, |f(B)|\}$ and minimizing $\max\{|\lambda(R)|, |\lambda(B)|\}$. Here, $|\cdot|$ denotes the cost (e.g., sum of edge lengths) of the structure.

The problems we study are natural variants of well-studied network optimization problems. Our motivation comes also from a model of secure connectivity in networks involving facilities with replicated data. Consider a set of facilities each having two (or more) replications of their data; the facilities are associated with pairs of points (or k-tuples of points in the case of higher levels of replication). Our goal may be to compute two networks (a "red" network and a "blue" network) to interconnect the facilities, each network visiting exactly one data site from each facility; for communication connectivity, we would require each network to be a tree, while for servicing facilities with a mobile agent, we would require each network to be a Hamiltonian path/cycle. By keeping the red and blue networks distinct, a malicious attack within one network is isolated from the other.

Our results. We show that several of these problems are NP-hard and give O(1)-approximation algorithms for each of them. Table 1 summarizes our O(1)-approximation results.

Related work. Several optimization problems have been studied of the following sort: Given sets of tuples of points (in a Euclidean space or a general metric space), select exactly one point or at least one point from each tuple in order to optimize a specified objective function on the selected set. Gabow et al. [12] explored the problem in which one is given a directed acyclic graph with a source node s and a terminal node t and a set of k pairs of nodes, where the objective was to determine if there exists a path from s to t that uses at most one node from each pair. Myung et al. [17] introduced the Generalized Minimum Spanning Tree Problem: Given an undirected graph with the nodes partitioned into subsets, compute a minimum spanning tree that uses exactly one point from each subset. They show that this problem is NP-hard and that no constant-factor approximation algorithm exists for this problem unless $P = NP$. Related work addresses the generalized traveling salesperson problem [6, 18, 19, 20], in which a tour must visit one point from each of the given subsets. Arkin et al. [4] studied the problem in which one is given a set V and a set of subsets of V, and one wants to select at least one element from each subset in order to minimize the diameter of the chosen set. They also considered maximizing the minimum distance between any two elements of the chosen set. In another recent paper, Consuegra et al. [8] consider several problems of this kind. Abellanas et al. [1], Das et al. [10] and Khantemouri et al. [15] considered the following problem. Given colored points in the Euclidean plane, find the smallest region of a certain type (e.g., strip, axis-parallel square, etc.) that encloses at least one point from each color. Barba et al. [5] studied the problem in which one is given a set of colored points (of t different colors) in the Euclidean plane and a vector $c = (c_1, c_2, \dots, c_t)$, and the goal is to find a region (axis-aligned rectangle, square, disk) that encloses exactly c_i points of color i for each i. Efficient algorithms are given for deciding whether or not such a region exists for a given c.

While optimization problems of the "one of a set" flavor have been studied extensively, the problems we study here are fundamentally different: we care not just about a single structure (e.g., network) that makes the best "one of a set" choices on, say, pairs of points; we must consider also the cost of a second network on the "leftover" points (one from each pair) *not* chosen. As far as we know, the problem of partitioning points from pairs into two sets in order to optimize objective functions on *both* sets has not been extensively studied. One recent work of Arkin et al. [3] does address optimizing objectives on both sets: Given a set of pairs of points in the Euclidean plane, color the points red and blue so that if one point of a pair is colored red (resp. blue), the other must be colored blue (resp. red). The objective is to optimize the radii of the minimum enclosing disk of the red points and the minimum enclosing disk of the blue points. They studied the objectives of minimizing the sum of the two radii and minimizing the maximum radius.

2 Spanning Trees

Let $MST(X)$ be a minimum spanning tree over the point set X, and $|MST(X)|$ be the cost of the tree, i.e. sum of edge lengths. Let $\lambda(X)$ be the longest edge in a bottleneck spanning tree on point set X and $|\lambda(X)|$ be the cost of that edge. Given n pairs of points in a metric space, find a feasible coloring which minimizes the cost of a pair of spanning trees, one built over each color class.

2.1 Minimum Sum

In this section we consider minimizing $|MST(R)| + |MST(B)|$.

▶ **Theorem 1.** *The Min-Sum 2-MST problem is NP-hard in general metric spaces.* [See full paper [2] for proof.]

An $O(1)$-approximation algorithm for Min-Sum 2-MST problem.

Compute $MST(S)$, a minimum spanning tree on all $2n$ points. Imagine removing the heaviest edge, h, from $MST(S)$. This leaves us with two trees; T_1 and T_2. Perform a preorder traversal on T_1, coloring nodes red as long as there is no conflict. If there is a conflict (q_i is reached in the traversal and p_i was already colored to red) then color the node blue. Repeat this for T_2. We then return the coloring $S = R \cup B$ as our approximate coloring.

Case 1: All nodes in T_1 are of the same color and all nodes in T_2 are of the same color. This partition is optimal. To see this, note that the weight of $MST(S) \setminus \{h\}$ is a lower bound on the cost of the optimal solution as it is the cheapest way to create two trees, the union of which span all of the input nodes. Since each tree is single colored, we know that each tree must have n points, exactly one from each pair, and thus is also feasible to our problem.

Case 2: One tree is multicolored and the other is not. Let OPT be the optimal solution. Suppose without loss of generality that T_1 contains only red nodes and T_2 contains both blue and red nodes. Then, there must be a pair with both nodes in T_2. Imagine also constructing an MST on each color class of an optimal coloring. By definition, in the MSTs built over each color class, at least one point in T_2 must be connected to a point in T_1. This implies that the weight of the optimal solution is at least as large as $|h|$, as h is the cheapest edge which spans the cut (T_1, T_2). Therefore, $|h| \leq |OPT|$.

Consider $MST(R)$. By the Steiner property, we have that an MST over a subset $U \subseteq S$ has weight at most $\alpha|MST(S)|$ where α is the Steiner ratio of the metric space. Recall that $|MST(S) \setminus \{h\}| \leq |OPT|$. In this case, since $|h| \leq |OPT|$, we have that $|MST(R)| \leq \alpha|MST(S)| \leq 2\alpha|OPT|$.

Next, consider building $MST(B)$. Since no blue node exists in T_1, there does not exist an edge that crosses the cut (T_1, T_2) in $MST(B)$, and thus we have that $|MST(B)| \leq \alpha|MST(S) \setminus \{h\}| \leq \alpha|OPT|$. Therefore, $|MST(R) \cup MST(B)| \leq 3\alpha|OPT|$.

Case 3: Both trees are multicolored. In this case, there are two pairs one with both nodes contained in T_1 and one with both nodes contained in T_2. Imagine, again, constructing an MST on each color class in this optimal coloring. In this case, there must be at least two edges crossing the cut (T_1, T_2), one edge belonging to each tree. Note that each of these edges has weight at least $|h|$ as h is the cheapest edge spanning the cut (T_1, T_2), implying that $|h| \leq |OPT|/2$. Thus, $|MST(S)| \leq 1.5|OPT|$ as $|MST(S) \setminus \{h\}| \leq |OPT|$ and $|h| \leq |OPT|/2$.

Using our approximate coloring, one can compute $MST(B)$ and $MST(R)$, each with weight at most $\alpha|MST(S)|$. Therefore $|MST(R) \cup MST(B)| \leq 2\alpha|MST(S)| \leq 3\alpha|OPT|$, where α is again the Steiner ratio of the metric space.

Using the above case analysis, we have the following theorem.

▶ **Theorem 2.** *There exists a 3α-approximation for the Min-Sum 2-MST problem.*

▶ Remark. The Steiner ratio is the supremum of the ratio of length of an minimum spanning tree and a minimum Steiner tree over a point set. In a general metric space $\alpha = 2$ and in the Euclidean plane $\alpha \leq 1.3546$ [14].

2.2 Min-max

In this section the objective is to $\min \max\{|MST(R)|, |MST(B)|\}$.

▶ **Theorem 3.** *The Min-Max 2-MST problem is strongly NP-hard in general metric spaces.*

Proof. The reduction is from a problem which we will call *connected partition* [11]. In *connected partition* one is given a graph $G = (V, E)$, where $|V| = n$, and asked if it is possible to remove a set of edges from G which breaks it into two connected components each of size $n/2$.

Given an instance of *connected partition*, $G = (V, E)$, we will create an instance of min-max 2-MST as follows. For each vertex $v_i \in V$ create an input pair $\{p_i, q_i\}$. For each edge $e = (v_i, v_j) \in E$ set the distance between the corresponding points, p_i and p_j to be one. Set the distances $d(q_i, q_j)$ to be zero for all i, j, and the distances $d(p_i, q_j)$ to be two for all i, j. In order to complete the construction, set all remaining distances to be the shortest path length among the distances defined above.

Claim: G can be partitioned into two connected components of size $n/2$ if and only if there is a solution to the corresponding instance of min-max 2-MST with value $n/2 + 1$.

To show the first direction, suppose that the graph G can be split into two connected components, C_1, C_2, of size exactly $n/2$. Without loss of generality suppose $\{v_i : 1 \leq i \leq n/2\} \in C_1$ and $\{v_j : n/2 < j \leq n\} \in C_2$. Then, it is easy to verify based on the pairwise distances in the metric space described above that the coloring $\{p_i : 1 \leq i \leq n/2\} \cup \{q_j : n/2 < j \leq n\} \in R$, $\{p_i : n/2 < i \leq n\} \cup \{q_j : 1 \leq j \leq n/2\} \in B$, achieves a cost of $n/2 + 1$.

To show the opposite direction, suppose that there is a solution to the instance of min-max 2-MST of cost $n/2 + 1$. Notice that the minimum distance from point p_i to any other point is at least one; therefore, there can be at most $n/2 + 2$ points from the set $P = \{p_i : 1 \leq i \leq n\}$ colored either red or blue in the solution which achieves this cost. Thus there are at least $n/2 - 2$ points from the set $Q = \{q_j : 1 \leq j \leq n\}$ colored either red or blue in this solution in order for it to be a feasible coloring. This implies that there will be at least one edge crossing the cut (P, Q) in both the red and blue MST which realize the cost of this solution, and this edge has cost two. Then, of the remaining budget of $n/2 - 1$ units in order to complete the trees which realize the cost of this solution, it must be the case that we can utilize $n/2 - 1$ edges of length one which interconnect exactly $n/2$ nodes from the set P in each color class.

The edges of length one in our metric space correspond directly to original edges of the graph G in *connected partition* thus showing that there exists two spanning trees each of which spans exactly $n/2$ nodes of G and thus G can be partitioned into two connected components of size exactly $n/2$. ◀

▶ **Theorem 4.** *There exists a 4α-approximation for the Min-Max 2-MST problem.*

Proof. We use the same algorithm as we did for the Min-Sum 2-MST problem. The approximation factor is dominated by case 2 in the Min-Sum 2-MST analysis. For the Min-Max objective function, we have that $\max\{|MST(B)|, |MST(R)|\} \leq \alpha|MST(S)|$ and that $|MST(S)| \leq 4|OPT|$. Thus, $\max\{|MST(B)|, |MST(R)|\} \leq 4\alpha|OPT|$. ◀

2.3 Bottleneck

In this section the objective is to $\min \max\{|\lambda(R)|, |\lambda(B)|\}$.

▶ **Lemma 5.** *Given n pairs of points on a line in $\mathbb{R}^2$ where consecutive points on the line are unit separated, there exists a feasible coloring of the points, such that* $\max\{|\lambda(R)|, |\lambda(B)|\} \leq 3$.

Proof. The proof will be constructive, using Algorithm 1. We partition the points into n disjoint buckets, where a bucket consists of two consecutive points on the line.

Observe that at the end of Algorithm 1, each bucket has exactly one red point and one blue point by construction. Thus, the maximum distance between any two points of the same color is 3. ◀

Algorithm 1: Coloring points on a line.

```
Color the leftmost point, $p$, red
Let $p'$ be the point that is in $p$'s bucket
Let $R$ be a set of red points and $B$ be a set of blue points
$R \leftarrow \{p\}$; $B \leftarrow \emptyset$
while There exists an uncolored point do
    while $p'$ is uncolored do
        if $p$ is red then
            Color $p$'s pair, $q$, blue
            $B \leftarrow B \cup \{q\}$
            $p \leftarrow q$
        else
            Let $p''$ be the point in $p$'s bucket
            Color $p''$ red
            $R \leftarrow R \cup \{p''\}$
            $p \leftarrow p''$
        end
    end
    Find the leftmost uncolored point $x$ and color it red. Let $x'$ be the point in $x$'s
      bucket
    $p \leftarrow x$; $p' \leftarrow x'$
end
return $\{R, B\}$
```

▶ **Theorem 6.** *There exists a 3-approximation algorithm for the Bottleneck 2-MST problem on a line.*

Proof. Note that if the leftmost n points do not contain two points from the same pair, then it is optimal to let R be the leftmost n points and B be the rightmost n points. Suppose now that the leftmost n points contain two points from the same pair. We run Algorithm 1 on the input. Imagine building two bottleneck spanning trees over the approximate coloring as well as over an optimal coloring. Let λ be the longest edge (between two points of the same color) in our solution and λ^* be the longest edge in the optimal solution.

Consider any two consecutive input points s_i and s_{i+1} on the line. We first show that $|\lambda^*| \geq |s_i s_{i+1}|$ by arguing that the optimal solution must have an edge that covers the interval $[s_i, s_{i+1}]$. Suppose to the contrary that no such edge exists. This means that s_i is connected to $n-1$ points only to its left and s_{i+1} is connected to $n-1$ points only to its right. This contradicts the assumption that the leftmost n points contain two points from the same pair.

Let the longest edge in our solution be defined by two points, p_i and p_j. Consider the number of input points in interval $[p_i, p_j]$. Input points in this interval other than p_i and p_j will have a different color than p_i and p_j. It is easy to see that if $[p_i, p_j]$ consists of two input points, that $|\lambda^*| = |\lambda|$, and if $[p_i, p_j]$ consists of three input points, that $|\lambda^*| \geq |\lambda/2|$. We know by lemma 5 that $[p_i, p_j]$ can consist of no more than four input points. In this last case, $|\lambda^*|$ must be at least the length of the longest edge of the three edges in $[p_i, p_j]$. Thus, we see that $|\lambda^*| \geq |\lambda|/3$. ◀

▶ **Theorem 7.** *There exists a 9-approximation algorithm for the Bottleneck 2-MST problem in a metric space.*

Proof. First, we compute $MST(S)$ and consider the heaviest edge, h. The removal of this edge separates the nodes into two connected components, H_1 and H_2. If $\nexists i : p_i, q_i \in H_j$ for $1 \leq i \leq n$ and $1 \leq j \leq 2$, then we let $R = H_1$ and $B = H_2$ and return R and B. Let λ^* be the heaviest edge in the bottleneck spanning trees built on an optimal coloring. Note that $MST(S)$ lexicographically minimizes the weight of the kth heaviest edge, $1 \leq k \leq 2n-1$, among all spanning trees over S, and thus the weight of the heaviest edge in $MST(S) \setminus \{h\}$ is a lower bound on $|\lambda^*|$. Thus, in this case, our solution is clearly feasible and is also optimal as $MST(R)$ and $MST(B)$ are subsets of $MST(S) \setminus \{h\}$.

Now suppose $\exists\, j \ \in \{1,2\} : p_i, q_i \in H_j, 1 \leq i \leq n$. This means that $|\lambda^*| \geq |h|$. In this case, we compute a bottleneck TSP tour on the entire point set. It is known that that a bottleneck TSP tour with bottleneck edge λ can be computed from $MST(S)$ so that $|\lambda| \leq 3|h| \leq 3|\lambda^*|$ [9].

Next we run Algorithm 1 on the TSP tour and return two paths, each having the property that the largest edge has weight no larger than $9|\lambda^*|$. ◀

▶ Remark. Consider the problem of computing a feasible partition which minimizes the bottleneck edge across two bottleneck TSP tours. Let the heaviest edge in the bottleneck TSP tours built on the optimal partition be λ^{**}. The above algorithm gives a 9-approximation to this problem as well because the algorithm returns two Hamilton paths and we know that (using the notation in the above proof) $|\lambda^*| \leq |\lambda^{**}|$. Thus, $|\lambda| \leq 9|\lambda^*| \leq 9|\lambda^{**}|$.

The following is a generalization of Lemma 5. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a set of n k-tuples of points on a line. Each set S_i, $1 \leq i \leq n$, must be colored with k colors. That is, no two points in set S_i can be of the same color.

Consider two consecutive points of the same color, p and q. We show that there exists a polynomial time algorithm that colors the points in S so that the number of input points in interval (p,q) is $O(k)$.

▶ **Lemma 8.** *There exists a polynomial time algorithm to color S so that for any two consecutive input points of the same color, p and q, the interval (p,q) contains at most $2k-2$ input points.*

Proof. The algorithm consists of k steps, where in the jth step, we color n of the yet uncolored points with color j. We describe the first step.

Divide the kn points into n disjoint buckets, each of size k, where the first bucket B_1 consists of the k leftmost points, the second bucket B_2 consists of the points in places $k+1, k+2, \ldots, 2k$, etc. Let $G = (V, E)$ be the bipartite graph, with node set $V = \{\mathcal{S} \cup B = \{B_1, \ldots, B_n\}\}$, in which there is an edge between B_i and S_j if and only if at least one of S_j's points lies in bucket B_i. According to Hall's theorem [13], there exists a perfect matching in G. Let M be such a matching and for each edge $e = (B_i, S_j)$ in M, color one of the points in $B_i \cap S_j$ with color 1. Now, remove from each tuple the point that was colored 1, and remove from each bucket the point that was colored 1. In the second step we color a single point in each bucket with the color 2, by again computing a perfect matching between the buckets (now of size $k-1$) and the $(k-1)$-tuples. It is now easy to see that for any two consecutive points of the same color, p and q, at most $2k-2$ points exist in interval (p,q). ◀

3 Matchings

Let $M(X)$ be the minimum weight matching on point set X and $|M(X)|$ be the cost of the matching. Let $\lambda(X)$ be the longest edge in a bottleneck matching on point set X and $|\lambda(X)|$ be the cost of that edge edge. Given n pairs of points in a metric space, find a feasible coloring which minimizes the cost of a pair of matchings, one built over each color class.

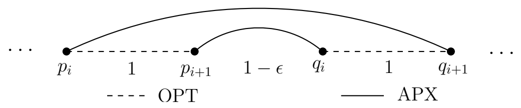


Figure 1 $\frac{|APX|}{|OPT|} \approx 2$.

3.1 Minimum Sum

In this section the objective is to minimize $|M(R)| + |M(B)|$.

▶ **Theorem 9.** *There exists a 2-approximation for the Min-Sum 2-Matching problem in general metric spaces.*

Proof. First, note that the weight of the minimum weight perfect matching on S, M^*, which forbids edges (p_i, q_i) for all i is a lower bound on $|OPT|$. Next, we define the minimum weight one of a pair matching, $\hat{M}$, to be a minimum weight perfect matching which uses exactly one point from each input pair $\{p_i, q_i\}$. Observe that $|\hat{M}|$, is a lower bound on the weight of the smaller of the matchings of OPT and therefore has weight at most $|OPT|/2$.

Our algorithm is to compute $\hat{M}$, and color the points involved in this matching red, and the remainder blue. We return the coloring $R \cup B$ as our approximate solution.

We have that $|M(R)| = |\hat{M}| \leq |OPT|/2$. To bound $|M(B)|$, consider the multigraph $G = (V = S, E = M^* \cup \hat{M})$. All $v \in B$ have degree 1 (from M^*), and all $u \in R$ have degree 2 (from M^* and $\hat{M}$). For each $v_i \in B$, either v_i is matched to $v_j \in B$ by M^*, or v_i is matched to $u_i \in R$ by M^*. In the former case we can consider v_i and v_j matched in B and charge the weight of this edge to $|M^*|$. In the latter case, note that each $u \in R$ is part of a unique cycle, or a unique path. If $u \in R$ is part of a cycle then no vertex in that cycle belongs to B due to the degree constraint. Thus, if $v_i \in B$ is matched to $u_i \in R$, u_i is part of a unique path whose other terminal vertex x belongs to B, due to the degree constraint. We can consider v_i, and x matched and charge the weight of this edge to the unique path connecting v_i and x in G. Thus, $|M(B)|$ can be charged to $|M^* \cup \hat{M}|$ and has weight at most $1.5|OPT|$.

Therefore, our partition guarantees $|M(R)| + |M(B)| \leq 2|OPT|$. Figure 1 shows the approximation factor using our algorithm is tight. ◀

3.2 Min-max

In this section the objective is to $\min \max\{|M(R)|, |M(B)|\}$.

▶ **Theorem 10.** *The Min-Max 2-Matching problem is weakly NP-hard in the Euclidean plane.* [See full paper [2] for proof.]

▶ Remark. The reduction used can be easily modified to also show that the *Min-Max 2-MST problem* is weakly NP-hard in the Euclidean plane.

▶ **Theorem 11.** *The approximation algorithm for the Min-Sum 2-Matching problem serves as a 3-approximation for the Min-Max 2-Matching problem in general metric spaces.* [See full paper [2] for proof.]

3.3 Bottleneck

In this section the objective is to $\min \max\{|\lambda(R)|, |\lambda(B)|\}$.

Algorithm 2: Algorithm $A(\mu, \beta)$. $0 < \mu < 1$ and $\beta > 1$.

```
Let $\overline{TSP_\beta}(X)$ denote a $\beta$-factor approximate TSP tour on set $X$.
1. Compute $\overline{TSP_\beta}(S)$.
2. Let $2k$ be the largest even number not exceeding $(2+\frac{1}{\mu})\beta$. Enumerate all ways of decomposing $\overline{TSP_\beta}(S)$ into $2k$ connected components: for each decomposition, color the nodes from consecutive components red and blue alternately (i.e. color all nodes in component one red, all nodes in component two blue, etc.). If this coloring is infeasible, then skip to the next decomposition; otherwise compute $\overline{TSP_\beta}(R)$ and $\overline{TSP_\beta}(B)$.
3. Compute a random feasible coloring, $S = R \cup B$, and compute $\overline{TSP_\beta}(R)$ and $\overline{TSP_\beta}(B)$.
4. Among all pairs of tours produced in steps 2 and 3, choose the pair of minimum sum.
```

▶ **Theorem 12.** *There exists a 3-approximation to the Bottleneck 2-Matching problem in general metric spaces.* [See full paper [2] for proof]

4 TSP Tours

Let $TSP(X)$ be a TSP tour on point set X and $|TSP(X)|$ be the cost of the tour. Let $\lambda(X)$ be the longest edge in a bottleneck TSP tour on point set X and $|\lambda(X)|$ be the cost of that longest edge. Given n pairs of points in a metric space, find a feasible coloring which minimizes the cost of a pair of TSP tours, one built over each color class.

It is interesting to note the complexity difference emerging here. In prior sections, the structures to be computed on each color class of a feasible coloring were computable exactly in polynomial time. Thus, the decision versions of these problems, which ask if there exists a feasible coloring such that some cost function over the pair of structures is at most k, are easily seen to be in NP. However, when the cost function is over a set of TSP tours or bottleneck TSP tours, this is no longer the case. That is, suppose that a non-deterministic Touring machine could in polynomial time, for a point set S and $k \in \mathbb{R}$, return a coloring for which it claimed the cost of the TSP tours generated over both color classes is at most k. Unless $P = NP$, the verifier cannot in polynomial time confirm that this is a valid solution, and therefore the problem is not in NP. Thus, the problems considered in this section are all NP-hard.

4.1 Minimum Sum

In this section the objective is to minimize $|TSP(R)| + |TSP(B)|$.

We will show for $\beta > 1$ and for the proper choice of μ, that Algorithm 2 gives a 3β-approximation for the Min-Sum 2-TSP problem. Fix a constant $\mu < 1$. Let OPT be the optimal (feasible) coloring $S = R^* \cup B^*$. Let $d(R, B)$ be the minimum point-wise distance between sets R and B. We call an instance of the problem μ-separable if there exists a feasible coloring $S = R \cup B : d(R, B) \geq \mu(|TSP(R)| + |TSP(B)|)$.

Let APX be the coloring returned by our algorithm. We will show that if S is not μ-separable, then $|APX| \leq \frac{2}{1-4\mu}\beta|OPT|$ (see Lemma 13) and that if S is μ-separable, then $|APX| \leq \frac{1}{4\mu}\beta|OPT|$ (see Lemma 14). Supposing both of these are true, then the approximation factor of our algorithm is $\max\{\frac{1}{4\mu}, \frac{2}{1-4\mu}\}\beta$. One can easily verify that $\mu =$

$1/12$ is the minimizer which gives the desired 3β factor. The following lemma states that if S is not μ-separable, then any feasible coloring yields a "good" approximation.

▶ **Lemma 13.** *If S is not μ-separable, then $|APX| \leq \frac{2}{1-4\mu}\beta|OPT|$.*

Proof. If S is not μ-separable, then for any feasible coloring $S = R \cup B$ we have $d(R, B) \leq \mu(|TSP(R)| + |TSP(B)|)$. In particular, for the coloring induced by the optimal solution, $S = R^* \cup B^*$, $d(R^*, B^*) \leq \mu(|TSP(R^*)| + |TSP(B^*)|)$. Then,

$$\begin{aligned}|TSP(S)| &\leq |OPT| + 2d(R^*, B^*)\\ &\leq |OPT| + 2\mu(|TSP(R^*)| + |TSP(B^*)|)\\ &\leq |OPT| + 4\mu|TSP(S)|.\end{aligned}$$

Hence, when $\mu < \frac{1}{4}$, $|TSP(S)| \leq \frac{1}{1-4\mu}|OPT|$. Let $S = \hat{R} \cup \hat{B}$ be the random feasible coloring computed by $\mathcal{A}(\mu, \beta)$. Then, as we are returning the best coloring between $\hat{R} \cup \hat{B}$ and all $O(n^{2k})$ colorings of $\overline{TSP_\beta}(S)$, we have $|APX| \leq \beta(|TSP(\hat{R})| + |TSP(\hat{B})|) \leq 2\beta|TSP(\mathcal{S})| \leq \frac{2\beta}{1-4\mu}|OPT|$. ◀

The following lemma states that if S is μ-separable, then any witness coloring to the μ-separability of S gives a "good" approximation.

▶ **Lemma 14.** *If S is μ-separable, then $|APX| \leq \frac{1}{4\mu}\beta|OPT|$.*

Proof. Suppose we successfully guessed a coloring $X_0 = R^0 \cup B^0$ that is a "witness" to the μ-separability of S (we will show how to guess X_0 later).

- Case 1: $OPT = X_0$. Then $|APX| \leq \beta(|TSP(R^0)| + |TSP(B^0)|) = \beta(|TSP(R^*)| + |TSP(B^*)|) = \beta|OPT|$.
- Case 2: $OPT \neq X_0$. Then $R^* \neq R^0, B^* \neq B^0$ which means each tour in OPT must contain at least 2 edges crossing the cut (R^0, B^0), hence the optimal solution must contain at least 4 edges crossing the cut (R^0, B^0). So $|OPT| \geq 4d(R^0, B^0) \geq 4\mu(|TSP(R^0)| + |TSP(B^0)|) \geq \frac{4\mu}{\beta}|APX|$. Equivalently, $|APX| \leq \frac{\beta}{4\mu}|OPT|$. ◀

The next two lemmas show how to guess a witness coloring X_0 in polynomial time. First, we show that if S is μ-separable with a witness coloring X_0, then $\overline{TSP_\beta}(S)$ cannot cross the red/blue cut defined by this coloring "too many" times.

▶ **Lemma 15.** *Let $\overline{TSP_\beta}(S)$ be an β-factor approximation for $TSP(S)$. Also, suppose S is μ-separable with witness X_0. Then $\overline{TSP_\beta}(S)$ crosses the cut $(R^0,\ B^0)$ at most $(2 + \frac{1}{\mu})\beta$ times.*

Proof. One can construct a TSP tour for S by adding two bridges to $TSP(R^0)$ and $TSP(B^0)$, thus we have $|TSP(S)| \leq |TSP(R^0)| + |TSP(B^0)| + 2d(R^0, B^0) \leq (2 + \frac{1}{\mu})d(R^0, B^0)$. Also, suppose $\overline{TSP_\beta}(S)$ crosses the cut $(R^0,\ B^0)$ $2k$ times. Then, $2kd(R^0, B^0) \leq |\overline{TSP_\beta}(S)| \leq \beta|TSP(S)|$. Combining the above two inequalities, we obtain $2k \leq (2 + \frac{1}{\mu})\beta$. ◀

The next lemma completes our proof.

▶ **Lemma 16.** *Suppose S is μ-separable. Let X_0 be any coloring which serves as a "witness". Then, in step 2 of $\mathcal{A}(\mu, \beta)$, we will encounter X_0 at some stage.*

Proof. Given a nonnegative integer k and a TSP tour P, define $\Pi(P, k)$ =$\{X$: X is a feasible coloring and P crosses X at most k times$\}$. By Lemma 15, we know $X_0 \in \Pi(\overline{TSP_\beta}(S), (2+\frac{1}{\mu})\beta)$. Since step 2 of $\mathcal{A}(\mu, \beta)$ is actually enumerating all colorings in $\Pi(\overline{TSP_\beta}(S), (2+\frac{1}{\mu})\beta)$, this completes the proof. ◀

Note that step 2 considers $O(n^{2k}) = O(n^{14\beta})$ decompositions and for each coloring that is feasible, we compute two approximate TSP tours. Suppose the running time to compute a β-factor TSP tour on n points is $h_\beta(n)$. Then the worst case running time of Algorithm 2 is $O(h_\beta(2n)n^{14\beta})$. Thus, we have the following Theorem.

▶ **Theorem 17.** *For any $\beta > 1$, the algorithm $\mathcal{A}(\frac{1}{12}, \beta)$ is a 3β-approximation for the Min-Sum 2-TSP problem with running time $O(h_\beta(2n)n^{14\beta})$.*

▶ Remark. If S is in the Euclidean plane then $\beta = 1 + \epsilon$ for some $\epsilon > 0$ [16] yielding a $(3 + \epsilon)$-approximation and if S is in a general metric space then $\beta = 3/2$ [7] yielding a 4.5-approximation. In both cases $h_\beta(2n)$ is polynomial.

4.2 Min-Max

In this section the objective is to $\min \max\{|TSP(R)|, |TSP(B)|\}$.

▶ **Theorem 18.** *There exists a 6β-approximation to the Min-Max 2-TSP problem, where β is the approximation factor for TSP in a certain metric space.* [See full paper [2] for proof.]

4.3 Bottleneck

In this section the objective is to $\min \max\{|\lambda(R)|, |\lambda(B)|\}$.

▶ **Theorem 19.** *There exists an 18-approximation algorithm for the Bottleneck 2-TSP problem.* [See full paper [2] for proof.]

References

1 Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. Smallest color-spanning objects. In FriedhelmMeyer auf der Heide, editor, *Algorithms — ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 278–289. Springer Berlin Heidelberg, 2001. doi:10.1007/3-540-44676-1_23.

2 Esther M Arkin, Aritra Banik, Paz Carmi, Gui Citovsky, Su Jia, Matthew J Katz, Tyler Mayer, and Joseph SB Mitchell. Network optimization on partitioned pairs of points. *arXiv, submission 2025570*, 2017. URL: https://arxiv.org/abs/1710.00876.

3 Esther M. Arkin, José M. Díaz-Báñez, Ferran Hurtado, Piyush Kumar, Joseph S. B. Mitchell, Belén Palop, Pablo Pérez-Lantero, Maria Saumell, and Rodrigo I. Silveira. Bichromatic 2-center of pairs of points. *Comput. Geom.*, 48(2):94–107, 2015. doi:10.1016/j.comgeo.2014.08.004.

4 Esther M. Arkin and Refael Hassin. Minimum-diameter covering problems. *Networks*, 36(3):147–155, 2000.

5 Luis Barba, Stephane Durocher, Robert Fraser, Ferran Hurtado, Saeed Mehrabi, Debajyoti Mondal, Jason Morrison, Matthew Skala, and Mohammad Abdul Wahid. On k-enclosing objects in a coloured point set. In *Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG 2013, Waterloo, Ontario, Canada, August 8-10, 2013*. Carleton

University, Ottawa, Canada, 2013. URL: http://cccg.ca/proceedings/2013/papers/paper_35.pdf.

6 Binay K. Bhattacharya, Ante Custic, Akbar Rafiey, Arash Rafiey, and Vladyslav Sokol. Approximation algorithms for generalized MST and TSP in grid clusters. In Zaixin Lu, Donghyun Kim, Weili Wu, Wei Li, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, Houston, TX, USA, December 18-20, 2015, Proceedings*, volume 9486 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2015. doi:10.1007/978-3-319-26626-8_9.

7 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

8 Mario E. Consuegra and Giri Narasimhan. Geometric avatar problems. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 389–400. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.389.

9 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms second edition, 2001.

10 Sandip Das, Partha P. Goswami, and Subhas C. Nandy. Smallest color-spanning object revisited. *International Journal of Computational Geometry and Applications*, 19(05):457–478, 2009. doi:10.1142/S0218195909003076.

11 Martin E Dyer and Alan M Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.

12 Harold N. Gabow, Shachindra N. Maheshwari, and Leon J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 2(3):227–231, 1976.

13 Philip Hall. On representatives of subsets. *J. London Math. Soc*, 10(1):26–30, 1935.

14 Dan Ismailescu and Joseph Park. Improved upper bounds for the steiner ratio. *Discrete Optimization*, 11:22–30, 2014. doi:10.1016/j.disopt.2013.10.004.

15 Payam Khanteimouri, Ali Mohades, MohammadAli Abam, and MohammadReza Kazemi. Computing the smallest color-spanning axis-parallel square. In Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam, editors, *Algorithms and Computation*, volume 8283 of *Lecture Notes in Computer Science*, pages 634–643. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45030-3_59.

16 Joseph SB Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

17 Young-Soo Myung, Chang-ho Lee, and Dong-wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995. doi:10.1002/net.3230260407.

18 Petrica C. Pop. New models of the generalized minimum spanning tree problem. *J. Math. Model. Algorithms*, 3(2):153–166, 2004. doi:10.1023/B:JMMA.0000036579.83218.8d.

19 Petrica C. Pop, Walter Kern, Georg Still, and Ulrich Faigle. Relaxation methods for the generalized minimum spanning tree problem. *Electronic Notes in Discrete Mathematics*, 8:76–79, 2001. doi:10.1016/S1571-0653(05)80085-X.

20 Petr Slavik. *Approximation algorithms for set cover and related problems.* PhD thesis, State University of New York at Buffalo, Buffalo, NY, USA, 1998.

Voronoi Diagrams for Parallel Halflines and Line Segments in Space *

Franz Aurenhammer[1], Bert Jüttler[2], and Günter Paulini[3]

1 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria**
auren@igi.tugraz.at

2 **Institute of Applied Geometry, Johannes Kepler University, Linz, Austria**
bert.juettler@jku.at

3 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria**
guenter.paulini@igi.tugraz.at

Abstract

We consider the Euclidean Voronoi diagram for a set of n parallel halflines in $\mathbb{R}^3$. A relation of this diagram to planar power diagrams is shown, and is used to analyze its geometric and topological properties. Moreover, an easy-to-implement space sweep algorithm is proposed that computes the Voronoi diagram for parallel halflines at logarithmic cost per face. Previously only an approximation algorithm for this problem was known. Our method of construction generalizes to Voronoi diagrams for parallel line segments, and to higher dimensions.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases Voronoi diagram, line segments, space-sweep algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.7

1 Introduction

The Voronoi diagram of a set of sites in Euclidean d-space $\mathbb{R}^d$ is a powerful and widely used geometric partitioning structure. It associates each site with the region of all points in $\mathbb{R}^d$ for which this site is closest among the given set. In the plane $\mathbb{R}^2$, many of the properties of Voronoi diagrams are well understood, also in generalized settings of various kinds; see e.g. [7].

Knowledge becomes quite sparse in dimensions larger than two, when sites of a shape more general than points are allowed. This concerns the structural as well as the algorithmic properties, and is already true for the seemingly harmless generalization from point sites to line segments. The combinatorial complexity of the Voronoi diagram for n line segments, and in particular, for n straight lines in $\mathbb{R}^d$ can be as large as $\Omega(n^{d-1})$, as has been shown in [4]. The only known upper bound follows from a general result in [17] on lower envelopes of hypersurfaces in $\mathbb{R}^{d+1}$ that represent the distance functions to the line segments, and is $O(n^{d+\varepsilon})$ for any $\varepsilon > 0$.

Even in $\mathbb{R}^3$, no better bounds than $\Omega(n^2)$ and $O(n^{3+\varepsilon})$, respectively, are known up to date. This may be partially due to the complicated shape of the arising bisector surfaces.

* This work was supported by Project I 1836-N15, Austria Science Fund (FWF)

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 7; pp. 7:1–7:10
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

They contain, among even more complex components, parabolic and hyperbolic patches, and can lead to a diagram of fairly complicated topological structure. Already for three straight lines as sites, the induced structure gets so intricate that a separate paper has been devoted to its exploration [12].

To make the problem more tractable, several restricted scenarios have been considered. For example, if the line segment sites are confined to have only constantly many orientations [15], then the size of the diagram reduces to $O(n^{2+\varepsilon})$. If, on the other hand, the underlying distance function is polyhedral and convex, then the diagram becomes piecewise-linear. The upper bound then can be tightened to $O(n^2\alpha(n)\log n)$ when the sites are straight lines [9], and to $O(n^{2+\varepsilon})$ when line segments and even constant-sized convex polyhedra are allowed as sites [14]. A practical algorithm for computing the medial axis of a nonconvex polytope in $\mathbb{R}^3$ under a convex polyhedral distance function is given in [3].

In the present note, we discuss a non-trivial special case for the Euclidean distance, namely, the case where all sites are parallel halflines in $\mathbb{R}^3$, being unbounded in the same direction. Apart from the theoretical interest, practical applications arise in certain problems in the drilling industry (mining exploitation, offshore drilling, hydraulics, etc.), as is reported by Adamou [1]. In particular, such Voronoi diagrams serve in the exploration of the nearest layers to avoid collision between wells and identifying unwanted plies. A related problem where these diagrams may be useful is approximate nearest-neighbor searching among a set of parallel line segments in $\mathbb{R}^3$, which has been studied in Emiris et al. [10].

The only known construction algorithm [1, 2] is algebraic and uses a box subdivision process. It produces a topologically correct approximation of the Voronoi diagram for parallel halflines; the runtime naturally depends on the accuracy of the approximation. (A related general approach, for planar Voronoi diagrams, is given in [11].) The algorithm we are going to present is exact, easy to implement (we have implemented and tested it on various inputs), and its runtime is output-sensitive. In particular, it runs at logarithmic cost per diagram face, extracts the correct topology of the diagram *without* manipulating any three-dimensional objects, and its most complex operation is solving a quadratic equation.

Our method is based on the insight that the diagram to be constructed is related to planar power diagrams, which are piecewise linear structures [5]. We describe this correspondence in Section 2, along with its structural implications. On the algorithmic side, a simple space-sweep algorithm is obtained in Section 3. Basically, a power diagram for fixed sites has to be updated under continuous changes of site weights. Section 4 studies the behavior of the trisector curves for the halfline Voronoi diagram, motivated by an attempt to reduce the $O(n^{2+\varepsilon})$ upper bound on its combinatorial complexity (which follows from the aforementioned result in [15]) to $O(n^2)$. Some extensions of our results are mentioned in Section 5. A preliminary version of this paper has appeared in [8].

2 Diagram

We start with a formal definition of the halfline Voronoi diagram. Let $H = \{h_1, \ldots, h_n\}$ be a set of parallel halflines in $\mathbb{R}^3$. We assume that each h_i is vertical (which is no loss of generality), and that each h_i is unbounded in negative z-direction (a restriction which is valid for the application mentioned before [1]). The upper endpoint of h_i is denoted by z_i. We call z_i the *tip* of h_i and, by slight abuse of notation, we will use z_i also to denote the z-coordinate of the tip. The distance of a point $x \in \mathbb{R}^3$ to a halfline h_i is defined as

$$d(x, h_i) = \min\{\delta(x, q) \mid q \in h_i\}$$

where δ denotes the Euclidean distance function. This distance is the normal distance of x to the supporting line, ℓ_i, of h_i, unless $d(x,h_i)$ is attained at the tip z_i. The *region* of a halfline in the Voronoi diagram, $\mathcal{V}(H)$, of H is given by

$$\text{reg}(h_i) = \{x \in \mathbb{R}^3 \mid d(x,h_i) \leq d(x,h_j), \text{for all } j\}.$$

Regions are bounded by *bisectors*, B_{ij}, for pairs of halflines h_i, h_j, which are sets of points equidistant from two halflines. Let the respective tips satisfy $z_i \geq z_j$. Then B_{ij} is composed of three parts: a planar patch, contained in the (vertical) bisecting plane of the lines ℓ_i and ℓ_j, a piece of a parabolic cylinder equidistant from line ℓ_i and point z_j, and another planar patch in the bisecting plane of the points z_i and z_j. In the special case $z_i = z_j$, the bisector B_{ij} is just a vertical plane.

Concerning the parabolic bisector patch, note that its intersection with the (vertical) plane through h_i and h_j is the parabola defined by ℓ_i and z_j. Moreover, its intersection with any horizontal plane is a straight line. That is, the generators of the parabolic bisector patches are horizontal lines, which leads us to conclude a property that eases the analysis of the structure of $\mathcal{V}(H)$.

▶ **Observation 1.** *The intersection of B_{ij} with any horizontal plane is a straight line.*

Denote now by E^Δ the horizontal plane $z = \Delta$, and consider the lines $b_{ij} = B_{ij} \cap E^\Delta$. As bisectors intersect triple-wise in so-called *trisectors*, $t_{ijk} = B_{ij} \cap B_{ik} \cap B_{jk}$, the lines b_{ij}, b_{ik}, and b_{jk} concur in a common point (or are parallel), for any pairwise different indices i, j, k. This implies, by a result in [6], that the line system $(b_{ij})_{1 \leq i < j \leq n}$ is the set of *power lines* defined by n weighted points (that represent n circles) in E^Δ. A more direct argument for this fact follows from the lemma below.

▶ **Lemma 2.** *Consider the point $p_i = \ell_i \cap E^\Delta$, and assign the weight $w_i = -\max\{0, (\Delta - z_i)\}^2$ to it. For any $x \in E^\Delta$, we have $d(x,h_i)^2 = \delta(x,p_i)^2 - w_i$.*

In other words, the squared Euclidean distance of a point $x \in \mathbb{R}^3$ to the halfline h_i is the *power distance* [5] of x to the point p_i with weight w_i.

Proof. If E^Δ lies below z_i then $p_i \in h_i$ and $w_i = 0$, and the assertion is trivial. Otherwise, it follows from the Pythagorean theorem, because h_i is normal to E^Δ. ◀

Notice that the weight but not the position of p_i in E^Δ depends on Δ. By Lemma 2 we have the following geometric relation:

▶ **Theorem 3.** *For all values Δ, the sectional diagram $\mathcal{V}(H) \cap E^\Delta$ is identical to the power diagram of the points $p_1, \ldots, p_n$, for the weights w_i in Lemma 2.*

In particular, if the plane E^Δ lies below all tips then the Euclidean Voronoi diagram of $p_1, \ldots, p_n$ is obtained.

Figure 1 displays the trisector arcs of $\mathcal{V}(H)$ for a set H of 10 halflines. A corresponding sectional power diagram is shown in Figure 2.

Theorem 3 indicates that $\mathcal{V}(H)$ must have an assessable structure, which we study now in more detail. First of all, the weights w_i, when seen as functions $w_i(\Delta)$, are continuous. More precisely, $w_i(\Delta)$ is zero for $\Delta \leq z_i$, and decreases quadratically for $\Delta > z_i$.

We watch the interplay on E^Δ when Δ is increased from $-\infty$ to ∞. The power cells $C_i(\Delta) = \text{reg}(h_i) \cap E^\Delta$ are convex polygons, whose vertices move continuously with Δ. For sufficiently small Δ, each cell $C_i(\Delta)$ is a planar Voronoi region, and therefore is non-empty.

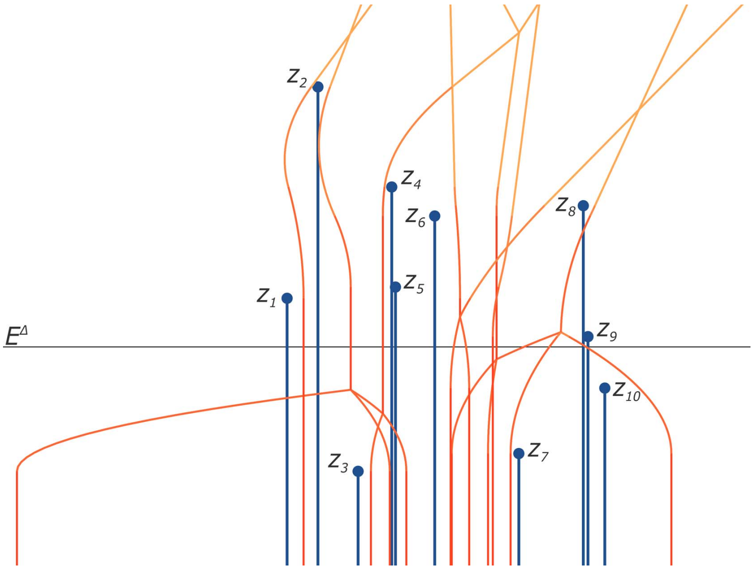


Figure 1 Halfline Voronoi diagram and some sectional plane E^{Δ} (projected view).

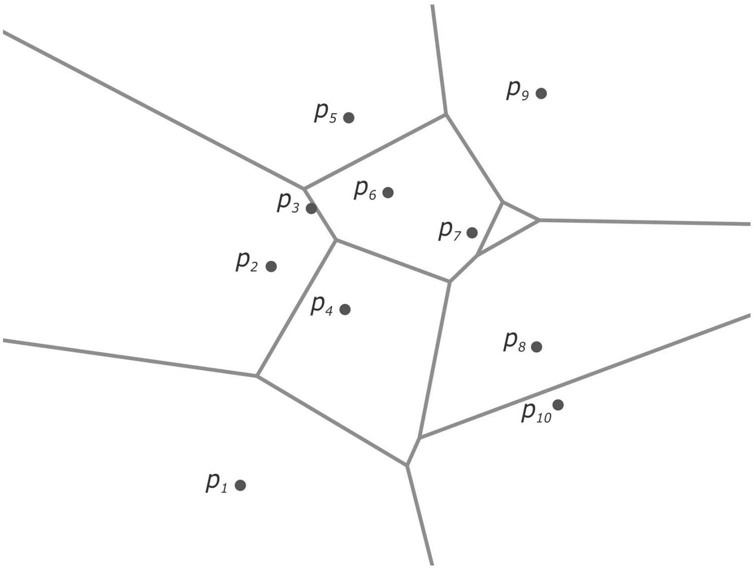


Figure 2 The sectional power diagram that corresponds to Figure 1.

Its edges first poise, and then move self-parallely because $p_1, \ldots, p_n$ stay fixed and any power line for a pair of points p_i, p_j has to be perpendicular to the line segment $\overline{p_i p_j}$. Moreover, the movement of each single edge is always in a fixed direction, by the afore-mentioned shape of a bisector B_{ij}. So each point $x \in E^{\Delta}$ can enter or leave $C_i(\Delta)$ at most once. Also, if $C_i(\Delta)$ disappears from the diagram it cannot reappear, by the monotone movement of its edges. We summarize by stating the following observations on the regions of $\mathcal{V}(H)$.

▶ **Property 4.** *The intersection of reg(h_i) with every vertical line is connected or empty. Moreover, reg(h_i) is a connected set.*

Note that a power cell $C_i(\Delta)$ survives for $\Delta \to \infty$ if and only if the tip z_i appears on the *upper convex hull* of $\{h_1, \ldots, h_n\}$, that is, on that part of the convex hull of the point set $\{z_1, \ldots, z_n\}$ which is visible from $z = \infty$.

Unfortunately, Property 4 does not imply that the combinatorial size of $\text{reg}(h_i)$ is $O(n)$: Although the number of bisectors B_{ij} that border $\text{reg}(h_i)$ is trivially limited to $n-1$, a single bisector may define more than one *facet* (connected boundary patch) of $\text{reg}(h_i)$. Indeed, there are multiple adjacencies between the regions in $\mathcal{V}(H)$ in general; see Section 4.

Let us now have a look at the Voronoi diagram $\mathcal{V}(\{h_i, h_j, h_k\})$ for only three halflines. The trisector curve t_{ijk} corresponds to a power diagram vertex $u^\Delta = t_{ijk} \cap E^\Delta$ for all Δ, unless the three points p_i, p_j, and p_k are collinear. (We exclude the latter case for the ease of description; it leads to $t_{ijk} = \emptyset$). This implies:

▶ **Property 5.** *Each trisector t_{ijk} is a connected curve, unbounded in both z-directions, and monotone with respect to z.*

In particular, t_{ijk} does not contain cycles. For pairwise different tip heights, the curve t_{ijk} is composed of 4 pieces, as can be easily verified: a halfline, two quadratic arcs, and another halfline. Therefore the algebraic degree of t_{ijk} is only two. Still, trisectors show a complicated intersection pattern in general. We will address this issue in Section 4.

3 Algorithm

We now turn to the algorithmic aspects of the halfline Voronoi diagramm $\mathcal{V}(H)$. Theorem 3 suggests a *space-sweep* algorithm that computes $\mathcal{V}(H)$ piece by piece, by moving a horizontal plane E^Δ in ascending z-direction. In fact, an entirely two-dimensional implementation is possible (and has been done by the authors), which avoids computing costly intersections of bisector surfaces in $\mathbb{R}^3$. Once the combinatorial structure of $\mathcal{V}(H)$ has been extracted, the bisector patches and trisector arcs that determine the geometry of $\mathcal{V}(H)$ can be 'filled in' in a final step.

The basic task is to maintain a power diagram for fixed points in the plane, under variation of their weights. The incidence structure of $\mathcal{V}(H)$ then can be inferred from the combinatorial changes that take place in the power diagram: When a power diagram edge appears (or disappears, respectively), then a facet of $\mathcal{V}(H)$ is born (or completed). Moreover, the collapse of a power cell signals the completion of a region in $\mathcal{V}(H)$.

To describe the combinatorial part of the algorithm in more detail, let $\text{PD}(\Delta)$ be the power diagram for the points $p_1, \ldots, p_n$ with weights $w_1(\Delta), \ldots, w_n(\Delta)$, as defined in Section 2. We start with any value $\Delta < \min\{z_1, \ldots, z_n\}$, and initialize $\text{PD}(\Delta)$ as the planar Voronoi diagram of the set $\{p_1, \ldots, p_n\}$.

There is only one type of *events* (z-values) where the power diagram can change. These are the anticipated life ends a_{ij} of its edges e_{ij}.

More specifically, a_{ij} is the z-value of the lowest intersection point above E^Δ of the respective two trisector curves t_{ijk} and t_{ijm}, which define the endpoints of e_{ij}. This value can be calculated in $O(1)$ time, by solving a single quadratic equation in the variable z, for each of the 5 intervals given by z_i, z_j, z_k, z_m. (These intervals determine the weights to be used.) In the diagram $\text{PD}(a_{ij})$, an update of constant complexity has to be performed. This update is either a 'flip' that replaces the edge e_{ij} by the edge e_{km} (and a facet of the halfline Voronoi diagram $\mathcal{V}(H)$ in the bisector B_{ij} gets completed), or a collapse of a triangular cell incident to the edge e_{ij}, say $C_i(a_{ij})$ (and the region $\text{reg}(h_i)$ of $\mathcal{V}(H)$ gets completed).

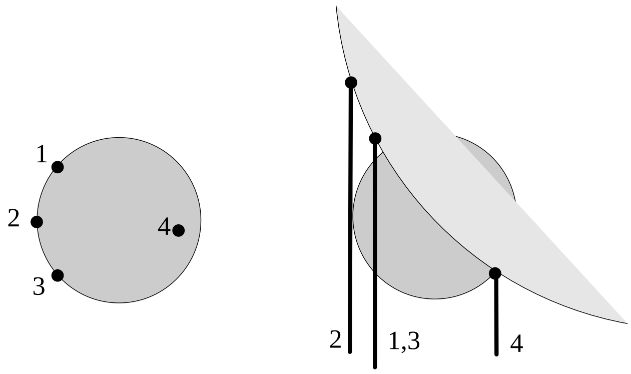


Figure 3 Two spheres touching the same set of 4 halflines. (Thanks go to Peter Widmayer's group for pointing us to this example.)

Notice that the tips z_i of the halflines h_i do not lead to combinatorial changes in $\mathrm{PD}(z_i)$. They only alter the speeds of the edges in the power cell $C_i(z_i)$. This information is already incorporated in the trisector intersection task above.

We use a priority queue organized by z-values to maintain the order of events. Only $O(n)$ entries need to be stored at a time, by the linear number of edges in a power diagram of n weighted points [5]. The next event to be performed then is accessible in $O(\log n)$ time. Moreover, the total number of entries a_{ij} is bounded by the number of facets of $\mathcal{V}(H)$.

Note finally that the numbers of facets, arcs, and nodes of $\mathcal{V}(H)$ are linearly related: A region in $\mathcal{V}(H)$ with f facets has $O(f)$ arcs and nodes by the Euler characteristic, because the degree of its nodes is at least 3. We conclude a main result of this paper:

▶ **Theorem 6.** *The halfline Voronoi diagram $\mathcal{V}(H)$ can be computed in $O(k \log n)$ time and $O(k)$ space, where the number k of faces is bounded by $O(n^{2+\varepsilon})$.*

4 Trisectors

The combinatorial size of $\mathcal{V}(H)$ tends to be near-linear for many data, as has been observed in our experiments. Thus our output-sensitive algorithm in Section 3 can be expected to run fast in practice. On the other hand, $\mathcal{V}(H)$ can attain a complexity of $\Omega(n^2)$, for example, when the tips z_i are arranged like in an $\Theta(n^2)$ worst-case example for the Voronoi diagram of n point sites in $\mathbb{R}^3$; see e.g. [13]. This almost matches the upper bound of $O(n^{2+\varepsilon})$ for $\mathcal{V}(H)$, which follows from the more general bound in [15]; see Section 1. Proving a possible quadratic upper bound is complicated by the fact that the trisector curves of $\mathcal{V}(H)$ do not behave like pseudo-lines. Let us comment on this fact and its consequences.

For the halfline h_L with lowest tip, its region is always convex; all the bisectors B_{Lj} either 'bend' towards h_L or are vertical planes. If the size of $\mathrm{reg}(h_L)$ can be shown to be $O(n)$, then an insertion argument for regions in ascending order of tip heights implies an overall $O(n^2)$ diagram size. Unfortunately, two trisector curves on the boundary of $\mathrm{reg}(h_L)$ can intersect in more than one point, such that the result in [16] on the linear size of surface envelopes does not apply to $\mathrm{reg}(h_L)$.

To see an example with two intersection points, consider four halflines h_1, h_2, h_3, and h_4 arranged as is illustrated in Figure 3, from the top view (left) and from the front view (right), respectively. The two trisector curves t_{123} and t_{124} (and two others) concur in a point x, if and only if there exists a sphere centered at x that simultaneously touches all four halflines. There are two such spheres, a smaller one resting on the tip of the rightmost halfline, and a bigger one passing through all four tips.

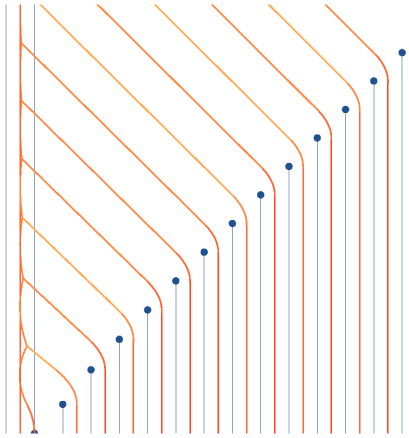


Figure 4 Two very high tips and a tip of height zero on the left, followed by $n-3$ tips of (roughly) linearly increasing heights.

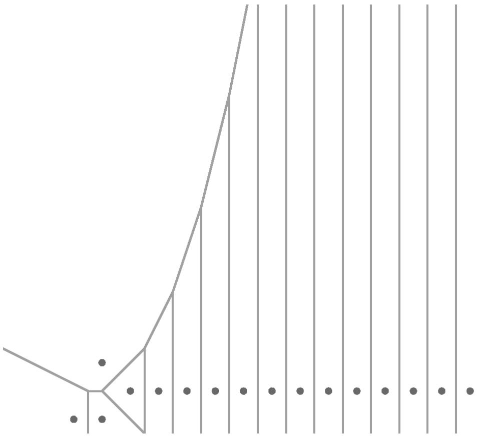


Figure 5 Power diagram for the lowest sectional plane, positioned at the bottom in Figure 4.

The trisectors defined by 4 halflines can have at most 3 intersection points, by a simple algebraic case analysis. This bound is actually attained, and even worse, there are constellations of n halflines for any $n \geq 4$ where every quadruple of related trisectors shows such an intersection behavior. For instance, this happens when the n tips are chosen to lie on the modified moment curve

$$M(t) = \begin{pmatrix} t \\ t^2 \\ tn2^t \end{pmatrix}, \text{ for } 1 \leq t \leq n.$$

As another approach to proving a quadratic upper bound for $\mathcal{V}(H)$, one can try to bound the overall number of edges that appear in the power diagram $\mathrm{PD}(\Delta)$ for varying Δ. This quantity describes the number of facets of $\mathcal{V}(H)$. There are $\binom{n}{2}$ potential power edges. However, once having disappeared, an edge between the same two power cells can appear again. In fact, this can happen a linear number of times: From the trisector pattern in Figure 4 it can be seen that the small horizontal power edge in Figure 5 (left lower corner) will vanish and reappear $n-3$ times when the sectional plane is raised up.

Stated differently, a fixed bisector B_{ij} can define $\Omega(n)$ facets where the two regions $\mathrm{reg}(h_i)$ and $\mathrm{reg}(h_j)$ are adjacent. On the other hand, edge speeds in $\mathrm{PD}(\Delta)$ are not arbitrary: Starting with 0, the speed of an edge increases at constant acceleration, until it stays constant forever. Thus a local multiple appearence of several edges might exclude some related power lines from contributing edges in the future. However, we found an example with 6 points where each of the $\binom{6}{2} - 6 = 9$ bounded power edges they define appears at least twice.

By the relationship between power diagrams and convex hulls (see e.g. [7]), the problem above can be transformed into a dynamic convex hull problem in $\mathbb{R}^3$. Starting from the paraboloid of revolution $z = x^2 + y^2$ at different times, n points move upwards vertically and at constant accelerations. The question of interest is now to bound the number of combinatorial changes that occur on their convex hull.

5 Extensions

The results of this paper can be extended in two different ways. One concerns the Voronoi diagram of parallel line segments that are bounded in *both* directions.

Lemma 2 generalizes straightforwardly to this case, such that Theorem 3 still holds. However, the resulting space-sweep algorithm now has to deal with the detection of new regions because, in general, not all of them will be witnessed by a power cell in the diagram $\mathrm{PD}(\Delta)$ for small Δ. Detection of new power cells cannot be done locally, unless involved data structures are maintained during the updates that occur in $\mathrm{PD}(\Delta)$ when Δ is increased.

A simple solution is to *insert* the input line segments $s_1, s_2, \ldots, s_n$ one by one, thereby using the space-sweep algorithm of Section 3. This is possible, because the region of a segment is always connected, even in the non-parallel case.[1] Initially, for those segments s_j which are unbounded in negative z-direction, the Voronoi diagram is constructed as before. If there are no such segments, we start with the Voronoi diagram of $\{s_1, s_2\}$, which just consists of the bisector of these two segments. An insertion step in the current diagram $\mathcal{V}$ then proceeds as follows:

Let s_i, for $i \geq 3$, be the line segment to be inserted. Choose a value of Δ where the corresponding point p_i has a non-empty cell in the power diagram, $\mathrm{PD}'(\Delta)$, for the weighted points whose segments have been inserted so far. (Clearly, the z-value of the midpoint of s_i is a valid choice.) Construct the power cell $C_i(\Delta)$ by insertion into $\mathrm{PD}'(\Delta)$, in $O(n)$ time. Now, starting from Δ in both z-directions, compute the Voronoi region of s_i and incooporate it into the diagram $\mathcal{V}$, by maintaining $C_i(\Delta)$ during two space sweeps. The parts of $\mathcal{V}$ that have to be deleted in this process can be identified on the fly. By adapting the runtime arguments given in Section 3, we obtain:

▶ **Lemma 7.** *Let f be the number of facets of the region of line segment s_i at the stage of its processing. Then s_i can be inserted in $O(f \log n)$ time.*

While the insertion of a single region thus can be done efficiently, the overall time for constructing the Voronoi diagram for parallel line segments can vary strongly, depending on the shapes of the regions and the insertion order. Unfortunately, no non-trivial upper bounds on the size of a region are known; the combinatorial complexity of the entire diagram is $O(n^{2+\varepsilon})$, by the results in [15]. In particular, the construction algorithm is not output-sentive any more. It still performs quite well in practice (if the search for a starting edge

[1] In particular, each point of a region is visible from its defining segment, like the two-dimensional case.

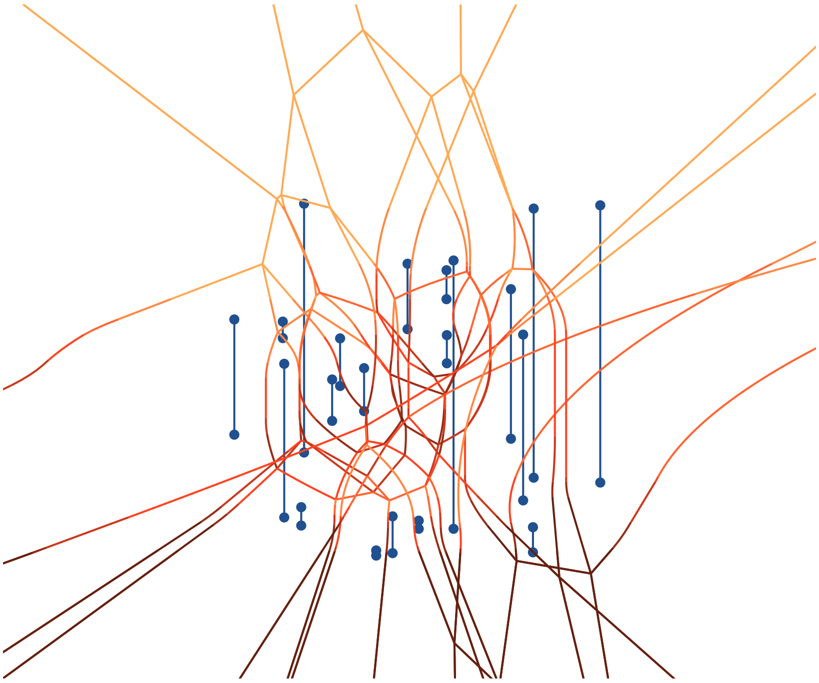


Figure 6 The Voronoi diagram for a set of 20 vertical line segments in 3-space

for $C_i(\Delta)$ is implemented efficiently), as we could observe in our experiments. Figure 6 gives an illustration of the output.

Our method of construction also generalizes to higher dimensions, as the geometric relations in Lemma 2 and Theorem 3 are dimension-independent. For example, to compute the Voronoi diagram of n vertical halflines in $\mathbb{R}^4$, a space-sweep algorithm that maintains a power diagram in a horizontal hyperplane E^Δ of $\mathbb{R}^4$ can be applied. This power diagram is a cell complex consisting of at most n convex 3-dimensional cells. For increasing Δ, the combinatorial changes (events) in this cell complex in E^Δ can be detected by the collapses of its edges. These edges correspond to 2-dimensional faces in the desired 4-dimensional Voronoi diagram, $\mathcal{V}$. That is, the sweep algorithm constructs the diagram $\mathcal{V}$ 2-face by 2-face. Thereby, a simultaneous collapse of 3 edges indicates the completion of a 3-face (facet) of $\mathcal{V}$, and a simultaneous collapse of 6 edges witnesses the completion of a region of $\mathcal{V}$ (in the generic case).

In this way, the topological structure of $\mathcal{V}$ is computed in an output-sensitive manner, namely, in $O(\log n)$ time per 2-face. The same approach works in arbitrary fixed dimensions, where the halfline Voronoi diagram in $\mathbb{R}^d$ can be derived from a dynamically changing power diagram in $\mathbb{R}^{d-1}$. We refrain from describing the details which get more involved rapidly.

▶ **Theorem 8.** *For constant d, let H be a set of n parallel halflines in $\mathbb{R}^d$ that are unbounded in the same direction. If the Voronoi diagram of H is of combinatorial size K, then it can be computed in $O(K \log n)$ time and $O(K)$ space.*

References

1 I. Adamou. Curvas y superficies bisectrices y diagrama de Voronoi de una familia finita de semirrectas paralelas end R^3. Ph.D. thesis, Department of Mathematics, University of Cantabria, Spain, 2013.

2 I. Adamou., M. Fioravanti, L. Gonzalez-Vega, B. Mourrain. Bisectors and Voronoi diagram of a family of parallel half-lines. In: SAGA-Advances in ShApes, Geometry, and Algebra, Springer Verlag, 2014, 241–279.

3 O. Aichholzer, W. Aigner, F. Aurenhammer, and B. Jüttler. Exact medial axis computation for triangulated solids with respect to piecewise linear metrics. *Proc. Curves and Surfaces 2011*, J.-D. Boissonnat et al. (eds.), Springer Lecture Notes in Computer Science 6920, 2011, 1–27.

4 B. Aronov. A lower bound on Voronoi diagram complexity. *Information Processing Letters* 83 (2002), 183–185.

5 F. Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM Journal on Computing* 16 (1987), 78–96.

6 F. Aurenhammer and H. Imai. Geometric relations among Voronoi diagrams. *Geometriae Dedicata* 27 (1988), 65–75.

7 F. Aurenhammer, R, Klein, and D.T. Lee. *Voronoi diagrams and Delaunay Triangulations.* World Scientific, Singapore, 2013.

8 F. Aurenhammer, G. Paulini, and B. Jüttler. Voronoi diagrams for parallel halflines in 3D. In *Proc. 32nd European Workshop on Computational Geometry*, 2016, 127-130.

9 L.P. Chew, K. Kedem, M. Sharir, B. Tagansky, and E. Welzl. Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *J. Algorithms* 29 (1998), 238–255.

10 I. Emiris, T. Malamatos, and E. Tsigaridas. Approximate nearest neighbor queries among parallel segments. *Proc. 26th European Workshop on Computational Geometry*, 2010.

11 I. Emiris, A. Mantzaflaris, B. Mourrain. Voronoi diagrams of algebraic distance fields. *Computer-Aided Design* 45 (2013), 511–516.

12 H. Everett, D. Lazard, S. Lazard, and M. Safey El Din. The Voronoi diagram of three lines. *Discrete & Computational Geometry* 42 (2009), 94–130.

13 S. Fortune. Voronoi diagrams and Delaunay triangulations. In: D.-Z. Du and F.K. Hwang (eds.), *Computing in Euclidean Geometry*, Lecture Notes Series on Computing 1, World Scientific, Singapore, 1992, 193–233.

14 V. Koltun and M. Sharir. Polyhedral Voronoi diagrams of polyhedra in three dimensions. *Discrete & Computational Geometry* 31 (2002), 83–124.

15 V. Koltun and M. Sharir. Three-dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. *SIAM Journal on Computing* 32 (2003), 616–642.

16 J. Schwartz and M. Sharir. On the two-dimensional Davenport-Schinzel problem. *Journal of Symbolic Computation* 10 (1990), 371–393.

17 M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry* 12 (1994), 327–345.

Faster Algorithms for Half-Integral T-Path Packing[*]

Maxim Babenko[1] **and Stepan Artamonov**[2]

1 **National Research University Higher School of Economics, Moscow, Russia**
maxim.babenko@gmail.com

2 **Moscow State University, Moscow, Russia**
stiartamonov@gmail.com

Abstract

Let $G = (V, E)$ be an undirected graph, $T \subseteq V$ be a set of *terminals*. Then a natural combinatorial problem consists in finding the maximum number of vertex-disjoint paths connecting distinct terminals. For this problem, a clever construction suggested by Gallai reduces it to computing a maximum non-bipartite matching and thus gives an $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$-time algorithm (hereinafter $n := |V|$, $m := |E|$).

Now let us consider the fractional relaxation, i.e. allow T-path packings with arbitrary nonnegative real weights. It is known that there always exists a half-integral solution, that is, one only needs to assign weights $0, \frac{1}{2}, 1$ to maximize the total weight of T-paths. It is also known that an optimum half-integral packing can be found in strongly-polynomial time but the actual time bounds are far from being satisfactory.

In this paper we present a novel algorithm that solves the half-integral problem within $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$ time, thus matching the complexities of integral and half-integral versions.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph algorithms

Keywords and phrases graph algorithms, multiflows, path packings, matchings

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.8

1 Introduction

1.1 Basic Notation and Definitions

We shall use some standard graph-theoretic notation throughout the paper. For an undirected graph G, we denote its sets of vertices and edges by $V(G)$ and $E(G)$, resp. A similar notation is used for paths, trees, and etc. A subgraph of G induced by a subset $U \subseteq V(G)$ is denoted by $G[U]$. Let $P = (x_1, \ldots, x_p)$ be a path viewed as a sequence of vertices. By $P[x_i, x_j]$ we denote the part of the path from vertex x_i to vertex x_j. For paths P and Q with a common endpoint, we denote by $P \circ Q$ the concatenation of P and Q.

Let $G = (V, E)$ be an undirected graph, $T \subseteq V$ be a set of *terminals*. By a *T-path* we mean a simple s-t path P connecting two distinct terminals $s, t \in T$ such that all intermediate vertices of P are in $V - T$. A *T-path packing* is a weighted collection $\mathcal{P} = \{\alpha_1 \cdot P_1, \ldots, \alpha_k \cdot P_k\}$ where each P_i is a T-path and each α_i is a non-negative real *weight*. The *value* of a T-path packing $\mathcal{P}$ is $\operatorname{val}(\mathcal{P}) := \alpha_1 + \ldots + \alpha_k$. For $v \in V$, let $\mathcal{P}(v) := \sum (\alpha_i : v \in V(P_i))$ be the total weight of paths in $\mathcal{P}$ containing v. Clearly, $\sum_{t \in T} \mathcal{P}(t) = 2 \operatorname{val}(\mathcal{P})$.

[*] Supported by RFBR grant 16-01-00362.

Let $c\colon V \to \mathbb{Z}_+$ be vertex *capacities.* A T-path packing $\mathcal{P}$ is called *c-feasible* if $\mathcal{P}(v) \leq c(v)$ for all $v \in V$. Then a natural combinatorial problem consists in finding a c-feasible T-path packing $\mathcal{P}$ of maximum value $\operatorname{val}(\mathcal{P})$.

1.2 Prior Art and Our Contribution

The above T-path packing problem has two flavors: one might either require all path weights to be integral or agree to deal with arbitrary nonnegative real weights.

One of the simplest cases is $c \equiv 1$. Then for the integral version of the problem a clever trick due to Gallai [10, Th. 74.1] reduces the path packing problem to finding a maximum matching in an auxiliary (possibly non-bipartite) graph. This implies an $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$-time algorithm [5] (hereinafter $n := |V|$, $m := |E|$). This connection to matchings, in fact, is not surprising since for $T = V$ the problem is equivalent to maximum matchings (either integral or fractional).

For general c, the integral version has a rich and sophisticated combinatorial structure, which was first studied by Mader [7]. Polynomial-time algorithms for this problem appear in, e.g., [8], [9]; however the actual complexity bounds are not satisfactory. In particular, strongly-polynomial approaches seem to be non-combinatorial and rely on general LP solvers.

For the fractional version of the problem, Pap [8] proved the primal *half-integrality*; i.e. there always exists a T-path packing with half-integral weights and maximum value. Due to this, we can double the capacities and search for an integral solution; hereinafter we mostly focus on integral packings.

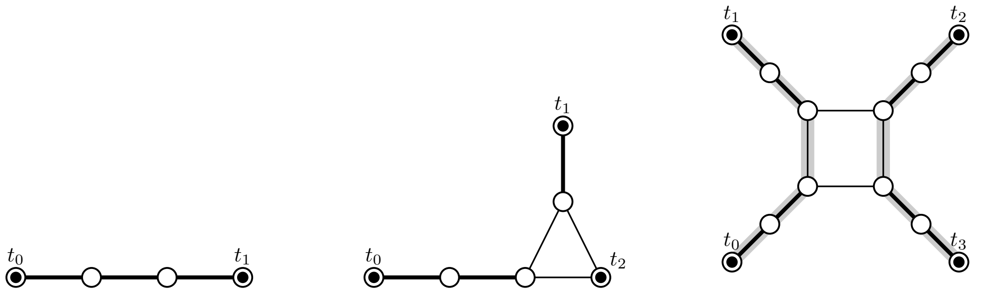
Thus the fractional relaxation of $c \equiv 1$ is exactly $c \equiv 2$. This case seems pretty natural and the existence of the elegant algorithm of Gallai suggests that some efficient direct combinatorial approach might exists here. Indeed, Babenko [1] proposed a combinatorial algorithm that computes a maximum path packing in $O(mn^2)$ time. The latter bound, however, is significantly worse than that of Gallai, which seems weird since typically fractional relaxations of integral problems are easier, both structurally and algorithmically.

Also for arbitrary even capacities, Babenko and Karzanov [3] proposed a weakly polynomial scaling algorithm with complexity $O(\Lambda(n, m, C)n^2 \log^2 n \log C)$, where C is the maximum vertex capacity and $\Lambda(n, m, C)$ denotes the complexity of finding a maximum integral flow in a digraph with n vertices, m edges, and integer edge capacities not exceeding C.

In this paper we present two new faster algorithms for the case $c \equiv 2$ (or, equivalently, the fractional version of the Gallai's problem). The first algorithm employs a simple labeling technique and runs in $O(mn)$ time. The second algorithm is more involved and runs in $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$ time. It relies on the Gallai's approach to construct the initial approximation (a packing with all path weights equal to 2) and then turns it into an optimal solution in linear time; this is where the previous labeling technique appears useful. Note that a conceptually similar approach was employed for computing maximum triangle-free 2-matchings [2].

The rest of the paper is organized as follows. In Section 2 we introduce some basic facts about T-path packings and the Edmonds–Gallai decomposition. Section 3 and Section 4 present the descriptions of $O(mn)$-time and $O(m\sqrt{n}\log\frac{n^2}{m})$-time algorithms, resp. Finally in Section 5 we conclude.

Note that due to the lack of space some proofs are moved into Appendix.



(a) A double path.

(b) An odd star.

(c) Turning an even star into double paths.

Figure 1 Elements of a canonical path packing. Terminal vertices are marked with dots.

2 Preliminaries

2.1 Canonical Decomposition

A 2-feasible T-path packing $\mathcal{P}$ is called *canonical* if it can be represented as a vertex-disjoint union of *elements* of two types: *double paths* and *odd stars*. Both of our algorithms will produce canonical packings.

A *double path* (Fig. 1(a)) is a collection $\mathcal{P}' = \{2 \cdot P\}$ consisting of a T-path P that is taken twice. A *star* (Fig. 1(b)) is a collection $\mathcal{P}'' = \{P_0, \ldots, P_{k-1}\}$ of T-paths that form a star-like shape. The formal definition is rather technical and proceeds as follows. There must be a sequence of distinct terminals $t_0, \ldots, t_{k-1}$ such that P_i connects t_i and t_{i+1} ($i = 0, \ldots, k-1$ and indices are taken modulo k). For each i, P_i and P_{i+1} share a common path incident to t_{i+1}. Such common paths are called *legs*. If a vertex v belongs to both $P_i \in \mathcal{P}''$ and $P_j \in \mathcal{P}''$ then $|i - j| = 1$ and v should belong to the corresponding leg. Vertices covered by a single path (resp. two paths) in $\mathcal{P}''$ are called *inner* (resp. *leg vertices*). Each vertex belonging to a leg is covered by exactly two paths in $\mathcal{P}''$. Edges that are traversed by exactly one P_i in $\mathcal{P}''$ form a simple cycle called the *inner cycle* of the star. A star is called *odd* (resp. *even*) if k is odd (resp. even). Note that each even star can be easily turned into a number of double paths as shown in Fig. 1(c); this preserves the packing value.

In both algorithms we only consider *canonical* path packings. We also stress that canonical elements $\mathcal{P}_1, \ldots, \mathcal{P}_l$ that form $\mathcal{P}$ are vertex-disjoint, i.e. each $v \in V$ is covered by at most one of $\mathcal{P}_1, \ldots, \mathcal{P}_l$. Clearly this is only important for odd stars since overlapping a double path with any other component violates vertex capacities.

2.2 Min-Max Relation

▶ **Lemma 1.** *Let $\mathcal{P}$ be a 2-feasible T-path packing, U be an arbitrary subset of V. Then*

$$\operatorname{val}(\mathcal{P}) \le |T| + |U \cap T| + 2|U \setminus T| - \operatorname{ot}(G - U), \tag{1}$$

where $G - U$ stands for the graph obtained from G by removing vertices in U, and for a subgraph H of G, $\operatorname{ot}(H)$ denotes the number of connected components of H containing exactly one terminal.

The $O(mn)$ algorithm will provide both a T-path packing $\mathcal{P}$ and a set U turning (1) into equality, thus proving the following min-max relation:

▶ **Theorem 2.** *For 2-feasible T-packings $\mathcal{P}$,*

$$\max_{\mathcal{P}} \operatorname{val}(\mathcal{P}) = \min_{U \subseteq V} |T| + |U \cap T| + 2|U \setminus T| - \operatorname{ot}(G - U). \tag{2}$$

2.3 Edmonds–Gallai Decomposition

For the faster algorithm we will need several standard facts concerning maximum matchings (see [6, Ch. 3] for a survey). For an undirected graph $G = (V, E)$, let $\nu(G)$ denote the size of a maximum matching in G and $\operatorname{odd}(G)$ be the number of connected components of G with an odd number of vertices. A graph G is *factor-critical* if for any $v \in V$, $G - v$ admits a perfect matching. For a matching M, a vertex v is called *exposed* if $v \notin V(M)$.

▶ **Theorem 3** (Tutte–Berge, [6, Ch. 3, Th. 3.1.14]). $\nu(G) = \min_{U \subseteq V} \frac{1}{2}\left(|V| + |U| - \operatorname{odd}(G - U)\right)$.

▶ **Theorem 4** (Edmonds–Gallai, [6, Ch. 3, Th. 3.2.1]). *For an undirected graph G, let*

$$\begin{aligned} D &:= \{v \in V \mid \textit{there exists a maximum matching exposing } v\}, \\ A &:= \{v \in V - D \mid v \textit{ is a neighbor of } D\}, \\ C &:= V - (A \cup D). \end{aligned}$$

Then $U := A$ achieves the minimum in the Tutte–Berge formula, and D is the union of the odd connected components of $G[V - A]$. Every connected component of $G[D]$ is factor-critical. Any maximum matching in G induces a perfect matching in $G[C]$ and a matching in $G[V - C]$ that matches all vertices of A to distinct connected components of $G[D]$.

Note that once a maximum matching M in G is found, an Edmonds–Gallai decomposition of G can be constructed in linear time by running a search for an M-augmenting path. Most algorithms that find M yield this decomposition as a by-product.

3 $O(mn)$-time algorithm

The algorithm will be iterative. Each iteration gets the current T-path packing $\mathcal{P}$, which is represented as a set of its canonical elements, and aims to increase $\operatorname{val}(\mathcal{P})$.

Each iteration will assign *labels* from the set $\{*\} \cup T$ to vertices of G, i.e. maintain a partial map $l \colon V \to \{*\} \cup T$. During one iteration, a once labeled vertex never changes its label. Here $*$ is a special label, the remaining labels correspond to terminals.

A vertex v is called *free* if $\mathcal{P}(v) = 0$, and *covered* otherwise. Similarly an edge e is called *free* if there does not exist $P \in \mathcal{P}$ such that $e \in E(P)$. Note that since $\mathcal{P}$ is canonical, if a terminal t is not free then $\mathcal{P}(t) = 2$. For a vertex $v \in V - T$ covered by $\mathcal{P}$, $\mathcal{P}(v) = 1$ if it is an inner vertex of some odd star and $\mathcal{P}(v) = 2$ otherwise.

The outline of the algorithm is the following. We initially set $l(t) := t$ for each free $t \in T$; other vertices are unlabeled. Then, if for the current $\mathcal{P}$ there is no free terminal then $\mathcal{P}$ is obviously maximum. Otherwise, we enumerate all free terminals and run a certain search procedure from each such terminal.

The search assigns labels to the vertices it visits; see Subsection 3.1 for the details. It builds a certain tree; let F_t be this tree for the current terminal t. During this search we may say that we obtained a *breakthrough*, which means that we can modify $\mathcal{P}$ to increase its value in $O(m + n)$ time. The detailed explanation of a breakthrough is given in Subsection 3.2. When the search at the current free terminal finishes and yields no breakthrough, we will ignore this t with its tree F_t and proceed with searches from other free terminals. If no free terminals are left then it will be shown in Subsection 3.3 that the current $\mathcal{P}$ is maximum.

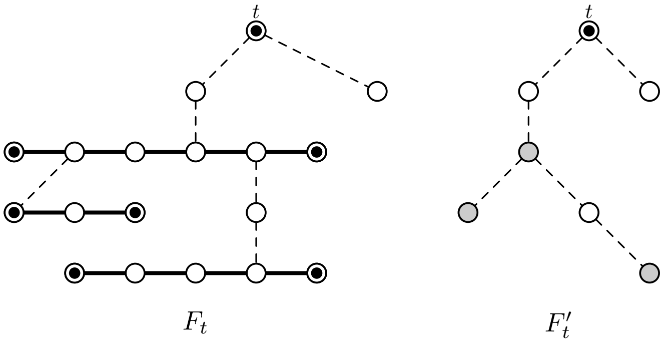


Figure 2 Example of F_t and F'_t. Grayed vertices correspond to contracted double paths.

3.1 Labeling Procedure

Let t be the current free terminal (see above). To start the search from t, we mark t as *active*. We will maintain the property that all active vertices are labeled. Then we run the following labeling procedure.

Let v be an active vertex, $l(v) = \alpha$. The algorithm scans all free edges incident to v. Let $e = \{v, w\}$ be some free edge. We perform a case-splitting as follows:

(A1) $l(w)$ is defined.

(A1-1) $l(w) = \alpha$ or $l(w) = *$. Then edge e is ignored.

(A1-2) $l(w) \neq \alpha$ and $l(w) \neq *$. Then we get a breakthrough.

(A2) $l(w)$ is not defined and w is a free vertex. Then we set $l(w) := \alpha$, make w active and add w together with e to F_t.

(A3) $l(w)$ is not defined and w is covered by a double path $P = (x_0 = \beta, \dots, x_k = \gamma)$ (viewed as a sequence of vertices). Here $\beta, \gamma \in T$. Let $w = x_j$. Then we set $l(x_i) := \beta$ for all $i < j$, $l(x_i) := \gamma$ for all $i > j$, and $l(x_j) := *$, mark all these labeled vertices except for x_j as active, and add the whole path P together with the edge e to the tree F_t.

(A4) $l(w)$ is not defined and w belongs to an odd star S. Then we get a breakthrough.

When all free edges incident to v are examined, v becomes inactive.

Note that once the algorithm reaches a double path P in the current $\mathcal{P}$, it immediately labels all of its vertices. In particular, $*$-vertices are only present on reached paths P, and for each such path there is a unique such vertex; note that this vertex might be a terminal.

Consider the tree F_t rooted at t and contract each double path P that belongs to F_t into a new vertex $v(P)$. Then we obtain a tree F'_t where each non-root vertex corresponds to either a double path or a free non-terminal vertex; see Fig. 2.

Note that by choosing the order in which we examine active vertices and scan edges incident to them, we may assume that this tree F'_t is a DFS-tree, that is, if there is an edge $\{x, y\}$ outside the tree between two distinct vertices x, y of the tree then x is an ancestor of y or vice versa. To achieve this property, we run the usual DFS from the terminal t; the only difference with the standard DFS is that when examining an edge e triggers case (A3), all vertices that become active are added to the DFS stack.

We note that all vertices that were labeled during the current search from t belong to F_t, and if for some edge e case (A1-2) applies then it either connects F_t with a free terminal distinct from t or the image of this edge in F'_t connects two vertices which either coincide or one is an ancestor of the other; note that loops are also possible here as there might be an edge outside F_t connecting two vertices that belong to one double path.

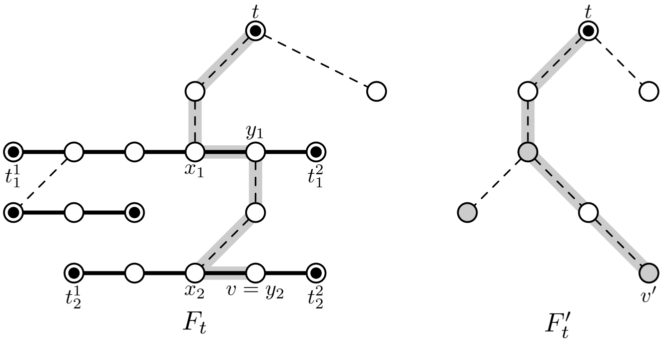


Figure 3 An example of trees F_t and F'_t. Grayed edges indicate paths Q in F_t and Q' in F'_t.

3.2 Breakthrough

Suppose that at some point during an iteration we obtained a breakthrough. We now prove that in this case we can augment the current T-path packing $\mathcal{P}$. Let t be the free terminal that is currently being evaluated, F_t be the tree that grows from it, F'_t be the tree obtained from F_t after contracting each double path. For trees F'_t containing at least one contracted vertex, we define the following operation that alters $\mathcal{P}$ preserving its size but making the terminal t covered and exposing some of the earlier covered terminals.

Let v be some vertex in F_t, v' be its image in F'_t, Q be the (unique) t–v path in F_t, and Q' be the image of Q in F'_t. Let $P_1, P_2, \dots, P_k$ be the sequence of double paths which appear as contracted vertices $p_1, p_2, \dots, p_k$ in Q' when travelling from t to v', and denote by t_i^1 and t_i^2 the endpoints of P_i. Path Q intersects each P_i by a segment; let x_i and y_i be the first and the last (resp.) vertices of this segment (assuming that Q is directed from t to v). Note that vertices x_i have label $*$. Also note that $x_i \neq y_i$ as we do not scan edges from x_i. See Fig. 3 for an example.

Note that for each P_i we may exchange t_i^1 with t_i^2; we shall be using this idea extensively to simplify the case-splitting.

The cornerstone is the following switching routine. Fix $i \in \{1, \dots, k\}$ and an arbitrary endpoint of P_i, say t_i^2. Then we can alter $\mathcal{P}$ (namely, change $P_1, \dots, P_i$) so that t becomes covered, all vertices between t_i^2 and x_i (excluding x_i) in P_i are made free, and the rest of the terminals do not change their covering status. We call this $\textsc{expose}(t_i^2)$.

To prove this, we apply induction on i. If $i = 1$ then we replace P_1 with $P_1[t_1^1, x_1] \circ Q[x_1, t]$. Now let $i > 1$. Exchanging t_{i-1}^1, t_{i-1}^2 if needed, we may assume that $P_{i-1}[t_{i-1}^1, y_{i-1}]$ contains x_{i-1}. We first invoke $\textsc{expose}(t_{i-1}^2)$ to update $P_1, \dots, P_{i-1}$ and make t_{i-1}^2 free. Then we replace P_i with $P_i[t_i^1, x_i] \circ Q[x_i, y_{i-1}] \circ P_{i-1}[y_{i-1}, t_{i-1}^2]$. It is straightforward to see that the resulting double paths do not intersect and cover the desired set of vertices.

Now, recall that we can get the breakthrough in cases (A1-2) and (A4). Let v be the current active vertex and $e = \{v, w\}$ be the edge triggering the breakthrough. We keep the above notation for Q, P_i, t_i^1, t_i^2, x_i, y_i. Exchanging t_i^1, t_i^2 if needed, we may assume that $P_i[t_i^1, y_i]$ contains x_i for all $i \in \{1, \dots, k\}$.

3.2.1 Case A1-2

Note that w can either be a free terminal or a vertex in F_t since the only other labeled vertices are the ones that belong to the trees $F_{t'}$ for free terminals t' that were already scanned, but in that case e should have already been visited while examining w and would have either led to a breakthrough or to vertex v being added to $F_{t'}$, which did not happen.

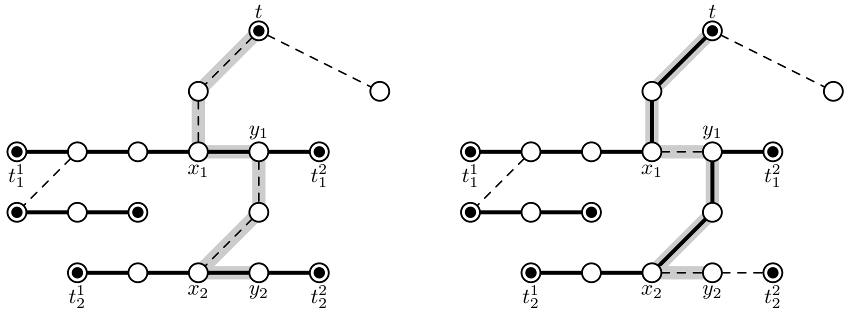


Figure 4 An example of EXPOSE(t_2^2) call. Grayed edges indicate paths Q in F_t and Q' in F_t'.

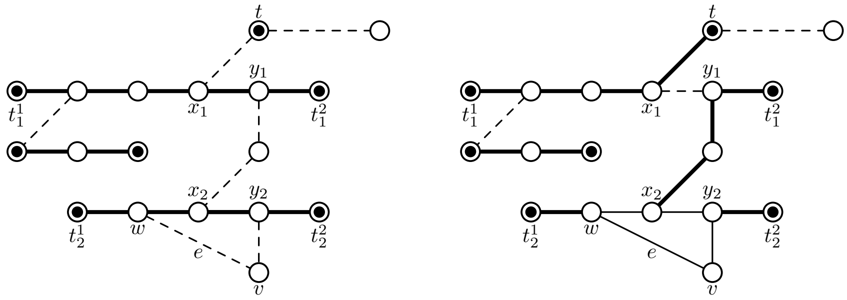


Figure 5 Subcase 2a ($k > 1$).

We thus consider these two cases.

1. w is a free terminal (other than t). If $k = 0$ then we just add $Q \circ e$ as a new double path to $\mathcal{P}$. Otherwise, we first apply EXPOSE(t_k^2) and then add $P_k[t_k^2, y_k] \circ Q[y_k, v] \circ e$ as a new double path.
2. w belongs to F_t. As mentioned above, F_t' is a DFS-tree, so the image of e in F_t' is either a loop or connects a vertex with its ancestor; it might be a loop if both v and w belong to the same double path P_k (in this case $P_k[v, w]$ contains x_k as $l(v) \neq l(w)$). We shall apply EXPOSE to alter some double paths in $\mathcal{P}$ and turn the rest into an odd star. Namely, we have the following possible subcases for w:
   - **a.** w belongs to P_k. We first note that y_k and v have the same label. Then, note that w belongs to $P_k[t_k^1, x_k]$ since otherwise w and y_k and thus v would have had the same label. If $k = 1$ then we form an odd star with legs $Q[t, x_1]$, $P_1[t_1^1, w]$, $P_1[t_1^2, y_1]$ and the inner cycle $P_1[w, y_1] \circ Q[y_1, v] \circ e$. Otherwise we apply EXPOSE(t_{k-1}^2) and create a similar odd star with legs $P_{k-1}[t_{k-1}^2, y_{k-1}] \circ Q[y_{k-1}, x_k]$, $P_k[t_k^1, w]$, $P_k[t_k^2, y_k]$ and the inner cycle $P_k[w, y_k] \circ Q[y_k, v] \circ e$, see Fig. 5.
   - **b.** w belongs to P_i for $i < k$ and x_i does not belong to $P_i[w, y_i]$. We apply EXPOSE(t_i^2) and create an odd star S with the inner cycle $P_i[w, y_i] \circ Q[y_i, v] \circ e$. The legs of S are formed as follows. For endpoint t_j^q of P_j for $j \in \{i+1, \ldots, k\}$, $q \in \{1, 2\}$, we take the part of P_j from t_j^q to the closest of x_j and y_j. For endpoint t_i^2 of P_i, we take the part of P_i from t_i^2 to the closest of y_i and w, see Fig. 6.
   - **c.** w belongs to P_i for $i < k$ and x_i belongs to $P_i[w, y_i]$. If $i = 1$ then we replace all $P_1, \ldots, P_k$ with an odd star. The inner cycle, the legs for terminals t_j^q for $j \in \{1, \ldots, k\}$, $q \in \{1, 2\}$, and the leg for terminal t are constructed similarly to the previous case; see Fig. 7. Otherwise we apply EXPOSE(t_{i-1}^2) and then again form a similar odd star using $P_{i-1}[t_{i-1}^2, y_{i-1}] \circ Q[y_{i-1}, x_i]$ as its leg from t_{i-1}^2.

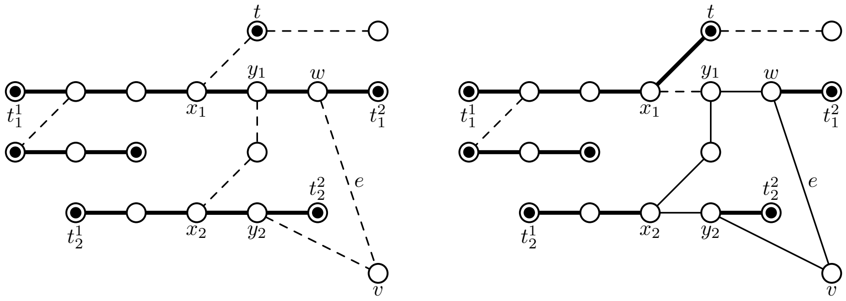


Figure 6 Subcase 2b.

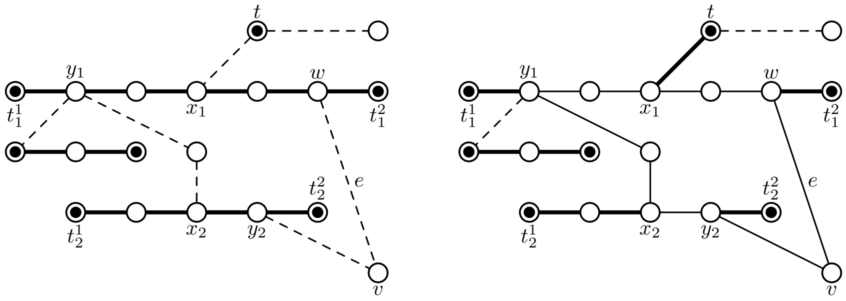


Figure 7 Subcase 2c ($i = 1$).

d. w is a free vertex belonging to F_t. We follow from w to t in F_t until either reaching P_i for some $i \in \{1, \ldots, k\}$ or the root t. In the first subcase the vertex where we stop is exactly y_i. Moreover, $i \neq k$ as $l(v) \neq l(w)$. We apply EXPOSE(t_i^2) and construct an odd star. In the second subcase no EXPOSE call is needed and we just construct an odd star out of $P_1, \ldots, P_k$. In both subcases the inner cycle of this star is $Q[w, v] \circ e$. We omit the details since they are straightforward and very similar to the above.

3.2.2 Case A4

Here we have two cases: either w is an inner vertex of a star S or w belongs to a leg of S.

1. w is an inner vertex of S. Then we apply EXPOSE(t_k^2), attach $P_k[t_k^2, y_k] \circ Q[y_k, v] \circ e$ as a new leg to S thus turning S into an even star, and finally dissolve S into a collection of double paths.
2. w belongs to a leg L of S ending in a terminal t_L. In this case we similarly apply EXPOSE(t_k^2), then remove leg L from S making S an even star, dissolve S into a collection of double paths, and finally add double path $P_k[t_k^2, y_k] \circ Q[y_k, v] \circ e \circ L[w, t_L]$.

Note that in each of the mentioned cases after the transformation all terminals that were covered by $\mathcal{P}$ remain covered by the new path packing, plus t becomes covered, which means that the value of path packing increases by 2, as desired.

3.3 Proof of Maximality

Suppose that we visited all free terminals and did not find a breakthrough. This means that there is no free edge between two labeled vertices with different non-$*$ labels or a non-$*$ labeled and an unlabeled vertex. This in turn means that for each free terminal t and its tree F_t (in which t is the unique free terminal), the only edges that connect F_t with vertices outside F_t have label $*$ on at least one of its ends. Define $U := \{v \mid l(v) = *\}$ and look at the connected components of $G - U$. Let $U' := U \cap T$, $U'' := U \setminus T$. Consider a terminal $s \in T \setminus U$ and suppose that s belongs to some tree F_t. Then in the connected component of $G - U$ containing s there is just one terminal (namely s itself). Indeed, we prove that this component is $G[S]$ for $S := \{v \mid l(v) = s\}$. Suppose first that s is covered by a double path P, and $x \in V(P)$ is the (unique) vertex with $l(x) = *$. In this case $S = V(P[s, x] - x)$, and s is obviously connected to all vertices in S. If there would be an edge connecting S with the rest of $G - U$, it would have to go either to a different tree, or to a vertex with a different label, or to an unlabeled vertex; all these cases are impossible. Thus S is the connected component of $G - U$ containing s and it indeed has just one terminal. Similar reasoning applies if s is a free terminal in F_t, meaning that $s = t$; here S consists of all vertices in F_t for which the path to t in F_t consists solely of free vertices.

Now suppose that we have k free terminals $t_1, \ldots, t_k$. Fix $i \in \{1, \ldots, k\}$ and consider F_{t_i}. Let u'_i (resp. u''_i) be the number of terminal (resp. non-terminal) vertices in this tree that belong to U. Then in F_{t_i} we have one free terminal t_i and $u'_i + 2u''_i$ covered terminals that are the only terminals in their connected components of $G - U$ (see above). Summing over all i, we get $\mathrm{ot}(G - U) \geq k + |U'| + 2|U''|$. We then obtain

$$\mathrm{val}(P) = |T| - k \geq |T| + |U'| + 2|U''| - \mathrm{ot}(G - U) = |T| + |U \cap T| + 2|U \setminus T| - \mathrm{ot}(G - U).$$

This together with (1) proves the min-max relation Theorem 2 and the fact that the current $\mathcal{P}$ is optimal.

3.4 Complexity Analysis

▶ **Lemma 5.** *Each iteration can be implemented to run in $O(m + n)$ time.*

Since each iteration either results in a breakthrough which increases the value of $\mathcal{P}$ or reports that the current $\mathcal{P}$ is maximum, the number of iterations is $O(n)$, thus the total time complexity is $O(mn)$.

4 $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$-time algorithm

Following Gallai [10, Th. 73.1], we build an auxiliary graph $\widehat{G} = (\widehat{V}, \widehat{E})$ from G as follows. For each $v \in V - T$, we make a second copy v' of v. Vertices v and v' are called *mates*. For each edge $\{v, u\}$, we add edge $\{v', u\}$ and also $\{v', u'\}$ if $u \in V - T$. We also add edges $\{v, v'\}$ for each $v \in V - T$.

For each subset $U \subseteq V$, denote by $\widehat{U} \subseteq \widehat{V}$ its *image* in $\widehat{G}$ that consists of all $t \in T \cap U$ and includes both v and v' for all $v \in U \setminus T$. We call a subset $W \subseteq V(\widehat{G})$ *symmetric* if v belongs to W iff v' belongs to W, for each $v \in V - T$. Clearly, symmetric subsets of $\widehat{V}$ are exactly the images of subsets of V.

The outline of the algorithm is the following. First, we construct a maximum matching in $\widehat{G}$ with some special properties. Gallai [10, Th. 73.1] showed that a maximum matching in $\widehat{G}$ provides a maximum packing of T-paths subject to unit capacities; in particular, the number of vertices not covered by this matching in $\widehat{G}$ is equal to the number of terminals that are not covered by a maximum T-path packing.

Recall that we are interested in the case $c \equiv 2$, so after doubling path weights Gallai's packing is not necessarily a maximum one. However this matching gives us an approximation for the optimal solution.

The next step is to augment this approximation using the $O(mn)$ algorithm. We will need to run this algorithm in some subgraphs of G, however in each subgraph we will only need to run one iteration of the algorithm, thus leading to linear time complexity for this step.

4.1 Algorithm Description

First, we compute an arbitrary maximum matching in $\widehat{G}$ and an Edmonds–Gallai decomposition $(\widehat{D}, \widehat{A}, \widehat{C})$ of $\widehat{G}$. Let $\widehat{D_1}, \ldots, \widehat{D_k} \subseteq \widehat{D}$ be the factor-critical connected components of $\widehat{G} - \widehat{A}$. The following lemma justifies the notation and proves that these sets are images of certain sets $D, D_i, A, C \subseteq V$.

▶ **Lemma 6.** *$\widehat{D}, \widehat{D_i}, \widehat{A}, \widehat{C}$ are symmetric subsets of $\widehat{V}$.*

Each component $\widehat{D_i}$ consists of an odd number of vertices and thus contains at least one terminal. A component $\widehat{D_i}$ is called *bad* if D_i contains exactly one terminal, and *good* otherwise. Typically $\widehat{G}$ admits more than just one maximum matching. Our goal is to choose this matching in a proper way, i.e. to cover as many bad components as possible; cf. [2]. To this aim, we first build an auxiliary bipartite graph H with vertices in the *upper* part corresponding to contracted components $\widehat{D_i}$, and $\widehat{A}$ as the the *lower* part. Edges of H correspond to edges of $\widehat{G}$ between $\widehat{A}$ and $\widehat{D}$. Let H' be formed from H by dropping all vertices in the upper part corresponding to good components. We construct a maximum (bipartite) matching M' in H' and then augment it to a maximum matching M in H.

It is widely known that augmenting a matching only increases the set of covered vertices. Also the resulting M is a maximum matching in H and by Theorem 4 this M covers all vertices in $\widehat{A}$.

We convert M into a matching $\widehat{M}$ in $\widehat{G}$ by taking the preimages w.r.t. contractions. Now, for each component $\widehat{D_i}$ covered by M in H, we extend $\widehat{M}$ to cover all vertices of $\widehat{D_i}$ as it is factor-critical. For each good component $\widehat{D_j}$ not covered by M in H, we extend $\widehat{M}$ to cover all vertices of $\widehat{D_j}$ expect for one arbitrarily chosen terminal t_j. We also extend $\widehat{M}$ with some perfect matching in $\widehat{C}$.

Let $\widehat{D'}$ be the union of bad components that are not covered by M in H, T' be the set of terminals belonging to components in $\widehat{D'}$ (one per component). Let T'' be the set of terminals belonging to good components that are not covered by $\widehat{M}$ (again, one per component).

We now turn this matching $\widehat{M}$ into a path packing (subject to $c \equiv 2$) that covers terminals $T - (T' \cup T'')$ (twice); the reader may refer to [10, Th. 73.1]. Let us temporarily remove $\widehat{D'} \cup T''$ from $\widehat{G}$ and the corresponding $D \cup T''$ from G. Then $\widehat{M}$ is a perfect matching. Let N be the set of edges $\{v, v'\} \in \widehat{E}$ for $v \in V - T$. It is straightforward to see that the union $\widehat{M} \cup N$ consists of the following vertex-disjoint components: edges $\{v, v'\}$, simple cycles avoiding terminals, and paths between terminals. After shrinking edges $\{v, v'\}$ in these paths we obtain a collection of disjoint T-paths in G covering all terminals covered by $\widehat{M}$ in $\widehat{G}$. We assign weight 2 to these paths, i.e. regard them as double paths. Let $\mathcal{P}$ be the resulting path packing in G.

The last step is to deal with the terminals T''. Let $\widehat{D_i}$ be a good component not covered by M in H. Then $\widehat{M} \cap E(\widehat{D_i})$ is a nearly-perfect matching in $\widehat{D_i}$, i.e. it exposes exactly one (terminal) vertex $t_i \in D_i \cap T$. Note that the current $\mathcal{P}$ covers all terminals in D_i expect for t_i. Our goal is to adjust $\mathcal{P}$ to cover t_i as well. To accomplish this, we run a search procedure from Section 3 for $G[D_i]$ (using t_i as the starting terminal). This iteration may either find a breakthrough or report that the current packing is maximum. We prove that the latter is not possible, completing the description of the algorithm.

▶ **Lemma 7.** *$\mathcal{P}$ can always be augmented in each good $\widehat{D_i}$.*

Proof. Suppose that the augmentation procedure did not find a breakthrough. Let F_t be the tree obtained by the algorithm, $P_1, \ldots, P_k$ be the set of double paths belonging to F_t. Note that $k > 0$, since otherwise it would mean that t is not connected to the other terminals in D_i, but it must be connected as $G[D_i]$ is factor-critical. Let $U := \{v \mid l(v) = *\}$, $u' = |U \cap T|$, $u'' = |U \setminus T|$. Note that due to the above observation U is not empty. Removing U from D_i gives us at least $u' + 2u'' + 1$ connected components containing exactly one terminal (see Subsection 3.3). Let us now look at the images of these sets in $\widehat{G}$. Note that $|\widehat{U}| = u' + 2u''$. Graph $\widehat{G}[\widehat{D_i} - \widehat{U}]$ has at least $u' + 2u'' + 1$ connected components of odd size. Indeed, consider the subgraph of $\widehat{G}$ induced by $\widehat{V(P_i)}$ for some double path P_i. Let x_i be the unique vertex of P_i with $l(x_i) = *$. If $x_i \in T$ then x_i does not have a mate in $\widehat{D_i}$, and removing it gives one connected component which is symmetric and contains a single terminal (the other endpoint of P_i). If $x_i \notin T$ then removing $\{x_i, x_i'\}$ gives rise to two connected components, each of which is symmetric and contains a single terminal (the respective endpoint of P_i).

Now, recall again that $\widehat{G}[\widehat{D_i}]$ is factor-critical. We pick and remove an arbitrary vertex in $\widehat{U}$; the resulting graph must contain a perfect matching. However, if we proceed and remove the remaining $u' + 2u'' - 1$ vertices of $\widehat{U}$ then the resulting graph would contain $u' + 2u'' + 1$ connected components of odd size, which is a contradiction to Theorem 3. ◀

4.2 Proof of Maximality

▶ **Theorem 8.** *The above algorithm indeed produces a maximum packing $\mathcal{P}$.*

Proof. Consider the maximum matching M' in H' constructed by the algorithm and let L be the corresponding minimum vertex cover in H'. It is straightforward to see that the set $L \cap \widehat{A}$ is symmetric, i.e. is equal to $\widehat{L_A}$ for some $L_A \subseteq A$. Indeed, suppose that some vertex $v \in V - T$ belongs to $L \cap \widehat{A}$ but its mate v' does not. This means that for every edge $e = \{v', w\} \in E(H')$, w should belong to L. But then v can be removed from this vertex cover, contradicting its minimality. The case $v' \in L \cap \widehat{A}$, $v \notin L \cap \widehat{A}$ is symmetric.

Define $L_D := L \setminus \widehat{L_A}$. Let β be the number of bad components $\widehat{D_i}$. Then $\widehat{G} - \widehat{L_A}$ contains at least $\beta - |L_D|$ connected components with just one terminal (each bad component $\widehat{D_i}$ not covered by L can only be connected to $\widehat{L}_A$ and hence separates when $\widehat{L_A}$ gets removed). Therefore $\operatorname{ot}(G - L_A) \geq \beta - |L_D|$.

Note that in the resulting path packing $\mathcal{P}$ exactly $\beta - |M'|$ terminals are not covered. Also $|M'| = |L_D| + |\widehat{L_A}|$ by the max-matching min-cover duality. Then $\operatorname{val}(\mathcal{P}) = |T| - (\beta - |M'|) = |T| - (\beta - |\widehat{L_A}| - |L_D|) \geq |T| + |\widehat{L_A}| - \operatorname{ot}(G - L_A) = |T| + |L_A \cap T| + 2|L_A \setminus T| - \operatorname{ot}(G - L_A)$, which by Theorem 2 implies that $\mathcal{P}$ is maximum. ◀

4.3 Complexity Analysis

Using the algorithm from [5], a maximum matching in $\widehat{G}$ and an Edmonds-Gallai decomposition for $\widehat{G}$ can be constructed in $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$ time. The algorithm from [4], which has the same time bound, can be applied to construct $\widehat{M}$ in the bipartite H, and finally again the matching algorithm for general graphs from [5] can be used to augment $\widehat{M}$ in each $\widehat{D_i}$ which are not covered by M in H.

Turning the union $\widehat{M} \cup N$ into the desired collection of T-paths can obviously be done in $O(m + n)$ time. As for the last step that deals with good components not covered by M

in H, in each such component we need to run exactly one iteration of the first algorithm, thus for a component D_i it takes $O(|V(G[D_i])| + |E(G[D_i])|)$ time. Summing over all such components we get $O(m+n)$ time; hence the total time complexity is $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$.

5 Conclusions

We have presented an efficient algorithm for constructing maximum integral packings of T-path subject to capacities $c \equiv 2$ (or, equivalently, fractional packings subject to $c \equiv 1$). A natural question is if some similar approach could handle the case of arbitrary even capacities and give rise to, e.g., a pseudo-polynomial or a even strongly-polynomial algorithm.

Note that the standard vertex splitting (which replaces each vertex v with capacity $c(v)$ by $c(v)/2$ copies $v_1, \ldots, v_{c(v)/2}$ with capacity 2 each and adjusts the edges appropriately) does not help, since one needs to prevent paths in $\mathcal{P}$ from having (distinct) endpoints to be split-mates that correspond to a single $t \in T$ — a new restriction, which is non-local and is seemingly inexpressible in terms of the Gallai's auxiliary graph.

We suspect that there is a way to extend the labeling approach presented in Section 3 in the needed way and hope to deal with this topic in a subsequent paper.

References

1 M. A. Babenko. A fast algorithm for the path 2-packing problem. *Theory of Computing Systems*, 46(1):59, 2008. URL: `http://dx.doi.org/10.1007/s00224-008-9141-y`, `doi:10.1007/s00224-008-9141-y`.

2 M. A. Babenko, A. Gusakov, and I. P. Razenshteyn. Triangle-free 2-matchings revisited. *Discrete Math., Alg. and Appl.*, 2:643–654, 2010.

3 M. A. Babenko and A. V. Karzanov. Min-cost multiflows in node-capacitated undirected networks. *Journal of Combinatorial Optimization*, 24(3):202–228, 2012. URL: `http://dx.doi.org/10.1007/s10878-011-9377-3`, `doi:10.1007/s10878-011-9377-3`.

4 T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51:261–272, October 1995.

5 A. V. Goldberg and A. V. Karzanov. Maximum skew-symmetric flows and matchings. *Mathematical Programming*, 100(3):537–568, 2004.

6 L. Lovász and M. D. Plummer. *Matching Theory.* Akadémiai Kiadó - North Holland, Budapest, 1986.

7 W. Mader. Über die maximalzahl kantendisjunkter H-wege. *Archiv der Mathematik (Basel)*, 31:382–402, 1978.

8 G. Pap. Some new results on node-capacitated packing of A-paths. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 599–604, New York, NY, USA, 2007. ACM. URL: `http://doi.acm.org/10.1145/1250790.1250878`, `doi:10.1145/1250790.1250878`.

9 G. Pap. Strongly polynomial time solvability of integral and half-integral node-capacitated multiflow problems. *EGRES Technical Report*, TR-2008-12, 2008.

10 A. Schrijver. *Combinatorial Optimization.* Springer, Berlin, 2003.

Shortcuts for the Circle[*]

Sang Won Bae[†1], Mark de Berg[‡2], Otfried Cheong[§3], Joachim Gudmundsson[¶4], and Christos Levcopoulos[5]

1 **Kyonggi University, Suwon, Republic of Korea**
swbae@kgu.ac.kr
2 **TU Eindhoven, Eindhoven, The Netherlands**
mdberg@win.tue.nl
3 **KAIST, Daejon, Republic of Korea**
otfried@kaist.edu
4 **University of Sydney, Sydney, Australia**
joachim.gudmundsson@gmail.com
5 **Lund University, Sweden**
christos@cs.lth.se

Abstract

Let C be the unit circle in $\mathbb{R}^2$. We can view C as a plane graph whose vertices are all the points on C, and the distance between any two points on C is the length of the smaller arc between them. We consider a graph augmentation problem on C, where we want to place $k \geqslant 1$ *shortcuts* on C such that the diameter of the resulting graph is minimized.

We analyze for each k with $1 \leqslant k \leqslant 7$ what the optimal set of shortcuts is. Interestingly, the minimum diameter one can obtain is not a strictly decreasing function of k. For example, with seven shortcuts one cannot obtain a smaller diameter than with six shortcuts. Finally, we prove that the optimal diameter is $2 + \Theta(1/k^{\frac{2}{3}})$ for any k.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Computational geometry, graph augmentation problem, circle, shortcut, diameter

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.9

1 Introduction

Graph augmentation problems have received considerable attention over the years. The goal in such problems is typically to add extra edges to a given graph G in order to improve some quality measure. One natural quality measure is the (vertex- or edge-)connectivity of G. This has led to work where one tries to find the minimum number of edges that can be added to the graph to ensure it is k-connected, for a desired value of k. Another natural measure is the diameter of G, that is, the maximum distance between any pair of vertices.

[*] A full version of this extended abstract is available on the ArXiv [1], https://arxiv.org/abs/1612.02412

[†] SWB is supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057220).

[‡] MdB is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.

[§] OC is supported by NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

[¶] JG is supported under Australian Research Council's Discovery Projects funding scheme (project number DP150101134)

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 9; pp. 9:1–9:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The goal then becomes to reduce the diameter as much as possible by adding a given number of edges, or to achieve a given diameter with a small number of extra edges; see for example the papers by Erdös, Rényi, and Sós [7, 8].

Chung and Garey [6] studied this problem for the special case where the original graph is the n-vertex cycle. They showed that if k edges are added, then the diameter of the resulting graph is at least $\frac{n}{k+2} - 3$ for even k and $\frac{n}{k+1} - 3$ for odd k, and that there is a way to add k edges so that the resulting graph has diameter at most $\frac{n}{k+2} - 1$ for even k and $\frac{n}{k+1} - 1$ for odd k. (For paths, slightly better bounds are known [13].)

The algorithmic problem of finding a set of $k \geqslant 1$ edges that minimizes the diameter of the augmented graph was first asked by Chung [5] in 1987. Since then many papers have considered the problem for general graphs, see [2, 9, 11, 12, 13]. Große et al. [10] were the first to consider the diameter minimization problem in the geometric setting where the graph is embedded in the Euclidean plane. They presented an $O(n \log^3 n)$ time algorithm that determined the optimal shortcut that minimizes the diameter of a polygonal path with n vertices. The running time was later improved to $O(n \log n)$ by Wang [14].

In the above papers only the discrete setting is considered, that is, shortcuts connect two vertices and the diameter is measured between vertices. In the continuous setting all points along the edges of the network are taken into account when placing a shortcut and when measuring distances in the augmented network. In the continuous setting, Yang [15] studied the special case of adding a single shortcut to a polygonal path and gave several approximation algorithms for the problem. De Carufel et al. [4] considered the problem for paths and cycles. For paths they showed that an optimal shortcut can be determined in linear time. For cycles they showed that a single shortcut can never decrease the diameter, while two shortcuts always suffice. They also proved that for convex cycles the optimal pair of shortcuts can be computed in linear time. Recently, Cáceres et al. [3] gave a polynomial time algorithm that can determine whether a plane geometric network admits a reduction of the continuous diameter by adding a single shortcut.

We are interested in a geometric continuous variant of this problem. Let C be a unit circle in the plane. We define the *distance* $d(p, q)$ between two points $p, q \in C$ to be the length of the smaller arc along C that connects p to q. Thus the diameter of C in this metric is π. We now want to add a number of *shortcuts*—a shortcut is a chord of C—to improve the diameter. Here the distance $d_S(p, q)$ between p and q for a given collection S of shortcuts is defined as the length of the shortest path between p and q that can travel along C and along the shortcuts where, if two shortcuts intersect in their interior, we do not allow the path to switch from one shortcut to the other at the intersection point. In other words, if the path uses a shortcut, it has to traverse it completely. Note that if we view the circle C as a graph with infinitely many vertices (namely all points on C) where the graph distance is the distance along C, then adding shortcuts corresponds to adding edges to the graph. For a set S of shortcuts, define $\operatorname{diam}(S) := \max_{p,q \in C} d_S(p, q)$ to be the diameter of the resulting "graph." We are interested in the following question: given k, the number of shortcuts we are allowed to add, what is the best diameter we can achieve? In other words, we are interested in the quantity $\operatorname{diam}(k) := \inf_{|S|=k} \operatorname{diam}(S)$.

It is obvious that $\pi = \operatorname{diam}(0) \geqslant \operatorname{diam}(1) \geqslant \cdots \geqslant \operatorname{diam}(k) \geqslant \cdots \geqslant \lim_{k\to\infty} \operatorname{diam}(k) = 2$. Our main results are as follows.

- For $1 \leqslant k \leqslant 7$, we determine $\operatorname{diam}(k)$ exactly. Our results show that $\operatorname{diam}(k)$ is not strictly decreasing as a function of k. This not only holds at the very beginning—it is easy to see that $\operatorname{diam}(1) = \operatorname{diam}(0)$—but, interestingly also for certain larger values of k. In particular, we show that $\operatorname{diam}(7) = \operatorname{diam}(6)$.

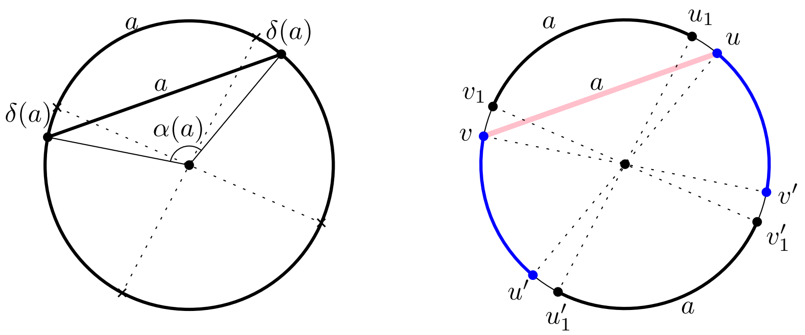


Figure 1 For a shortcut s of length $a = |s|$, (left) $\alpha(a) = a + 2\delta(a)$ and (right) the umbra $U(s)$ (consisting of two arcs of length a in thick black) and radiance (in thick blue).

- We have $\mathrm{diam}(8) < \mathrm{diam}(7)$.
- We show that $\mathrm{diam}(k) = 2 + \Theta(1/k^{\frac{2}{3}})$.

We rely on a number of numerical calculations. A Python script that performs these calculations can be found at `http://github.com/otfried/circle-shortcuts`.

2 The umbra and the region of a shortcut

A shortcut s is a chord of C. A shortcut of length $a = |s| \in [0, 2]$ spans an angle of $\alpha(a) \in [0, \pi]$, where $\alpha(a) := 2 \arcsin\left(\frac{a}{2}\right)$. The following function $\delta : [0, 2] \mapsto [0, \pi/2 - 1]$ will play a key role in our arguments:

$$\delta(a) := \frac{\alpha(a) - a}{2} = \arcsin\left(\frac{a}{2}\right) - \frac{a}{2}.$$

Note that both $\alpha(a)$ and $\delta(a)$ are increasing and convex functions, and $\alpha(a) = a + 2\delta(a)$. See Figure 1. To simplify the notation, we will allow shortcuts themselves as the function argument, with the understanding that $\alpha(s) = \alpha(|s|)$ and $\delta(s) = \delta(|s|)$.

We parameterize the points on the circle C using their polar angle in $[0, 2\pi)$. For a shortcut s with endpoints u and v we will write $s = uv$ if the *counter-clockwise* arc $\widehat{uv}$ is the *shorter* arc of C connecting u and v. Only for $|s| = 2$, we have $s = uv = vu$; in this case u and v are *antipodal points*, that is $v = u + \pi$.

The *inner umbra* of a shortcut $s = uv$ is the arc $\widehat{u_1 v_1}$ where $u_1 = u + \delta(s)$ and $v_1 = v - \delta(s)$. The *outer umbra* is the set of antipodal points of the inner umbra, that is the arc $\widehat{u_1' v_1'}$ where $x' = x + \pi$. Together they form the *umbra* $U(s)$ of s. Since $\alpha(s) = |s| + 2\delta(s)$, the inner and outer umbra have length $|s|$. The *radiance* of s consists of the two arcs $\widehat{vu'}$ and $\widehat{v'u}$. For $|s| = 2$, we cannot distinguish inner and outer umbra, and the radiance consists of two isolated points, see Figure 1.

Let $p \in U(s)$. Then a path going from p to one endpoint of s and traversing the shortcut is at least as long as going directly from p to the other endpoint—so the shortcut is not useful. This gives us the following observation:

▶ **Observation 1.** *Given a set S of shortcuts, if the shortest path γ from p to q uses shortcuts $s_1, s_2, \dots, s_m \in S$ in this order, then $p \notin U(s_1)$ and $q \notin U(s_m)$. If γ traverses s_i from its endpoint u_i to its other endpoint v_i, then $v_i \notin U(s_{i+1})$ and $u_{i+1} \notin U(s_i)$ for $i = 1, \dots, m-1$.*

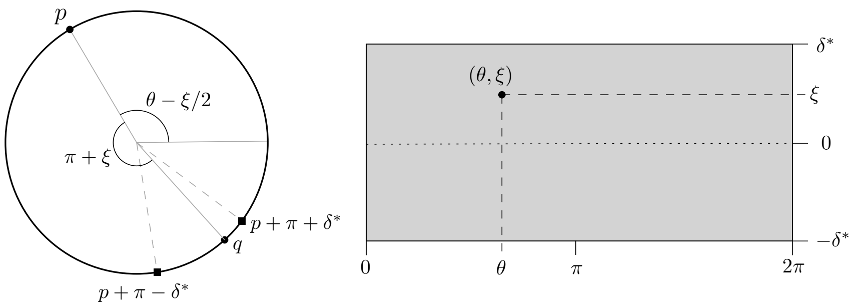


Figure 2 (left) (θ, ξ) corresponds to the pair of points $p = \theta - \xi/2$ and $q = \theta + \pi + \xi/2$. (right) $\mathfrak{S}(\delta^*)$ represents all pair of points $p = \theta - \xi/2$ and $q = \theta + \pi + \xi/2$ with $-\delta^* \leqslant \xi \leqslant \delta^*$.

(For the boundary cases, we will assume that a shortest path uses the minimum number of shortcuts possible.) An immediate implication is that one shortcut alone cannot help to improve the diameter, that is, $\operatorname{diam}(1) = \operatorname{diam}(0) = \pi$.

Another useful observation is the following (remember that $d(p, q) = \min(|\widehat{pq}|, |\widehat{qp}|)$ is the distance along C without shortcuts):

▶ **Observation 2.** *Given a set S of shortcuts, if the shortest path from p to q uses the set of shortcuts $\{s_1, s_2, \ldots, s_m\} \subseteq S$, then $d_S(p, q) \geqslant d(p, q) - 2\sum_{i=1}^{m} \delta(s_i)$.*

Indeed, if γ is the shortest path, we can replace each shortcut s_i by walking along the circle instead, increasing the path length by exactly $2\delta(s_i)$.

Let us now fix a target diameter of the form $\pi - \delta^*$, for some $\delta^* \in [0, \pi - 2]$. To achieve the target diameter, pairs of points $p, q \in C$ that span an angle of at most $\pi - \delta^*$ do not need a shortcut, so it suffices to consider pairs of points $p, q \in C$ where $q = p + \pi + \xi$, for $-\delta^* \leqslant \xi \leqslant \delta^*$. We represent these point pairs by the rectangle $\mathfrak{S}(\delta^*) = [0, 2\pi] \times [-\delta^*, +\delta^*]$, where (θ, ξ) corresponds to the pair of points $p = \theta - \xi/2$ and $q = \theta + \pi + \xi/2$, as illustrated in Figure 2. So the counter-clockwise angle from p to q is $\pi + \xi$.

$\mathfrak{S}(\delta^*)$ is topologically a cylinder: the right edge $\theta = 2\pi$ is identified with the left edge $\theta = 0$. Furthermore, if the point pair $(p, q) \in C \times C$ corresponds to (θ, ξ), then the point pair (q, p) corresponds to $(\theta + \pi, -\xi)$. Since $d_S(p, q) = d_S(q, p)$, we could therefore identify the middle segment $\theta = \pi$ with the left edge $\theta = 0$, but with opposite orientation, resulting in a Möbius strip topology. As will become clear shortly, for our purposes it is easier to work with the cylinder topology, but keep in mind that, for instance, the upper boundary $\xi = \delta^*$ and the lower boundary $\xi = -\delta^*$ of $\mathfrak{S}(\delta^*)$ really represent the same point pairs.

For a shortcut s, we define the region $\mathfrak{R}(s, \delta^*) \subset \mathfrak{S}(\delta^*)$ consisting of those pairs $(\theta, \xi) \in \mathfrak{S}$ where $d_s(\theta - \xi/2, \theta + \pi + \xi/2) \leqslant \pi - \delta^*$. In the following, we will use $d_s(p, q)$ for $d_{\{s\}}(p, q)$.)

Let us fix a shortcut s of length $a > 0$, and let $\alpha = \alpha(a)$ and $\delta = \delta(a)$. Rotating a shortcut around the origin means translating $\mathfrak{R}(s, \delta^*)$ horizontally in (the cylinder) $\mathfrak{S}(\delta^*)$. We can thus choose s to be vertical and connect the points $-\alpha/2$ and $\alpha/2$. This implies that the umbra of s consists of the two intervals $[-\alpha/2 + \delta, \alpha/2 - \delta]$ and $[\pi - \alpha/2 + \delta, \pi + \alpha/2 - \delta]$. The radiance of s consists of the two intervals $[\alpha/2, \pi - \alpha/2]$ and $[\pi + \alpha/2, 2\pi - \alpha/2]$.

The following function gives the length of the path from p to q that uses the shortcut s from top to bottom, that is from the point $\alpha/2$ to $-\alpha/2$:

$$f(\theta, \xi) := |\alpha/2 - p| + a + |q - (2\pi - \alpha/2)|, \qquad \text{where} \quad (p, q) = (\theta - \xi/2, \theta + \pi + \xi/2).$$

By the observation about the Möbius topology above, it suffices to understand $\mathfrak{R}(s, \delta^*)$

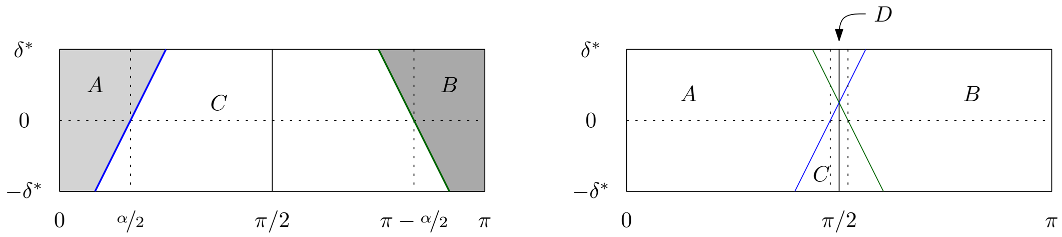


Figure 3 The regions A, B, C, D.

for $0 \leqslant \theta \leqslant \pi$. We claim that for $0 \leqslant \theta \leqslant \pi$ we have $d_s(p,q) < \pi - \delta^*$ if and only if $f(\theta,\xi) < \pi - \delta^*$.

This is clearly true if the shortest path from p to q uses s from top to bottom, or not at all, because the length of the shorter circle arc between p and q is $\pi - |\xi| \geqslant \pi - \delta^*$. It remains to consider the case when the shortest path uses s from bottom to top. This can only happen when p is closer to the bottom end of s than to its top end—in other words, when $\pi < p < 2\pi$. Since $0 \leqslant \theta \leqslant \pi$ and $p = \theta - \xi/2$, this implies either $\theta < \delta^*/2$ and $\xi > 2\theta$, or $\theta > \pi - \delta^*/2$ and $\xi < -2(\pi - \theta)$. Since $q = \theta + \pi + \xi/2$, the first case implies $\pi \leqslant q \leqslant \pi + \delta^* < 2\pi$, while the second case implies $\pi < 2\pi - \delta^* \leqslant q \leqslant 2\pi$. In both cases, q lies closer to the bottom end of the shortcut than to its top end, a contradiction to the shortcut being used from bottom to top to go from p to q.

It follows that for $0 \leqslant \theta \leqslant \pi$, we have $(\theta,\xi) \in \mathfrak{R}(s,\delta^*)$ if and only if $f(\theta,\xi) \leqslant \pi - \delta^*$. To analyze f, we partition the rectangle $[0,\pi] \times [-\delta^*, \delta^*]$ into regions, depending on the signs of $\alpha/2 - p$ and $q - (2\pi - \alpha/2)$. First, we have $p < \alpha/2$ if and only if $\xi > 2\theta - \alpha$. This is the light gray region above the blue line in Figure 3(left). Second, we have $q > 2\pi - \alpha/2$ if and only if $\xi > 2\pi - \alpha - 2\theta$. This is the dark gray region above the green line in Figure 3(left). If the two regions do not intersect then we get three regions as shown in Figure 3(left). Otherwise, if $\alpha > \pi - \delta^*$, or equivalently, $\delta^* > \pi - a - 2\delta$, then the regions intersect and we get four regions as illustrated in Figure 3(right). We now study $\mathfrak{R}(s,\delta^*)$ independently for each of the three or four regions.

In region A, we have $p < \alpha/2$ and $q < 2\pi - \alpha/2$. It follows that

$$\begin{aligned} f(\theta,\xi) &= \alpha/2 - p + \alpha - 2\delta + 2\pi - \alpha/2 - q \\ &= -\theta + \xi/2 + \alpha - 2\delta + 2\pi - \theta - \pi - \xi/2 \\ &= \pi - 2\delta + 2(\alpha/2 - \theta). \end{aligned}$$

This implies that $f(\theta,\xi) \leqslant \pi - \delta^*$ if and only if $\theta \geqslant \alpha/2 + \delta^*/2 - \delta = a/2 + \delta^*/2$. This is the blue area as shown in Figure 4.

In region B, we have $p \geqslant \alpha/2$ and $q > 2\pi - \alpha/2$. This implies

$$\begin{aligned} f(\theta,\xi) &= p - \alpha/2 + \alpha - 2\delta + q - 2\pi + \alpha/2 \\ &= \theta - \xi/2 + \alpha - 2\delta + \theta + \pi + \xi/2 - 2\pi \\ &= 2(\theta - (\pi - \alpha/2)) + \pi - 2\delta, \end{aligned}$$

and so we have $f(\theta,\xi) \leqslant \pi - \delta^*$ if and only if $\theta \leqslant \pi - \alpha/2 - \delta^*/2 + \delta$. This is the green area in Figure 4.

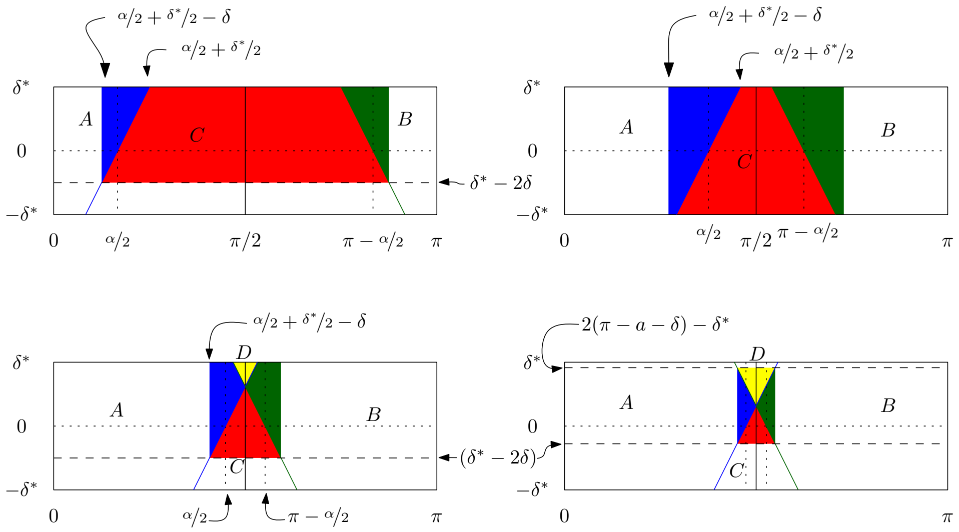


Figure 4 The region $\mathfrak{R}(s, \delta^*)$ in four different situations.

Next, in region C, we have $p \geqslant \alpha/2$ and $q \leqslant 2\pi - \alpha/2$. Therefore,

$$\begin{aligned} f(\theta, \xi) &= p - \alpha/2 + \alpha - 2\delta + 2\pi - \alpha/2 - q \\ &= \theta - \xi/2 - 2\delta + 2\pi - \theta - \pi - \xi/2 \\ &= \pi - 2\delta - \xi. \end{aligned}$$

We have $f(\theta, \xi) \leqslant \pi - \delta^*$ if and only if $\xi \geqslant \delta^* - 2\delta$. This is the red area in Figure 4.

When $\alpha > \pi - \delta^*$ regions A and B intersect in region D, as shown in Figure 3(right). In region D we have $p < \alpha/2$ and $q > 2\pi - \alpha/2$, and therefore

$$\begin{aligned} f(\theta, \xi) &= \alpha/2 - p + \alpha - 2\delta + q - 2\pi + \alpha/2 \\ &= 2\alpha - 2\delta - 2\pi - \theta + \xi/2 + \theta + \pi + \xi/2 \\ &= 2(\alpha - \delta) - \pi + \xi \\ &= 2(a + \delta) - \pi + \xi, \end{aligned}$$

since $\alpha - \delta = a + \delta$. Thus, we have $f(\theta, \xi) \leqslant \pi - \delta^*$ if and only if $\xi \leqslant 2(\pi - a - \delta) - \delta^*$. This is the yellow area in region D in Figure 4. There are two cases that can occur, as is shown on the bottom left and bottom right of Figure 4. The discussion of these cases can be found in the proof of the following lemma, given in the full paper [1].

▶ **Lemma 3.** *Let $\delta^* \in [0, \pi - 2]$, and let s be a shortcut of length $a \in (0, 2]$. Then, the region $\mathfrak{R}(s, \delta^*)$ of s in the cylinder $\mathfrak{S}(\delta^*) = [0, 2\pi] \times [-\delta^*, +\delta^*]$ forms two identical rectangles whose width is exactly $\pi - a - \delta^*$ and whose height is*

$$\begin{cases} 2\delta^* & \text{if } \delta^* \leqslant \delta(a) \\ 2\delta(a) & \text{if } \delta^* > \delta(a) \text{ and } \delta^* \leqslant \pi - a - \delta(a) \\ 2(\pi - a - \delta^*) & \text{otherwise.} \end{cases}$$

Note that if $\delta^* \leqslant \delta(2) = \pi/2 - 1$, then it always holds that $\delta^* \leqslant \pi - a - \delta(a)$ for any $0 \leqslant a \leqslant 2$ since $\pi - a - \delta(a) \geqslant \pi - 2 - \delta(2) = \delta(2) \geqslant \delta^*$. Hence, the last case of Lemma 3 where $\delta^* > \pi - a - \delta(a)$ only happens when $\delta^* > \delta(2) = \pi/2 - 1$.

Table 1 The values a_k^*, δ_k^*, and $\pi - \delta_k^*$.

k	a_k^*	δ_k^*	$\mathrm{diam}(S) = \pi - \delta_k^*$
2	1.4782	0.0926	3.0490
3	1.8435	0.2509	2.8907
4	1.9619	0.3943	2.7473
5	1.9969	0.5164	2.6252
6	2.0000	0.5708	2.5708

We will also be interested in the length of the intersection of $\mathfrak{R}(s, \delta^*)$ with the middle line $\mathfrak{M} = \{\xi = 0\}$ of $\mathfrak{S}(\delta^*)$. Note that $\mathfrak{M}$ has length 2π. We have the following corollary to Lemma 3:

▶ **Corollary 4.** *Let $\delta^* \in [0, \pi - 2]$, and let s be a shortcut of length $a \in (0, 2]$. Then*

$$|\mathfrak{M} \cap \mathfrak{R}(s, \delta^*)| = \begin{cases} 2(\pi - a - \delta^*) & \text{if } \delta(a) \geqslant \delta^*/2 \\ 0 & \text{otherwise.} \end{cases}$$

3 Up to five shortcuts

In this section we derive the exact value of $\mathrm{diam}(k)$ for $k \in \{2, 3, 4, 5\}$, and show the unique optimal configuration of shortcuts in each case. The proof is quite easy, comparing the areas of $\mathfrak{R}(s, \delta^*)$ with the area of $\mathfrak{S}(\delta^*)$, if one assumes that the shortest path between any pair of points uses at most one shortcut. Showing that using a combination of shortcuts does not help takes considerable additional effort.

Using only one shortcut. Again we consider a target diameter of the form $\pi - \delta^*$, with $\delta^* \in [0, \pi - 2]$. By Lemma 3, the region $\mathfrak{R}(s, \delta^*)$ of a shortcut s of length a consists of two rectangles of width $\pi - a - \delta^*$ and height $2\delta(a)$ for $\delta(a) < \delta^*$, and height $2\delta^*$ for $\delta(a) \geqslant \delta^*$. We define a^* such that $\delta(a^*) = \delta^*$, or $a^* = 2$ when $\delta^* > \delta(2)$.

Then the area $A(a, \delta^*)$ of $\mathfrak{R}(s, \delta^*)$ is

$$A(a, \delta^*) = \begin{cases} 4\delta^*(\pi - a - \delta^*) & \text{for} \quad a > a^* \\ 4\delta(a)(\pi - a - \delta^*) & \text{for} \quad a \leqslant a^* \end{cases}$$

▶ **Lemma 5.** *For fixed $\delta^* \leqslant 0.7$, the function $a \mapsto A(a, \delta^*)$ is increasing for $a \leqslant a^*$ and decreasing for $a \geqslant a^*$. Its maximum value is $A(a^*, \delta^*) = 4\delta^*(\pi - a^* - \delta^*)$.*

The proof can be found in the full paper [1].

Let $k \in \{2, 3, 4, 5\}$. Since $a \mapsto a + \delta(a)$ is an increasing function that maps $[0, 2]$ to $[0, \pi/2 + 1]$, there is a unique a_k^* that solves the equation

$$a_k^* + \delta(a_k^*) = \frac{k-1}{k}\pi.$$

We set $\delta_k^* := \delta(a_k^*)$, and will show that this number determines the optimal diameter for k shortcuts. Table 1 shows the numerical values. For completeness, we already include the case $k = 6$ in the table by setting $a_6^* = 2$.

▶ **Lemma 6.** *For $k \in \{2, 3, 4, 5\}$ there is a set S of k shortcuts that achieves* $\mathrm{diam}(S) = \pi - \delta_k^*$. *Assuming that no pair of points uses more than one shortcut, this is optimal and the solution is unique up to rotation.*

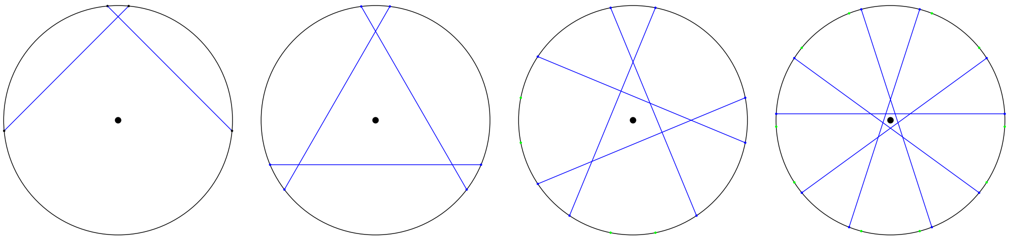


Figure 5 The optimal shortcut configurations for $k = 2, 3, 4, 5$.

Proof. By Lemma 3, the region $\mathfrak{R}(s, \delta_k^*)$ of a shortcut s of length $|s| = a_k^*$ consists of two rectangles of height $2\delta_k^*$ and width $\pi - (a_k^* + \delta_k^*) = \pi/k$. Each rectangle covers the entire height of $\mathfrak{S}(\delta^*)$, and by rotating s about the origin we can translate the rectangles anywhere inside $\mathfrak{S}(\delta^*)$. This implies that we can use k such rectangles to cover the range $0 \leqslant \theta \leqslant \pi$. Then for every $(\theta, \xi) \in \mathfrak{S}(\delta^*)$ there is a shortcut s such that $d_s(\theta - {}^{\xi}\!/2, \theta + \pi + {}^{\xi}\!/2) \leqslant \pi - \delta_k^*$, and $\operatorname{diam}(S) = \pi - \delta_k^*$. Figure 5 shows the resulting configurations.

Assume now that a set $S = \{s_1, \dots, s_k\}$ of k shortcuts is given with $\operatorname{diam}(S) \leqslant \pi - \delta^*$, where $\delta^* \geqslant \delta_k^*$, and that no pair of points uses more than one shortcut. This implies that the regions $\mathfrak{R}(s_i, \delta^*)$ must entirely cover the strip $\mathfrak{S}(\delta^*)$, and in particular

$$\sum_{i=1}^{k} A(|s_i|, \delta^*) \geqslant 4\delta^*\pi.$$

If we choose a^* such that $\delta(a^*) = \delta^*$, then $a^* \geqslant a_k^*$. By Lemma 5 we have

$$A(|s_i|, \delta^*) \leqslant A(a^*, \delta^*) = 4\delta^*(\pi - a^* - \delta^*).$$

From $kA(a^*, \delta^*) \geqslant 4\delta^*\pi$ we have $k(\pi - a^* - \delta^*) \geqslant \pi$, or $a^* + \delta^* \leqslant \frac{k-1}{k}\pi$, which implies $a^* = a_k^*$ and $\delta^* = \delta_k^*$. But then the regions $\mathfrak{R}(s_i, \delta_k^*)$ must be non-overlapping, and the solution is unique up to rotation. ◀

Shortcuts cannot be combined. It remains to show that the configurations in Figure 5 are optimal even if combinations of shortcuts can be used. We start by defining $\mu_k \in [0, 2]$ to be such that $\delta(\mu_k) = \delta_k^*/2$. By Lemma 3, $\mathfrak{R}(s, \delta_k^*)$ intersects the middle line $\mathfrak{M}$ if and only if $|s| \geqslant \mu_k$. In other words, for two antipodal points p and q we can have $d_s(p, q) \leqslant \pi - \delta_k^*$ only if $|s| \geqslant \mu_k$.

We handle the somewhat special case $k = 2$ first.

▶ **Lemma 7.** *If S is a set of two shortcuts that achieves diameter $\operatorname{diam}(S) \leqslant \pi - \delta_2^*$, then S is identical to the configuration of Figure 5 up to rotation.*

Proof. Let $S = \{s_1, s_2\}$ with $|s_1| \leqslant |s_2|$. Let p and q be the midpoints of the inner and outer umbra of s_2. The shortest path between p and q cannot use s_2 at all by Observation 1, so $d_{s_1}(p, q) \leqslant \pi - \delta_2^*$. This implies $|s_1| \geqslant \mu_2 \approx 1.2219$. Since $\delta_2^* \approx 0.0926 < \mu_2/2$, the interval $[q - \delta_2^*, q + \delta_2^*]$ lies in $U(s_2)$, and so we have $d_{s_1}(p, q') \leqslant \pi - \delta_2^*$ for all $q' \in [q - \delta_2^*, q + \delta_2^*]$. This implies $|s_1| \geqslant a_2^*$.

We next observe that $U(s_1) \cap U(s_2) = \emptyset$. Otherwise, Observation 1 applied to an antipodal pair in $U(s_1) \cap U(s_2)$ implies $\operatorname{diam}(S) = \pi$, a contradiction.

The two arcs between the inner and outer umbras of s_1 have length $\pi - |s_1| \leqslant \pi - a_2^*$. The inner umbra $U(s_2)$ has length $|s_2| \geqslant a_2^*$ and lies in one of these arcs. That leaves a gap of

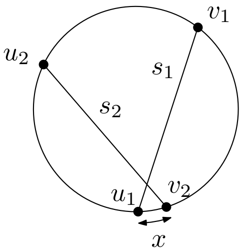


Figure 6 The two shortcuts must intersect.

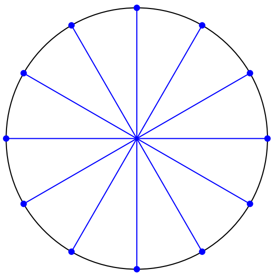


Figure 7 An optimal configuration of six shortcuts.

most $\pi - 2a_2^* = 2\delta_2^*$ between the two inner umbras (by definition of a_2^*, we have $a_2^* + \delta_2^* = \pi/2$). Since $\delta(s_2) \geqslant \delta(s_1) \geqslant \delta_2^*$, this implies that the two shortcuts intersect, see Figure 6.

Let x be the length of overlap of the arcs of s_1 and s_2, that is, $x = |\widehat{u_1 v_2}|$ in Figure 6. Any path that uses both s_1 and s_2 has length at least $|s_1| + |s_2| + x \geqslant 2a_2^* + x = \pi - 2\delta_2^* + x$. This is bounded by $\pi - \delta_2^*$ only if $x \leqslant \delta_2^*$. But then the arc $\widehat{v_1 u_2}$ has length at most

$$2\pi - \alpha(s_1) - \alpha(s_2) + x \leqslant 2\pi - 2(a_2^* + 2\delta_2^*) + \delta_2^* = \pi + (\pi - 2a_2^*) - 3\delta_2^* = \pi - \delta_2^*,$$

and there is no reason to use the two shortcuts at all. It follows that there is no pair of points that uses more than one shortcut, and Lemma 6 implies the claim. ◀

For $3 \leqslant k \leqslant 6$, the key insight is the following lemma, proven in the full paper [1].

▶ **Lemma 8.** *Let S be a set of k shortcuts for $k \in \{3, 4, 5, 6\}$ such that* $\operatorname{diam}(S) \leqslant \pi - \delta_k^*$. *Then there is no antipodal pair of points $p, q \in C$ such that the path of length $d_S(p, q)$ uses more than one shortcut.*

▶ **Lemma 9.** *Let S be a set of k shortcuts for $k \in \{3, 4, 5\}$ such that* $\operatorname{diam}(S) \leqslant \pi - \delta_k^*$. *Then there is no pair of points $p, q \in C$ such that the path of length $d_S(p, q)$ uses more than one shortcut.*

Proof. By Lemma 8 pairs of antipodal points cannot use more than one shortcut. This implies that the middle line $\mathfrak{M}$ of $\mathfrak{S}(\delta_k^*)$ is covered by the regions $\mathfrak{R}(s_i, \delta_k^*)$. The region $\mathfrak{R}(s_i, \delta_k^*)$ intersects $\mathfrak{M}$ only if $|s_i| \geqslant \mu_k$, so by Corollary 4 $\mathfrak{R}(s_i, \delta_k^*)$ covers at most $2(\pi - \mu_k - \delta_k^*)$ of $\mathfrak{M}$. Calculation shows that $(k-1)(\pi - \mu_k - \delta_k^*) < \pi$, so all k shortcuts have length at least μ_k. Since $2\mu_k > \pi$, this implies that no shortcuts can be combined. ◀

Combining Lemmas 6, 7, and 9, we obtain our first theorem.

▶ **Theorem 10.** *For $k \in \{2, 3, 4, 5\}$ there is a set S of k shortcuts that achieves* $\operatorname{diam}(S) = \pi - \delta_k^*$. *This is optimal and the solution is unique up to rotation.*

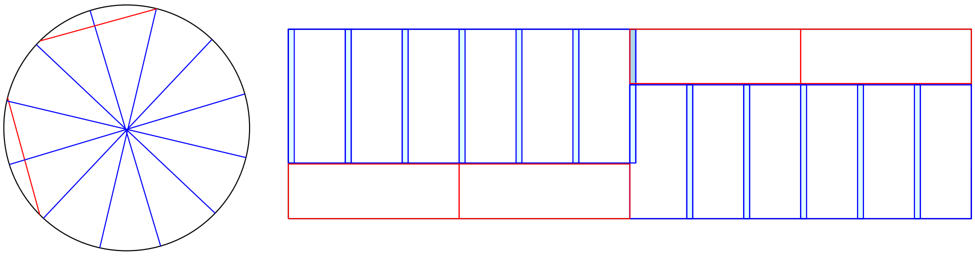


Figure 8 A shortcut configuration S of 8 shortcuts with $\operatorname{diam}(S) < \operatorname{diam}(6)$, and the corresponding regions in the strip $\mathfrak{S}(\delta^*)$.

4 Six and seven shortcuts

The configuration of six shortcuts of length 2 (that is, all shortcuts are diameters of the circle) shown in Figure 7 achieves diameter $\pi - \delta(2) = {}^{\pi}/_{2} + 1$. Unlike the cases $2 \leqslant k \leqslant 5$, this configuration is not unique—it can be perturbed quite a bit without changing the diameter.

It remains to argue that the configuration is indeed optimal, that is, there is no set S of six shortcuts that achieves $\operatorname{diam}(S) < \pi - \delta(2)$. Here, we cannot use a simple area argument as in the case $k < 6$, as the regions of the optimal solution in $\mathfrak{S}(\delta_6^*)$ overlap heavily.

In fact, we can show that even if we allow *seven* shortcuts, there is no set S of shortcuts that achieves $\operatorname{diam}(S) < \pi - \delta(2)$. This implies a collapse between the cases of $k = 6$ and $k = 7$, that is, $\operatorname{diam}(7) = \operatorname{diam}(6) = \pi - \delta(2)$. The proof is quite long and rather technical, and can be found in the full paper [1].

▶ **Theorem 11.** *There is a set S of six shortcuts that achieves* $\operatorname{diam}(S) = \pi - \delta(2) = {}^{\pi}/_{2} + 1$. *There is no configuration of six or seven shortcuts that has diameter smaller than* ${}^{\pi}/_{2} + 1$. *Therefore, we have* $\operatorname{diam}(7) = \operatorname{diam}(6) = {}^{\pi}/_{2} + 1$.

5 Eight shortcuts

With eight shortcuts we can improve on the diameter, obtaining $\operatorname{diam}(8) < \operatorname{diam}(7) = \operatorname{diam}(6)$. Our construction S consists of six long shortcuts with length $a_1 \approx 1.999870869$ and two short ones with length $a_2 \approx 0.988571799$, placed as in Figure 8(left), and achieves the diameter $\operatorname{diam}(S) \approx \pi - 0.5822245291 = 2.559368125 < \operatorname{diam}(6)$.

We obtained this construction by maximizing δ^* with constraints $\pi - a_1 - \delta^* \geqslant \pi/6$, $\pi - a_2 - \delta^* \geqslant \pi/2$, and $\delta(a_1) + \delta(a_2) \geqslant \delta^*$. We can thus cover $\mathfrak{S}(\delta^*)$ as seen in the diagram in Figure 8(right). In particular, we have $\pi - a_2 - \delta^* = \pi/2$ and $\delta(a_1) + \delta(a_2) = \delta^*$, while we have a strict inequality $\pi - a_1 - \delta^* > \pi/6$ in our construction. So, in the strip $\mathfrak{S}(\delta^*)$, the regions slightly overlap.

6 An asymptotically tight bound

In this final section, we show that $\operatorname{diam}(k) = 2 + \Theta(1/k^{\frac{2}{3}})$ as k goes to infinity.

▶ **Theorem 12.** *To achieve diameter at most $2 + 1/m$, $\Theta(m^{\frac{3}{2}})$ shortcuts are both necessary and sufficient.*

Proof. We prove the necessary condition first. Consider two points p, q that form an angle of $\pi - t/m$, for some integer $0 \leqslant t \leqslant \sqrt{m} - 2$. Consider two intervals I_p and I_q, both of arc length $4/m$, and centered at p and q, respectively. We claim that if there is no shortcut

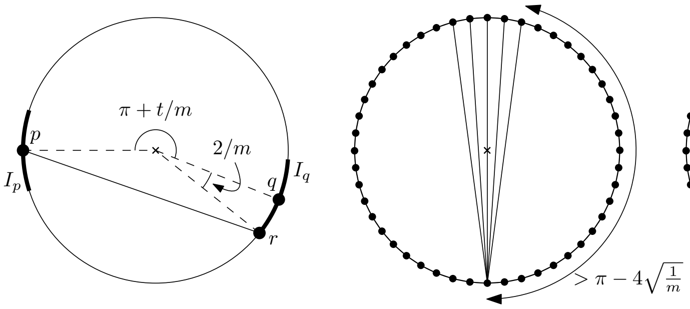


Figure 9 (left) If there is no shortcut between I_p and I_q then the shortest path from p to q must visit a point r on the circle not in either interval. (center) Shortcut between every pair that makes an angle larger than $\pi - 4\sqrt{1/m}$. (right) Adding shortcuts of arc length $\pi - t/m$.

connecting a point of I_p with a point of I_q, then the distance between p and q is larger than $2 + 1/m$.

If there is no such shortcut, then the shortest path from p to q must visit a point r on the circle not in either interval, see Figure 9(left). The sum $|pr| + |rq|$ is minimized when r is the point making angle $2/m$ with q, so we have $\alpha(pr) = \pi - (t+2)/m$ and $\alpha(rq) = 2/m$. This gives us

$$|qr| = 2\sin\frac{2}{2m} = 2\sin\frac{1}{m} \geqslant \frac{2}{m} - \frac{2}{3!}\frac{1}{m^3} > \frac{2}{m} - \frac{1}{3m} = \frac{5}{3m},$$
$$|pr| = 2\sin(\frac{\pi}{2} - \frac{t+2}{2m}) = 2\cos\frac{t+2}{2m} \geqslant 2\cos\frac{\sqrt{m}}{2m} = 2\cos\frac{1}{2\sqrt{m}} \geqslant 2 - \frac{1}{4m},$$

and so $|pr| + |rq| > 2 + 1/m$.

We now subdivide C into $\Theta(m)$ intervals of length at least $6/m$. Consider a pair of intervals I, J at arc distance at least $\pi - 1/\sqrt{m}$. Then there are points $p \in I$ and $q \in J$ with $I_p \subset I$ and $I_q \subset J$ and p, q forming an angle of the form $\pi - t/m$ for an integer $0 \leqslant t \leqslant \sqrt{m} - 2$. It follows that there must be some shortcut connecting I and J. Since there are $\Theta(m^{3/2})$ such pairs of intervals, we must have at least $\Omega(m^{3/2})$ shortcuts.

We now turn to the sufficient condition, and construct a set of $\Theta(m^{3/2})$ shortcuts that give a diameter of $2 + 1/m$.

We start by placing $4\pi m$ points uniformly around the circle, and connect each pair that makes an angle larger than $\pi - 4\sqrt{1/m}$, as shown in Figure 9(center). This creates $\Theta(m^{3/2})$ shortcuts and ensures that for points p, q with angle larger than $\pi - 4\sqrt{1/m}$ the distance between p and q is bounded by $2 + 1/m$.

It remains to add shortcuts to decrease the distance of point pairs p, q that form an arc between 2 and $\pi - 4\sqrt{1/m}$. For each integer t with $4\sqrt{m} < t < 2m$ we will create a set of shortcuts of arc length $\pi - t/m$, see Figure 9(right). These shortcuts will be used for pairs p, q forming an arc between $\pi - t/m$ and $\pi - (t-1)/m$.

Let us fix such a t, and consider a shortcut s of arc length $\pi - t/m$. Then the length of the shortcut is

$$|s| = 2\sin\frac{\pi - t/m}{2} = 2\cos\frac{t}{2m}.$$

Using the bound $\cos x \leqslant 1 - \frac{x^2}{2} + \frac{x^4}{24} \leqslant 1 - (\frac{1}{2} - \frac{1}{24})x^2 = 1 - \frac{11}{24}x^2$ for $x < 1$, we have

$$|s| \leqslant 2 - 2\frac{11}{24}\frac{t^2}{4m^2} = 2 - \frac{11}{48}\frac{t^2}{m^2} < 2 - \frac{1}{6}\frac{t^2}{m^2} = 2 - 2\Delta,$$

where we define $\Delta = \frac{1}{12}(\frac{t}{m})^2$. Since $t > 4\sqrt{m}$ we have $\Delta > \frac{16}{12}\frac{1}{m} > \frac{1}{m}$.

We repeat shortcuts of this length every arc interval of length Δ. Consider now a pair of points p, q forming an angle in the interval $\pi - t/m$ to $\pi - (t-1)/m$. We can go from p to q by first going to the nearest shortcut along an arc of length at most Δ, then following the shortcut of length at most $2 - 2\Delta$, and finally going backwards by at most Δ, or forward by at most $1/m < \Delta$. It follows that the distance between p and q is at most $2 - 2\Delta + 2\Delta = 2$.

The number of shortcuts of length $\pi - t/m$ is $2\pi/\Delta$, and so the total number of shortcuts of this type is

$$\sum_{t=4\sqrt{m}+1}^{2m} 24\pi\frac{m^2}{t^2} = 24\pi m^2 \sum_{t=4\sqrt{m}+1}^{2m} \frac{1}{t^2} \leqslant 24\pi m^2 \int_{4\sqrt{m}}^{\infty} \frac{1}{x^2}\, dx = 6\pi m^{3/2}.$$

This completes the proof. ◀

7 Conclusions

We have given exact bounds on the diameter for up to seven shortcuts. In all cases, the shortcuts are of equal length. For $k = 8$, however, our upper bound construction uses shortcuts of two different lengths. On the other hand, it is not difficult to see that eight shortcuts of equal length cannot even achieve a slightly better diameter than diam(6). In general, what is the diameter achievable with k shortcuts of equal length?

We have shown that for $k = 0$ and $k = 6$ we have $\text{diam}(k) = \text{diam}(k+1)$. Are there any other values of k for which this holds?

References

1 S. W. Bae, M. de Berg, O. Cheong, J. Gudmundsson, and C. Levcopoulos. Shortcuts for the Circle, 2016. arXiv:1612.02412.

2 D. Bilò, L. Gualà, and G. Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theoretical Computer Science*, 417:12–22, 2012.

3 J. Cáceres, D. Garijo, A. González, A. Márquez, M. L. Puertas, and P. Ribeiro. Shortcut sets for plane Euclidean networks (extended abstract). *Electronic Notes in Discrete Mathematics*, 54:163 – 168, 2016. doi:10.1016/j.endm.2016.09.029.

4 J.-L. De Carufel, C. Grimm, A. Maheshwari, and M. Smid. Minimizing the Continuous Diameter when Augmenting Paths and Cycles with Shortcuts. In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:14, 2016. doi:10.4230/LIPIcs.SWAT.2016.1.

5 F. R. K. Chung. Diameters of graphs: old problems and new results. *Congressus Numerantium*, 60:295–317, 1987.

6 F. R. K. Chung and M. R. Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8:511–534, 1984.

7 P. Erdös and A. Rényi. On a problem in the theory of graphs (in Hungarian). *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 7:623–641, 1962.

8 P. Erdös, A. Rényi, and V. T. Sós. On a problem of graph theory. *Studia Scientarium Mathematicarum Hungar*, 1:215–235, 1966.

9 F. Frati, S. Gaspers, J. Gudmundsson, and L. Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72(4):995–1010, 2015.

10 U. Große, J. Gudmundsson, C. Knauer, M. Smid, and F. Stehn. Fast algorithms for diameter-optimally augmenting paths. In *42nd International Colloquium Automata, Languages, and Programming (ICALP)*, pages 678–688. Springer, 2015.

11 S. Kapoor and M. Sarwat. Bounded-diameter minimum-cost graph problems. *Theory of Computing Systems*, 41(4):779—794, 2007.

12 C. L. Li, S. T. McCormick, and D. Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Operation Research Letters*, 11(5):303—-308, 1992.

13 A. A. Schoone, H. L. Bodlaender, and J. van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11:409–427, 1987.

14 H. Wang. An improved algorithm for diameter-optimally augmenting paths in a metric space, 2016. `arXiv:1608.04456`.

15 B. Yang. Euclidean chains and their shortcuts. *Theoretical Computer Science*, 497:55 – 67, 2013. `doi:10.1016/j.tcs.2012.03.021`.

Routing in Polygonal Domains*

Bahareh Banyassady[†1], Man-Kwun Chiu[‡2], Matias Korman[§3], Wolfgang Mulzer[4], André van Renssen[¶5], Marcel Roeloffzen[‖6], Paul Seiferth[7], Yannik Stein[††8], Birgit Vogtenhuber[9], and Max Willert[10]**

1 **Institut für Informatik, Freie Universität Berlin, Germany**
bahareh@inf.fu-berlin.de

2 **National Institute of Informatics (NII), Tokyo, and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
chiumk@nii.ac.jp

3 **Tohoku University, Sendai, Japan**
mati@dais.is.tohoku.ac.jp

4 **Institut für Informatik, Freie Universität Berlin, Germany**
mulzer@inf.fu-berlin.de

5 **National Institute of Informatics (NII), Tokyo, and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
andre@nii.ac.jp

6 **National Institute of Informatics (NII), Tokyo, and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
marcel@nii.ac.jp

7 **Institut für Informatik, Freie Universität Berlin, Germany**
pseiferth@inf.fu-berlin.de

8 **Institut für Informatik, Freie Universität Berlin, Germany**
yannikstein@inf.fu-berlin.de

9 **Institute of Software Technology, Graz University of Technology, Graz, Austria**
bvogt@ist.tugraz.at

10 **Institut für Informatik, Freie Universität Berlin, Germany**
willerma@inf.fu-berlin.de

Abstract

We consider the problem of routing a data packet through the visibility graph of a polygonal domain P with n vertices and h holes. We may preprocess P to obtain a *label* and a *routing table* for each vertex. Then, we must be able to route a data packet between any two vertices p and q of P, where each step must use only the label of the target node q and the routing table of the current node.

For any fixed $\varepsilon > 0$, we present a routing scheme that always achieves a routing path that exceeds the shortest path by a factor of at most $1 + \varepsilon$. The labels have $\mathcal{O}(\log n)$ bits, and the routing tables are of size $\mathcal{O}((\varepsilon^{-1} + h) \log n)$. The preprocessing time is $\mathcal{O}(n^2 \log n + hn^2 + \varepsilon^{-1}hn)$. It can be improved to $\mathcal{O}(n^2 + \varepsilon^{-1}n)$ for simple polygons.

[*] A full version of the paper is available at https://arxiv.org/abs/1703.09533.

[†] BB was supported in part by DFG project MU/3501-2.
[‡] MC was supported by JST ERATO Grant Number JPMJER1201, Japan.
[§] MK was supported in part by KAKENHI Nos. 15H02665 and 17K12635, Japan.
[¶] AvR was supported by JST ERATO Grant Number JPMJER1201, Japan.
[‖] MR was supported by JST ERATO Grant Number JPMJER1201, Japan.
[**] PS was supported in part by DFG project MU/3501-1.
[††] YS was supported by the DFG within the research training group ‘Methods for Discrete Structures’ (GRK 1408) and by GIF grant 1161.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 10; pp. 10:1–10:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems–Geometrical problems and computations

Keywords and phrases polygonal domains, routing scheme, small stretch, Yao graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.10

1 Introduction

Routing is a crucial problem in distributed graph algorithms [11, 22]. We would like to preprocess a given graph G in order to support the following task: given a data packet that lies at some *source* node p of G, route the packet to a given *target* node q in G that is identified by its *label*. We expect three properties from our routing scheme: first, it should be *local*, i.e., in order to determine the next step for the packet, it should use only information stored with the current node of G or with the packet itself. Second, the routing scheme should be *efficient*, meaning that the packet should not travel much more than the shortest path distance between p and q. The ratio between the length of the routing path and the shortest path in the graph is also called *stretch*. Third, it should be *compact*: the total space requirement should be as small as possible.

There is an obvious solution: for each node v of G, we store at v the complete shortest path tree for v. Thus, given the label of a target node w, we can send the packet for one more step along the shortest path from v to w. Then, the routing scheme will have perfect efficiency, sending each packet along a shortest path. However, this method requires that each node stores the entire topology of G, making it not compact. Thus, the challenge lies in finding the right balance between the conflicting goals of compactness and efficiency.

Thorup and Zwick introduced the notion of a *distance oracle* [30]. Given a graph G, the goal is to construct a compact data structure to quickly answer *distance queries* for any two nodes in G. A routing scheme can be seen as a distributed implementation of a distance oracle [24].

The problem of constructing a compact routing scheme for a general graph has been studied for a long time [1, 3, 7–9, 23, 24]. One of the most recent results, by Roditty and Tov, dates from 2016 [24]. They developed a routing scheme for a general graph G with n vertices and m edges. Their scheme needs to store a poly-logarithmic number of bits with the packet, and it routes a message from s to t on a path with length $\mathcal{O}(k\Delta + m^{1/k})$, where Δ is the shortest path distance between s and t and $k > 2$ is any fixed integer. The routing tables use $mn^{\mathcal{O}(1/\sqrt{\log n})}$ total space. In general graphs, any efficient routing scheme needs to store $\Omega(n^c)$ bits per node, for some constant $c > 0$ [22]. Thus, it is natural to ask whether there are better algorithms for specialized graph classes. For instance, trees admit routing schemes that always follow the shortest path and that store $\mathcal{O}(\log n)$ bits at each node [10, 25, 29]. Moreover, in planar graphs, for any fixed $\varepsilon > 0$, there is a routing scheme with a poly-logarithmic number of bits in each routing table that always finds a path that is within a factor of $1 + \varepsilon$ from optimal [28].

Another approach is called *geometric routing*. Here, the graph is embedded in a geometric space and the routing algorithm has to determine the next vertex for the data packet based on the knowledge of the source and target vertex, the current vertex, and its neighbourhood, see for instance [5, 6] and references therein. A recent result by Bose et al. [6] is very close to our setting. They show that under certain conditions, no geometric routing scheme can achieve stretch $o(\sqrt{n})$.

Here, we consider the class of visibility graphs of a polygonal domain P with h holes and n vertices. Two vertices p and q in P are connected by an edge if and only if they can *see* each other, i.e., if and only if the line segment between p and q is contained in the (closed) region P. The problem of computing a shortest path between two vertices in a polygonal domain has been well-studied in computational geometry [2,4,12,13,16,17,19–21,26,27,31]. Nevertheless, to the best of our knowledge, prior to our work there have been no routing schemes for visibility graphs of polygonal domains that fall into our model. For any $\varepsilon > 0$, our routing scheme needs $\mathcal{O}((\varepsilon^{-1} + h)\log n)$ bits in each routing table, and for any two vertices s and t, it produces a routing path that is within a factor of $1 + \varepsilon$ of optimal. This constitutes a dramatic improvement over traditional geometric routing. Thus, we believe that it makes sense to look for compact routing schemes for geometrically defined graphs.

2 Preliminaries

Let $G = (V, E)$ be an *undirected*, *connected* and *simple* graph. In our model, G is embedded in the Euclidean plane: a *node* $p = (p_x, p_y) \in V$ corresponds to a point in the plane, and an edge $\{p, q\} \in E$ is represented by the line segment $\overline{pq}$. The *length* $|\overline{pq}|$ of an edge $\{p, q\}$ is given by the Euclidean distance between the points p and q. The length of a shortest path between two nodes $p, q \in V$ is denoted by $d(p, q)$.

Now, we formally define a *routing scheme* for G. Each node p of G is assigned a *label* $\ell(p) \in \{0, 1\}^*$ that identifies it in the network. Furthermore, we store with p a *routing table* $\rho(p) \in \{0, 1\}^*$. The routing scheme works as follows: the packet contains the label $\ell(q)$ of the target node q, and initially it is situated at the start node p. In each step of the routing algorithm, the packet resides at a current node $p' \in V$. It may consult the routing table $\rho(p')$ of p' and the label $\ell(q)$ of the target to determine the next node q' to which the packet is forwarded. The node q' must be a neighbor of p' in G. This is repeated until the packet reaches its destination q. The scheme is modeled by a *routing function* $f : \rho(V) \times \ell(V) \to V$.

In the literature, there are varying definitions for the notion of a routing scheme [15,24,32]. For example, we may sometimes store additional information in the *header* of a data packet (it travels with the packet and can store information from past vertices). Similarly, the routing function sometimes allows the use of an *intermediate* target label. This is helpful for recursive routing schemes. Here, however, we will not need any of these additional capabilities.

As mentioned, the routing scheme operates by repeatedly applying the routing function. More precisely, given a start node $p \in V$ and a target label $\ell(q)$, the scheme produces the sequence of nodes $p_0 = p$ and $p_i = f(\rho(p_{i-1}), \ell(q))$, for $i \geq 1$. Naturally, we want routing schemes for which every packet reaches its desired destination. More precisely, a routing scheme is *correct* if for any $p, q \in V$, there exists a finite $k = k(p, q) \geq 0$ such that $p_k = q$ (and $p_i \neq q$ for $0 \leq i < k$). We call $p_0, p_1, \ldots, p_k$ the *routing path* between p and q. The *routing distance* between p and q is defined as $d_\rho(p, q) = \sum_{i=1}^{k} |\overline{p_{i-1}p_i}|$.

The quality of the routing scheme is measured by several parameters: (i) the *label size* $L(n) = \max_{|V|=n} \max_{p \in V} |\ell(p)|$, (ii) the *table size* $T(n) = \max_{|V|=n} \max_{p \in V} |\rho(p)|$, (iii) the *stretch* $\zeta(n) = \max_{|V|=n} \max_{p \neq q \in V} d_\rho(p, q)/d(p, q)$, and (iv) the preprocessing time.

Let P be a polygonal domain with n vertices. The *boundary* ∂P of P consists of h pairwise disjoint simple closed polygonal chains: one *outer boundary* and $h - 1$ *hole boundaries*, or h *hole boundaries* with no outer boundary. All hole boundaries lie inside the outer boundary, and no hole boundary lies inside another hole boundary. In both cases, we say that P has h holes. The interior induced by a hole boundary and the exterior of the outer boundary

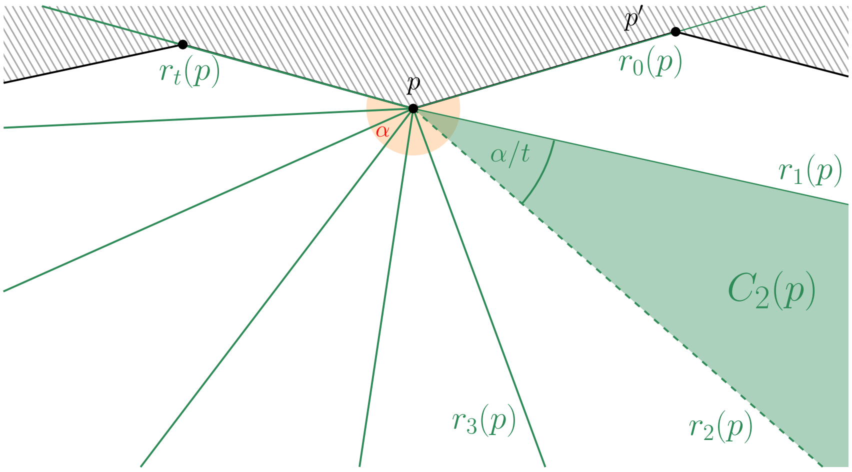


Figure 1 The cones and rays of a vertex p with apex angle α.

are not contained in P. We denote the (open) *interior* of P by $\operatorname{int} P$, i.e., $\operatorname{int} P = P \setminus \partial P$. We make no general position assumption on P. Let n_i, $0 \leq i \leq h-1$, be the number of vertices on the i-th boundary of P. For each boundary i, we number the vertices from 0 to $n_i - 1$, in clockwise order, if i is a hole boundary, or in counterclockwise order if i is the outer boundary. The kth vertex of the ith boundary is denoted by $p_{i,k}$.

Two points p and q in P can *see each other* in P if and only if $\overline{pq} \subset P$. In particular, note that the line segment $\overline{pq}$ may touch ∂P. The *visibility graph* of P, $\mathrm{VG}(P)$, has the same vertices as P and an edge between two vertices if and only if they see each other in P. We show the following main theorem:

▶ **Theorem 2.1.** *Let $\varepsilon > 0$, and let P be a polygonal domain with n vertices and h holes. There is a routing scheme for $\mathrm{VG}(P)$ with stretch $\zeta(n) = 1+\varepsilon$, label size $L(n) = \mathcal{O}(\log n)$ and routing table size $T(n) = \mathcal{O}((\varepsilon^{-1} + h)\log n)$. The preprocessing time is $\mathcal{O}(n^2 \log n + hn^2 + \varepsilon^{-1}hn)$. If P is a simple polygon, the preprocessing time can be improved to $\mathcal{O}(n^2 + \varepsilon^{-1}n)$.*

3 Cones in Polygonal Domains

Let P be a polygonal domain with n vertices and h holes. Furthermore, let $t > 2$ be a parameter, to be determined later. Following Yao [33], we subdivide the visibility polygon of each vertex in P into t *cones* with a small enough apex angle. This will allow us to achieve small stretch and compact routing tables.

Let p be a vertex in P and p' the clockwise neighbor of p if p is on the outer boundary, or the counterclockwise neighbor of p if p lies on a hole boundary. We denote with $\mathbf{r}$ the *ray* from p through p'. To obtain our cones, we rotate $\mathbf{r}$ by certain angles. Let α be the inner angle at p. For $j = 0, \ldots, t$, we write $r_j(p)$ for the ray $\mathbf{r}$ rotated clockwise by angle $j \cdot \alpha/t$.

Now, for $j = 1, \ldots, t$, the cone $C_j(p)$ has apex p, boundary $r_{j-1}(p) \cup r_j(p)$, and opening angle α/t; see Figure 1. For technical reasons, we define $r_j(p)$ not to be part of $C_j(p)$, for $0 \leq j < t$, whereas we consider $r_t(p)$ to be part of $C_t(p)$. Furthermore, we write $\mathcal{C}(p) = \{C_j(p) \mid 1 \leq j \leq t\}$ for the set of all cones with apex p. Since the opening angle of each cone is $\alpha/t \leq 2\pi/t$ and since $t > 2$, each cone is convex.

▶ **Lemma 3.1.** *Let p be a vertex of P and let $\{p, q\}$ be an edge of $\mathrm{VG}(P)$ that lies in the cone $C_j(p)$. Furthermore, let s be a vertex of P that lies in $C_j(p)$, is visible from p, and that*

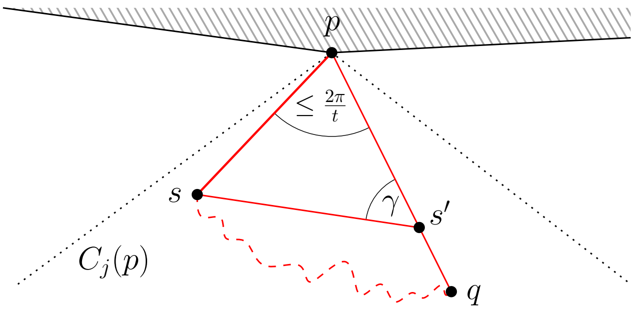


Figure 2 Illustration of Lemma 3.1. The points s and s' have the same distance to p. The dashed line represents the shortest path from s to q.

is closest to p. Then, $d(s,q) \leq |\overline{pq}| - (1 - 2\sin(\pi/t))\,|\overline{ps}|$.

Proof. Let s' be the point on the line segment $\overline{pq}$ with $|\overline{ps'}| = |\overline{ps}|$; see Figure 2. Since p can see q, we have that p can see s' and s' can see q. Furthermore, s can see s', because p can see s and s' and we chose s to be closest to p, so the triangle $\Delta(p,s,s')$ cannot contain any vertices or (parts of) edges of P in its interior. Now, the triangle inequality yields $d(s,q) \leq |\overline{ss'}| + |\overline{s'q}|$. Let β be the inner angle at p between the line segments $\overline{ps}$ and $\overline{ps'}$. Since both segments lie in the cone $C_j(p)$, we get $\beta \leq 2\pi/t$. Thus, the angle between $\overline{s'p}$ and $\overline{s's}$ is $\gamma = \pi/2 - \beta/2$. Using the sine law and $\sin 2x = 2\sin x \cos x$, we get

$$|\overline{ss'}| = |\overline{ps}| \cdot \frac{\sin\beta}{\sin\gamma} = |\overline{ps}| \cdot \frac{\sin\beta}{\sin\left((\pi/2) - (\beta/2)\right)} = |\overline{ps}| \cdot \frac{2\sin(\beta/2)\cos(\beta/2)}{\cos(\beta/2)} \leq 2|\overline{ps}|\sin(\pi/t).$$

Furthermore, we have $|\overline{s'q}| = |\overline{pq}| - |\overline{ps'}| = |\overline{pq}| - |\overline{ps}|$. Thus, the triangle inequality gives

$$d(s,q) \leq 2|\overline{ps}|\sin(\pi/t) + |\overline{pq}| - |\overline{ps}| = |\overline{pq}| - (1 - 2\sin(\pi/t))\,|\overline{ps}|. \qquad \blacktriangleleft$$

4 The Routing Scheme

Let $\varepsilon > 0$, and let P be a polygonal domain with n vertices and h holes. We describe a routing scheme for VG(P) with stretch factor $1+\varepsilon$. The idea is to compute for each vertex p the corresponding set of cones $\mathcal{C}(p)$ and to store a certain interval of indices for each cone $C_j(p)$ in the routing table of p. If an interval of a cone $C_j(p)$ contains the target vertex t, we proceed to the nearest neighbor of p in $C_j(p)$; see Figure 3. We will see that this results in a routing path with small stretch.

In the preprocessing phase, we first compute the label of each vertex $p_{i,k}$. The label of $p_{i,k}$ is the binary representation of i, concatenated with the binary representation of k, that is, $\ell(p_{i,k}) = (i,k)$. Thus, all labels are distinct binary strings of length $\lceil \log h \rceil + \lceil \log n \rceil$.

Let p be a vertex in P. Throughout this section, we will write $\mathcal{C}$ and C_j instead of $\mathcal{C}(p)$ and $C_j(p)$. The routing table of p is constructed as follows: first, we compute a shortest path tree T for p. For a vertex s of P, let T_s be the subtree of T with root s, and denote the set of all vertices on the i-th hole in T_s by $I_s(i)$. The following well-known observation lies at the heart of our routing scheme. For space reasons, we omit the proof from this extended abstract.

▶ **Observation 4.1.** *Let q_1 and q_2 be two vertices of P. Let π_1 be the shortest path in T from p to q_1, and π_2 the shortest path in T from p to q_2. Let l be the lowest common ancestor of q_1 and q_2 in T. Then, π_1 and π_2 do not cross or touch in a point x with $d(p,x) > d(p,l)$.*

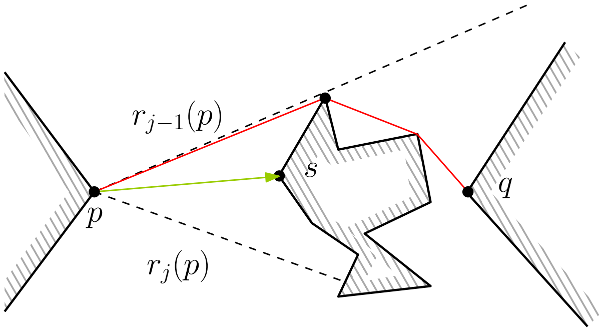


Figure 3 The idea of the routing scheme. The first edge on a shortest path from p to q (red) is contained in $C_j(p)$. The routing algorithm will route the packet from p to s (green), the closest vertex to p in C_j.

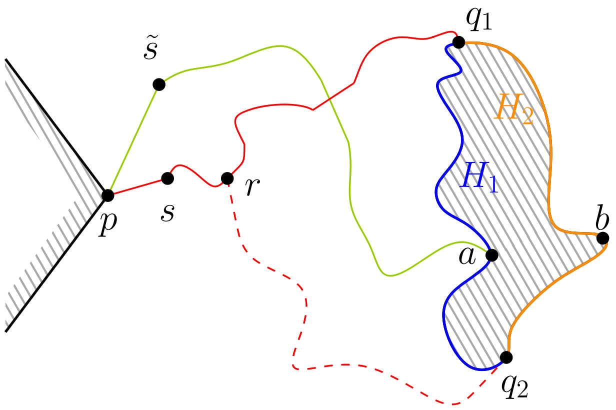


Figure 4 The shortest path from p to a (green) crosses the shortest path from p to q_1 (red). This gives a contradiction by Observation 4.1.

▶ **Lemma 4.2.** *Let $e = (p, s)$ be an edge in T. Then, the indices of the vertices in $I_s(i)$ form an interval. Furthermore, let $f = (p, s')$ be another edge in T, such that e and f are consecutive in the cyclic order around p in T. Then, the indices of the vertices in $I_s(i) \cup I_{s'}(i)$ are again an interval.*

Proof. For the first part of the lemma, suppose that the indices for $I_s(i)$ do not form an interval. Then, there are two vertices $q_1, q_2 \in I_s(i)$ such that if we consider the two polygonal chains H_1 and H_2 with endpoints q_1 and q_2 that constitute the boundary of hole i, there are two vertices $a, b \notin I_s(i)$ with $a \in H_1$ and $b \in H_2$ (see Figure 4). Let π_1 and π_2 be the shortest paths in T from s to q_1 and from s to q_2. Let r be the last common vertex of π_1 and π_2, and suppose without loss of generality that H_1, the subpath of π_1 from r to q_1, and the subpath of π_2 from r to q_2 bound a region inside P. Then, there has to be a child $\tilde{s}$ of p in T such that $a \in I_{\tilde{s}}(i)$ and such that the shortest path from $\tilde{s}$ to a intersects $\pi_1 \cup \pi_2$. Since p is the lowest common ancestor of a and q_1 and of a and q_2, this contradicts Observation 4.1.

The proof for the second part of the lemma is almost identical. We assume for the sake of contradiction that the indices in $I_s(i) \cup I_{s'}(i)$ do not form an interval, and we find vertices $q_1, q_2 \in I_s(i) \cup I_{s'}(i)$ such that if we split the boundary of hole i into two chains H_1 and H_2 between q_1 and q_2, there are two vertices $a, b \notin I_s(i) \cup I_{s'}(i)$ with $a \in H_1$ and $b \in H_2$. Again, let π_1 be the shortest path in T from s to q_1 and π_2 the shortest path in T from s to q_2, and consider the least common ancestor r of q_1 and q_2 in T. Without loss of generality, we assume that the region R bounded by H_1, the subpath of π_1 from r to q_1, and the subpath of π_2 from r to q_2 lies inside P. Now, the lowest common ancestor r may be p, but since s and s' are consecutive in the cyclic order around p, the other children of p are either all inside or all outside R. In either case, we can derive a contradiction to Observation 4.1 by noting that either the shortest path from s to a or the shortest path from s to b has to cross $\pi_1 \cup \pi_2$. ◀

Lemma 4.2 indicates how to construct the routing table $\rho(p)$ for p. We set

$$t = \pi / \arcsin\left(\frac{1}{2\left(1+\varepsilon^{-1}\right)}\right), \tag{1}$$

and we construct a set $\mathcal{C}$ of cones for p as in Section 3. Let $C_j \in \mathcal{C}$ be a cone, and let Π_i be a hole boundary or the outer boundary. We define $C_j \sqcap \Pi_i$ as the set of all vertices q on Π_i for which the first edge of the shortest shortest path from p to q lies in C_j. By Lemma 4.2, the indices of the vertices in $C_j \sqcap \Pi_i$ form a (possibly empty) cyclic interval $[k_1, k_2]$. If $C_j \sqcap \Pi_i = \emptyset$, we do nothing. Otherwise, if $C_j \sqcap \Pi_i \neq \emptyset$, there is a vertex $r \in C_j$ closest to p, and we add the entry (i, k_1, k_2, r) to $\rho(p)$. This entry needs $\lceil \log h \rceil + 3 \cdot \lceil \log n \rceil$ bits.

Now, the routing function $f : \rho(V) \times \ell(V) \to V$ is quite simple. Given a current vertex p and a target label $\ell(t) = (i, k)$, we search the routing table $\rho(p)$ for an entry (i, k_1, k_2, r) with $k \in [k_1, k_2]$. By construction, this entry is unique. We then forward the packet from p to the neighbor r (see Figure 3).

5 Analysis

We analyze the stretch factor of our routing scheme and give upper bounds on the size of the routing tables and the preprocessing time. Let $\varepsilon > 0$ be fixed, and let $1 + \varepsilon$ be the desired stretch factor. We set t as in (1). First, we bound t in terms of ε. This immediately gives that $|\mathcal{C}(p)| \in \mathcal{O}(\varepsilon^{-1})$, for every vertex p.

▶ **Lemma 5.1.** *We have* $t \leq 2\pi\left(1+\varepsilon^{-1}\right)$.

Proof. For $x \in (0, 1/2]$, we have $\sin x \leq x$, so for $z \in [2, \infty)$, we get that $\sin(1/z) \leq 1/z$. Applying $\arcsin(\cdot)$ on both sides, this gives $1/z \leq \arcsin(1/z) \Leftrightarrow 1/\arcsin(1/z) \leq z$. We set $z = 2(1 + \varepsilon^{-1})$ and multiply by π to derive the desired inequality. ◀

5.1 The Routing Table

Let p be a vertex of P. We again write $\mathcal{C}$ for $\mathcal{C}(p)$ and C_j instead of $C_j(p)$. To bound the size of $\rho(p)$, we need some properties of holes with respect to cones. For $i = 0, \ldots, h-1$, we write $m(i)$ for the number of cones $C_j \in \mathcal{C}$ with $\mathcal{C}_j \sqcap \Pi_i \neq \emptyset$. Then, $\rho(p)$ contains at most $|\rho(p)| \leq \mathcal{O}\left(\sum_{i=0}^{h-1} m(i) \log n\right)$ bits. We say that Π_i is *stretched for the cone* C_j if there are indices $0 \leq j_1 < j < j_2 < t$ such that $C_{j_1} \sqcap \Pi_i$, $C_j \sqcap \Pi_i$ and $C_{j_2} \sqcap \Pi_i$ are non-empty. If Π_i is not stretched for any cone of p, then $m(i) \leq 2$. We prove the following lemma:

▶ **Lemma 5.2.** *For every* $C_j \in \mathcal{C}$*, there is at most one boundary that is stretched for* C_j*.*

Proof. Let Π_i be a hole boundary that is stretched for C_j. There are indices $j_1 < j < j_2$ and vertices $q \in C_{j_1} \sqcap \Pi_i$, $r \in C_j \sqcap \Pi_i$, and $s \in C_{j_2} \sqcap \Pi_i$. We subdivide P into three regions Q, R and S: the boundary of Q is given by the shortest path from p to r, the shortest path from p to q, and the part of Π_i from r to q not containing s. Similarly, the region R is bounded by the shortest path from p to r, the shortest path from p to s and the part of Π_i between r

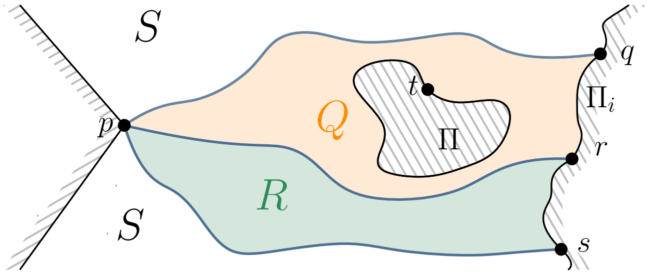


Figure 5 The shortest paths from p to q, r, s (blue). The hole Π contains t and lies in Q.

and s that does not contain q. Finally, S is the closure of $P \setminus (Q \cup R)$. The interiors of Q, R, and S are pairwise disjoint; see Figure 5.

Suppose there is another boundary Π that is stretched for C_j. Then, Π must lie entirely in either Q, R, or S. We discuss the first case, the other two are symmetric. Since Π is stretched for C_j, there is an index $j' > j$ and a vertex $t \in C_{j'} \sqcap \Pi$. Consider the shortest path π from p to t. Since $j' > j$, the first edge of π lies in R or S, and π has to cross or touch the shortest path from p to q or from q to r. Furthermore, by definition, we have $C_j \cap C_{j'} = \{p\}$ and $C_{j_1} \cap C_{j'} = \{p\}$. Therefore, p is the lowest common ancestor of all three shortest paths, and Observation 4.1 leads to a contradiction. ◀

For $i = 0, \dots, h-1$, let $s(i)$ be the number of cones in $\mathcal{C}$ for which Π_i is stretched. By Lemma 5.2, we get $\sum_{i=0}^{h-1} s(i) \le |\mathcal{C}(p)| \in \mathcal{O}(\varepsilon^{-1})$. Since $m(i) \le s(i) + 2$, we conclude

$$|\rho(p)| \in \mathcal{O}\left(\sum_{i=0}^{h-1} m(i) \log n\right) = \mathcal{O}\left(\sum_{i=0}^{h-1} (s(i)+2) \log n\right) \\ = \mathcal{O}\left((|\mathcal{C}(p)| + 2h) \log n\right) = \mathcal{O}\left((\varepsilon^{-1} + h) \log n\right).$$

5.2 The Stretch Factor

Next, we bound the stretch factor. First, we prove that the distance to the target decreases after the first step. This will then give the bound on the overall stretch.

▶ **Lemma 5.3.** *Let p and q be two vertices in P. Let s be the next vertex computed by the routing scheme for a data packet from p to q. Then, $d(s,q) \le d(p,q) - |\overline{ps}|/(1+\varepsilon)$.*

Proof. By construction of $\rho(p)$, we know that the next vertex q' on the shortest path from p to q lies in the same cone as s. Hence, by the triangle inequality and Lemma 3.1, we obtain

$$\begin{aligned} d(s,q) &\le d(s,q') + d(q',q) \le |\overline{pq'}| - \left(1 - 2\sin\frac{\pi}{t}\right)|\overline{ps}| + d(q',q) \\ &= d(p,q) - \left(1 - 2\sin\frac{\pi}{t}\right)|\overline{ps}| = d(p,q) - \left(1 - \frac{1}{1+\varepsilon^{-1}}\right)|\overline{ps}| \qquad \text{(definition of } t\text{)} \\ &= d(p,q) - |\overline{ps}|/(1+\varepsilon). \end{aligned}$$

◀

Lemma 5.3 immediately implies the correctness of the routing scheme: since the distance to the target q decreases strictly in each step and since there is a finite number of vertices, there is a $k = k(p,q) \le n$ such that after k steps, the packet arrives at q. Using this, we can now bound the stretch factor of the routing scheme.

▶ **Lemma 5.4.** *Let p and q be two vertices of P. Then, $d_\rho(p,q) \le (1+\varepsilon)d(p,q)$.*

Proof. Let $\pi = p_0p_1\dots p_k$ be the routing path from $p = p_0$ to $q = p_k$. By Lemma 5.3, we have $d(p_{i+1},q) \le d(p_i,q) - |\overline{p_ip_{i+1}}|/(1+\varepsilon)$. Thus,

$$\begin{aligned} d_\rho(p,q) &= \sum_{i=0}^{k-1} |\overline{p_ip_{i+1}}| \le (1+\varepsilon)\sum_{i=0}^{k-1}(d(p_i,q) - d(p_{i+1},q)) \\ &= (1+\varepsilon)\,(d(p_0,q) - d(p_k,q)) = (1+\varepsilon)d(p,q). \end{aligned}$$

◀

5.3 The Preprocessing Time

Finally, we discuss the details of the preprocessing algorithm and its time complexity.

▶ **Lemma 5.5.** *The preprocessing time for our routing scheme is $\mathcal{O}(n^2\log n + hn^2 + \varepsilon^{-1}hn)$.*

Proof. Let p be a vertex of P. We compute the shortest path tree T for p. Using the algorithm of Hershberger and Suri [13], this can be done in time $\mathcal{O}(n\log n)$. Now, we perform a post-order traversal of T to compute the intervals for each child of p. Given a node q, the post-order traversal provides at most h different intervals. For each hole, we compute the union of the intervals among the children. Lemma 4.2 shows that the union of these intervals is again an interval, and it can be found in time $\mathcal{O}(h\,\mathrm{outdeg}(q))$, where $\mathrm{outdeg}(q)$ is the number of q's children in T. In total, the post-order traversal needs $\mathcal{O}(hn)$ time.

Let $q_1,\dots,q_k$ be the children of p, and let $\alpha_1,\dots,\alpha_k$ be the angles between the ray $r_0(p)$ and the edges (p,q_i), $i = 1,\dots,k$. By construction, the q_i are sorted by increasing angle α_i. Into this sorted sequence, we insert the rays $r_j(p)$, and we call the resulting sequence L. By Lemma 5.1, the sequence L has $\mathcal{O}(\varepsilon^{-1} + \mathrm{outdeg}(p))$ elements. We scan through L, and between each two consecutive rays $r_{j-1}(p)$ and $r_j(p)$, we join all the corresponding intervals for each hole. Again by Lemma 4.2, this gives a set of intervals. Finally, we compute the vertex closest to p in each cone, and we store the appropriate entries in the routing table $\rho(p)$. This last step takes time $\mathcal{O}(h(\varepsilon^{-1} + \mathrm{outdeg}(p))) = \mathcal{O}(h\varepsilon^{-1} + hn)$. Thus, the preprocessing time for p is $\mathcal{O}(n\log n + hn + h\varepsilon^{-1})$, for a total of $\mathcal{O}(n^2\log n + hn^2 + \varepsilon^{-1}hn)$. ◀

Combining the last two lemmas with Section 4, we get the following theorem.

▶ **Theorem 5.6.** *Let P be a polygonal domain with n vertices and h holes. For any $\varepsilon > 0$ we can construct a routing scheme for $\mathrm{VG}(P)$ with labels of $\mathcal{O}(\log n)$ bits and routing tables of $\mathcal{O}((\varepsilon^{-1} + h)\log n)$ bits. For any two sites $p, q \in P$, the scheme produces a routing path with stretch factor at most $1+\varepsilon$. The preprocessing time is $\mathcal{O}(n^2\log n + hn^2 + \varepsilon^{-1}hn)$.*

6 Improvement for Simple Polygons

We show how to improve the preprocessing time for polygons without holes. Let P be a simple polygon with n vertices, and let $1+\varepsilon$, $\varepsilon > 0$, be the stretch factor. The previous section computes a shortest path tree for each vertex, which leads to $\mathcal{O}(n^2\log n)$ preprocessing time. In simple polygons, we can use a different technique to avoid this large overhead in the preprocessing phase. The routing function, the vertex labels, and the structure of the routing tables remain unchanged.

Let p be a vertex of P. We compute the visibility polygon $\mathrm{vis}(p)$ for p. This gives a sequence V of points $v_0, v_1, \dots v_m$ with $p = v_0 = v_m$. Some points of V may not be vertices

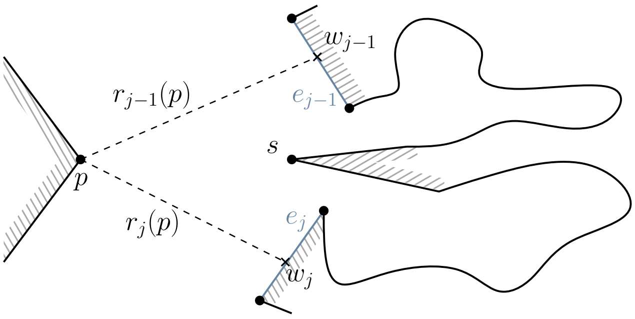


Figure 6 The boundaries of C_j hit ∂P in the points w_{j-1} and w_j. The vertex s is the vertex in C_j with smallest distance to p.

of P. We assume that V is sorted clockwise. Then, the sequence $\alpha_1, \alpha_2, \dots, \alpha_{m-1}$ of the angles α_j between the ray $r_0(p)$ and the edges $\{p, v_j\}$, $j = 1, \dots, m-1$, is increasing. For $j = 1, \dots, t-1$, let w_j be the intersection point of $r_j(p)$ and vis(p) that is closest to p. The sequence of edges e_j of P that contain the points w_j can be found in $O(n)$ time by traversing the sorted sequence V; see Figure 6.

Next, let $C_j \in \mathcal{C}$ be a cone. Recall that C_j is bounded by the rays $r_{j-1}(p)$ and $r_j(p)$. The vertices related to C_j are determined as follows: starting from w_{j-1}, we walk along the boundary of P, until we meet w_j. During the walk, we collect all the visited vertices. This set forms a (possibly empty) interval $I(j)$. We let s be the vertex in $I(j)$ with the smallest distance to p. As before, we add the endpoints of $I(j)$ together with s to $\rho(p)$. This needs $3 \cdot \lceil \log n \rceil$ bits. By Lemma 5.1, the routing table $\rho(p)$ has $\mathcal{O}(\varepsilon^{-1} \log n)$ bits, as in the previous section. To show correctness, we need the following lemma.

▶ **Lemma 6.1.** *Let p and q be two vertices of P, and let (p, q') bet the first edge on the shortest path from p to q. If $q \in I(j)$, then $q' \in C_j$.*

Proof. Suppose that $q' \notin C_j$. Since $q \in I(j)$, the shortest path π from p to q has to meet $\overline{pw_{i-1}}$ or $\overline{pw_i}$ at least twice. The first intersection is p itself. Let $a \neq p$ be the second intersection, and π' the subpath of π from p to a. By the triangle inequality $|\overline{pa}|$ is strictly smaller than the length of π'; see Figure 7. This contradicts the fact that π is a shortest path from p to q. ◀

Thus, we obtain our main theorem for simple polygons.

▶ **Theorem 6.2.** *Let P be a simple polygon with n vertices. For any $\varepsilon > 0$, we can construct a routing scheme for $\mathrm{VG}(P)$ with labels of $\lceil \log n \rceil$ bits and routing tables of $\mathcal{O}(\varepsilon^{-1} \log n)$ bits. For any two vertices $p, q \in P$, the scheme produces a routing path with stretch $1 + \varepsilon$. The preprocessing time is $\mathcal{O}(n^2 + \varepsilon^{-1} n)$.*

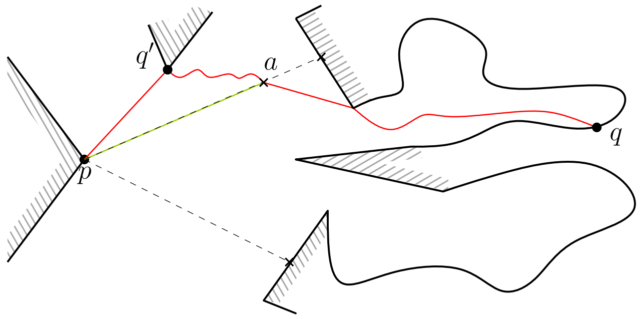


Figure 7 The red curve is the "shortest" path from p to q with q' as first step, whereas the green dashed line represents a shortcut from p to a.

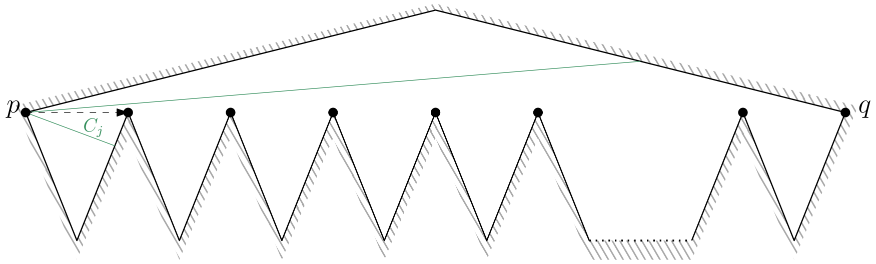


Figure 8 In this polygon, p and q can see each other, so their hop-distance is 1. Our routing scheme routes from one spire to the next, giving stretch $\Theta(n)$.

Proof. Let p be a vertex of P. First, we compute the visibility polygon of the vertex p. This needs time $\mathcal{O}(n)$ [14, 18]. Let V be the vertices of $\mathrm{vis}(p)$, sorted by increasing angle. Using V, we can find in time $\mathcal{O}(n + \varepsilon^{-1})$ all the intersection points w_j and the edges e_j of P that contain them. Finally, let C_j be a cone. We can find in constant time the endpoints of $I(j)$ and in $\mathcal{O}(|I(j)|)$ time the vertex s in $I(j)$ with the smallest distance to p. This step costs $\mathcal{O}(n + \varepsilon^{-1})$ time in total over all cones. The total running time is $\mathcal{O}(n^2 + \varepsilon^{-1}n)$. ◀

7 Conclusion

We gave an efficient routing scheme for the visibility graph of a polygonal domain. Our scheme produces routing paths whose length can be made arbitrarily close to the optimum.

Several open questions remain. First of all, we would like to obtain an efficient routing scheme for the *hop-distance* in polygonal domains P, where each edge of $\mathrm{VG}(P)$ has unit weight. For our routing scheme, we can easily construct examples where the stretch is $\Omega(n)$; see Figure 8. Moreover, it would be interesting to improve the preprocessing time or the size of the routing tables, perhaps using a recursive strategy.

A final open question concerns routing schemes in general: what is the time needed by a data packet to travel through the graph? In particular, it would be interesting to see how much time a data packet needs at one single vertex until it knows the vertex where it is forwarded. It would be a sightly different, but important measure for routing schemes.

References

1 Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *Proc. 25th DISC*, pages 404–415, 2011.

2 Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1–4):49–63, 1986.

3 Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *JALG*, 11(3):307–341, 1990.

4 Reuven Bar-Yehuda and Bernard Chazelle. Triangulating disjoint Jordan chains. *IJCGA*, 4(04):475–481, 1994.

5 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SICOMP*, 44(6):1626 – 1649, 2015.

6 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Competitive local routing with constraints. *JoCG*, 8(1):125–152, 2017.

7 Shiri Chechik. Compact routing schemes with improved stretch. In *Proc. PODC*, pages 33–41, 2013.

8 Lenore J Cowen. Compact routing with minimum stretch. *JALG*, 38(1):170–183, 2001.

9 Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *JALG*, 46(2):97–114, 2003.

10 Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. 28th ICALP*, pages 757–772, 2001.

11 Silvia Giordano and Ivan Stojmenovic. Position based routing algorithms for ad hoc networks: A taxonomy. In *Ad hoc wireless networking*, pages 103–136. Springer-Verlag, 2004.

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(1-4):209–233, 1987.

13 John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SICOMP*, 28(6):2215–2256, 1999.

14 Barry Joe and Richard B Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987.

15 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. In *Proc. 12th LATIN*, pages 536–548, 2016.

16 Sanjiv Kapoor and SN Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proc. 4th SoCG*, pages 172–182, 1988.

17 Sanjiv Kapoor, SN Maheshwari, and Joseph SB Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *DCG*, 18(4):377–383, 1997.

18 Der-Tsai Lee. Visibility of a simple polygon. *CGVIP*, 22(2):207–221, 1983.

19 Joseph SB Mitchell. A new algorithm for shortest paths among obstacles in the plane. *AMAI*, 3(1):83–105, 1991.

20 Joseph SB Mitchell. Shortest paths among obstacles in the plane. *IJCGA*, 6(03):309–332, 1996.

21 Mark H Overmars and Emo Welzl. New methods for computing visibility graphs. In *Proc. 4th SoCG*, pages 164–171, 1988.

22 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

23 Liam Roditty and Roei Tov. New routing techniques and their applications. In *Proc. PODC*, pages 23–32, 2015.

24 Liam Roditty and Roei Tov. Close to linear space routing schemes. *Distributed Computing*, 29(1):65–74, 2016.

25 Nicola Santoro and Ramez Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28(1):5–8, 1985.

26 Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. *SICOMP*, 15(1):193–215, 1986.

27 James A Storer and John H Reif. Shortest paths in the plane with polygonal obstacles. *J. ACM*, 41(5):982–1012, 1994.

28 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

29 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proc. 13th SPAA*, pages 1–10, 2001.

30 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

31 Emo Welzl. Constructing the visibility graph for n-line segments in $\mathcal{O}(n^2)$ time. *IPL*, 20(4):167–171, 1985.

32 Chenyu Yan, Yang Xiang, and Feodor F Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *CGTA*, 45(7):305–325, 2012.

33 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SICOMP*, 11(4):721–736, 1982.

Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces*

Aaron T. Becker†1, Sándor P. Fekete2, Phillip Keldenich3, Dominik Krupke4, Christian Rieck5, Christian Scheffer6, and Arne Schmidt7

1 **Department of Electrical and Computer Engineering, University of Houston, USA**
atbecker@uh.edu

2 **Department of Computer Science, TU Braunschweig, Germany**
s.fekete@tu-bs.de

3 **Department of Computer Science, TU Braunschweig, Germany**
texttp.keldenich@tu-bs.de

4 **Department of Computer Science, TU Braunschweig, Germany**
d.krupke@tu-bs.de

5 **Department of Computer Science, TU Braunschweig, Germany**
c.rieck@tu-bs.de

6 **Department of Computer Science, TU Braunschweig, Germany**
c.scheffer@tu-bs.de

7 **Department of Computer Science, TU Braunschweig, Germany**
arne.schmidt@tu-bs.de

Abstract

We present algorithmic results for the parallel assembly of many micro-scale objects in two and three dimensions from tiny particles, which has been proposed in the context of programmable matter and self-assembly for building high-yield micro-factories. The underlying model has particles moving under the influence of uniform external forces until they hit an obstacle; particles can bond when being forced together with another appropriate particle.

Due to the physical and geometric constraints, not all shapes can be built in this manner; this gives rise to the Tilt Assembly Problem (TAP) of deciding constructibility. For simply-connected polyominoes P in 2D consisting of N unit-squares ("tiles"), we prove that TAP can be decided in $O(N \log N)$ time. For the optimization variant MaxTAP (in which the objective is to construct a subshape of maximum possible size), we show *polyAPX*-hardness: unless P=NP, MaxTAP cannot be approximated within a factor of $N^{\frac{1}{3}}$; for tree-shaped structures, we give an $N^{\frac{1}{2}}$-approximation algorithm. For the efficiency of the assembly process itself, we show that any constructible shape allows *pipelined* assembly, which produces copies of P in $O(1)$ amortized time, i.e., N copies of P in $O(N)$ time steps. These considerations can be extended to three-dimensional objects: For the class of polycubes P we prove that it is NP-hard to decide whether it is possible to construct a path between two points of P; it is also NP-hard to decide constructibility of a polycube P. Moreover, it is *expAPX*-hard to maximize a path from a given start point.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Programmable matter, micro-factories, tile assembly, tilt, approximation, hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.11

[*] A full version has been made available on arxiv.org [4], https://arxiv.org/abs/1709.06299.

[†] Work from this author was partially supported by National Science Foundation IIS-1553063 and IIS-1619278.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 11; pp. 11:1–11:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In recent years, progress on flexible construction at micro- and nano-scale has given rise to a large set of challenges that deal with algorithmic aspects of programmable matter. Examples of cutting-edge application areas with a strong algorithmic flavor include self-assembling systems, in which chemical and biological substances such as DNA are designed to form predetermined shapes or carry out massively parallel computations; and swarm robotics, in which complex tasks are achieved through the local interactions of robots with highly limited individual capabilities, including micro- and nano-robots.

Moving individual particles to their appropriate attachment locations when assembling a shape is difficult because the small size of the particles limits the amount of onboard energy and computation. One successful approach to dealing with this challenge is to use molecular diffusion in combination with cleverly designed sets of possible connections: in *DNA tile self-assembly*, the particles are equipped with sophisticated bonds that ensure that only a predesigned shape is produced when mixing together a set of tiles, see [18]. The resulting study of algorithmic tile self-assembly has given rise to an extremely powerful framework and produced a wide range of impressive results. However, the required properties of the building material (which must be specifically designed and finely tuned for each particular shape) in combination with the construction process (which is left to chemical reactions, so it cannot be controlled or stopped until it has run its course) make DNA self-assembly unsuitable for some applications.

An alternative method for controlling the eventual position of particles is to apply a uniform external force, causing all particles to move in a given direction until they hit an obstacle or another blocked particle. As two of us (Becker and Fekete, [1]) have shown in the past, combining this approach with custom-made obstacles (instead of custom-made particles) allows complex rearrangements of particles, even in grid-like environments with axis-parallel motion. The appeal of this approach is that it shifts the design complexity from the building material (the tiles) to the machinery (the environment). As recent practical work by Manzoor et al. [15] shows, it is possible to apply this to simple "sticky" particles that can be forced to bond, see Fig. 1: the overall assembly is achieved by adding particles one at a time, attaching them to the existing sub-assembly. Moreover, pipelining this process may result in efficient rates of production, see Fig. 2 [15].

One critical issue of this approach is the requirement of getting particles to their destination without being blocked by or bonding to other particles. As Fig. 3 shows, this is not always possible, so there are some shapes that cannot be constructed by Tilt Assembly.

This gives rise to a variety of algorithmic questions: (1) Can we decide efficiently whether a given polyomino can be constructed by Tilt Assembly? (2) Can the resulting process be pipelined to yield low amortized building time? (3) Can we compute a maximum-size subpolyomino that can be constructed? (4) What can be said about three-dimensional versions of the problem?

1.1 Our Contribution

We present the results shown in Table 1.

1.2 Related Work

Assembling polyominoes with tiles has been considered intensively in the context of *tile self-assembly*. In 1998, Erik Winfree [18] introduced the *abstract tile self-assembly model*

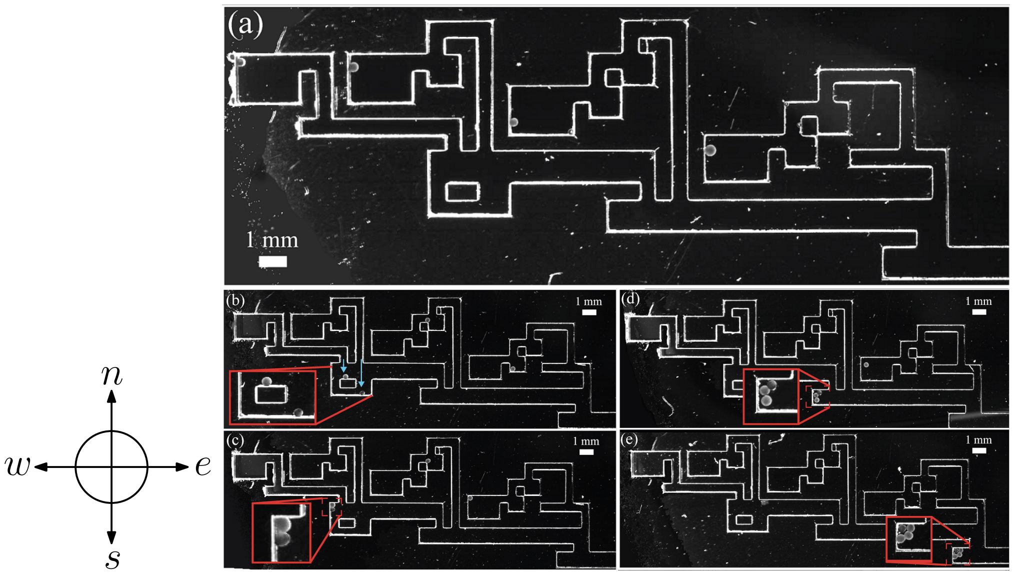


Figure 1 A practical demonstration of Tilt Assembly based on alginate (i.e., a gel made by combining a powder derived from seaweed with water) particles [15]. (a) Alginate particles in initial positions. (b) After control moves of $\langle e, s, w, n, e, s\rangle$ (for east, south, west, north), the alginate microrobots move to the shown positions. (c) After $\langle w, n\rangle$ inputs, the system produces the first multi-microrobot polyomino. (d) The next three microrobot polyominoes are produced after applying multiple $\langle e, s, w, n\rangle$ cycles. (e) After the alginate microrobots have moved through the microfluidic factory layout, the final 4-particle polyomino is generated.

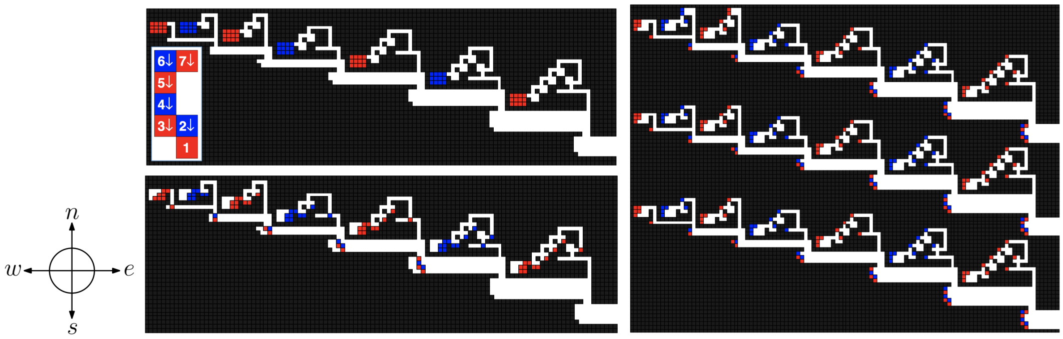


Figure 2 (Top left) Initial setup of a seven-tile polyomino assembly; the composed shape is shown enlarged on the lower left. The bipartite decomposition into blue and red particles is shown for greater clarity, but can also be used for better control of bonds. The sequence of control moves is $\langle e, s, w, n\rangle$, i.e., a clockwise order. (Bottom left) The situation after 18 control moves. (Right) The situation after 7 full cycles, i.e., after 28 control moves; shown are three parallel "factories".

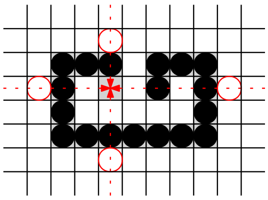


Figure 3 A polyomino (black) that cannot be constructed by Tilt Assembly: the last tile cannot be attached, as it gets blocked by previously attached tiles.

Table 1 Results for Tilt Assembly Problem (TAP) and its maximization variant (MAXTAP).

Dimension	Decision	Maximization	Approximation	Constructible Path
2D (simple)	$O(N \log N)$ (Sec. 3)	*polyAPX*-hard	$\Omega(N^{1/3})$, $O(\sqrt{N})$ (Sec. 4)	$O(N \log N)$(Sec. 4)
3D (general)	NP-hard (Sec. 5)	*polyAPX*-hard	$\Omega(N^{1/3})$, - (Sec. 4)	NP-hard (Sec. 5)

(aTAM), in which tiles have glue types on each of the four sides and two tiles can stick together if their glue type matches and the bonding strength is sufficient. Starting with a *seed tile*, tiles will continue to attach to the existing partial assembly until they form a desired polyomino; the process stops when no further attachments are possible. Apart from the aTAM, there are various other models like the *two-handed tile self-assembly model* (2HAM) [8] and the *hierarchical tile self-assembly model* [9], in which we have no single seed but pairs of subassemblies that can attach to each other. Furthermore, the *staged self-assembly model* [10, 11] allows greater efficiency by assembling polyominoes in multiple bins which are gradually combined with the content of other bins.

All this differs from the model in Tilt Assembly, in which each tile has the same glue type on all four sides, and tiles are added to the assembly one at a time by attaching them from the outside along a straight line. This approach of externally movable tiles has actually been considered in practice at the microscale level using biological cells and an MRI, see [12], [13], [5]. Becker et al. [6] consider this for the assembly of a magnetic *Gauß gun*, which can be used for applying strong local forces by very weak triggers, allowing applications such as micro-surgery.

Using an external force for moving the robots becomes inevitable at some scale because the energy capacity decreases faster than the energy demand. A consequence is that all non-fixed robots/particles perform the same movement, so all particles move in the same direction of the external force until they hit an obstacle or another particle. These obstacles allow shaping the particle swarm. Designing appropriate sets of obstacles and moves gives rise to a range of algorithmic problems. Deciding whether a given initial configuration of particles in a given environment can be transformed into a desired target configuration is NP-hard [1], even in a grid-like setting, whereas finding an optimal control sequence is shown to be PSPACE-complete by Becker et al. [2]. However, if it is allowed to *design* the obstacles in the first place, the problems become much more tractable [1]. Moreover, even complex computations become possible: If we allow additional particles of double size (i.e., two adjacent fields), full computational complexity is achieved, see Shad et al. [16]. Further related work includes gathering a particle swarm at a single position [14] and using swarms of very simple robots (such as Kilobots) for moving objects [7]. For the case in which human controllers have to move objects by such a swarm, Becker et al. [3] study different control options. The results are used by Shahrokhi and Becker [17] to investigate an automatic controller.

Most recent and most closely related to our paper is the work by Manzoor et al. [15], who use global control to assembly polyominoes in a pipelined fashion: after constructing the first polyomino, each cycle of a small control sequence produces another polyomino. However, the algorithmic part is purely heuristic; providing a thorough understanding of algorithms and complexity is the content of our paper.

2 Preliminaries

Polyomino. For a set $P \subset \mathbb{Z}^2$ of N grid points in the plane, the graph G_P is the induced grid graph, in which two vertices $p_1, p_2 \in P$ are connected if they are at unit distance. Any set P with connected grid graph G_P gives rise to a *polyomino* by replacing each point $p \in P$ by a unit square centered at p, which is called a *tile*; for simplicity, we also use P to denote the polyomino when the context is clear, and refer to G_P as the dual graph of the polyomino; P is *tree-shaped*, if G_P is a tree.

A polyomino is called *hole-free* or *simple* if and only if the grid graph induced by $\mathbb{Z}^2 \setminus P$ is connected.

Blocking sets. For each point $p \in \mathbb{Z}^2$ we define *blocking sets* N_p, $S_p \subseteq P$ as the set of all points $q \in P$ that are above or below p and $|p_x - q_x| \leq 1$. Analogously, we define the blocking sets E_p, $W_p \subseteq P$ as the set of all points $q \in P$ that are to the right or to the left of p and $|p_y - q_y| \leq 1$.

Construction step. A *construction step* is defined by a direction (north, east, south, west, abbreviated by n, e, s, w) from which a tile is added and a latitude/longitude l describing a column or row. The tile arrives from (l, ∞) for north, (∞, l) for east, $(l, -\infty)$ for south, and $(-\infty, l)$ for west into the corresponding direction until it reaches the first grid position that is adjacent to one occupied by an existing tile. If there is no such tile, the polyomino does not change. We note that a position p can be added to a polyomino P if and only if there is a point $q \in P$ with $||p - q||_1 = 1$ and one of the four blocking sets, N_p, E_p, S_p or W_p, is empty. Otherwise, if none of these sets are empty, this position is *blocked.*

Constructibility. Beginning with a seed tile at some position p, a polyomino P is *constructible* if and only if there is a sequence $\sigma = ((d_1, l_1), (d_2, l_2), \ldots, (d_{N-1}, l_{N-1}))$, such that the resulting polyomino P', induced by successively adding tiles with σ, is equal to P. We allow the constructed polyomino P' to be a translated copy of P. Reversing σ yields a *decomposition sequence*, i.e., a sequence of tiles getting removed from P.

3 Constructibility of Simple Polyominoes

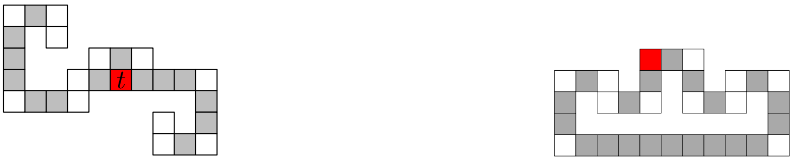
In this section we focus on hole-free (i.e., simple) polyominoes. We show that the problem of deciding whether a given polyomino can be constructed can be solved in polynomial time. This decision problem can be defined as follows.

▶ **Definition 1** (Tilt Assembly Problem). Given a polyomino P, the Tilt Assembly Problem (TAP) asks for a sequence of tiles constructing P, if P is constructible.

3.1 A Key Lemma

A simple observation is that construction and (restricted) decomposition are the same problem. This allows us to give a more intuitive argument, as it is easier to argue that we do not lose connectivity when removing tiles than it is to prove that we do not block future tiles.

▶ **Theorem 2.** *A polyomino P can be constructed if and only if it can be decomposed using a sequence of tile removal steps that preserve connectivity. A construction sequence is a reversed decomposition sequence.*



(a) Removing t destroys decomposability. The polyomino can be decomposed by starting with the three tiles above t.

(b) Removing the red convex tile leaves the polyomino non-decomposable; it can be decomposed by starting from the bottom or the sides.

Figure 4 Two polyominoes and their convex tiles (white). **(a)** Removing non-convex tiles may destroy decomposability. **(b)** In case of non-simple polygons we may not be able to remove convex tiles.

Proof. To prove this theorem, it suffices to consider a single step. Let P be a polyomino and t be a tile that is removed from P into some direction l, leaving a polyomino P'. Conversely, adding t to P' from direction l yields P, as there cannot be any tile that blocks t from reaching the correct position, or we would not be able to remove t from P in direction l. ◀

For hole-free polyominoes we can efficiently find a construction/decomposition sequence if one exists. The key insight is that one can greedily remove *convex* tiles. A tile t is said to be convex if and only if there is a 2×2 square solely containing t; see Fig. 4. If a convex tile is *not* a cut tile, i.e., it is a tile whose removal does *not* disconnect the polyomino, its removal does not interfere with the decomposability of the remaining polyomino.

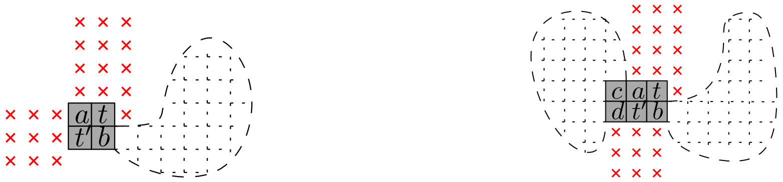
This conclusion is based on the observation that a minimal cut (i.e., a minimal set of vertices whose removal leaves a disconnected polyomino) of cardinality two in a hole-free polyomino always consists of two (possibly diagonally) adjacent tiles. Furthermore, we can always find such a removable convex tile in any decomposable hole-free polyomino. This allows us to devise a simple greedy algorithm.

We start by showing that if we find a non-blocked convex tile that is not a cut tile, we can simply remove it. It is important to focus on convex tiles, as the removal of non-convex tiles can harm the decomposability: see Fig. 4a for an illustration. In non-simple polyominoes, the removal of convex tiles can destroy decomposability, as demonstrated in Fig. 4b.

▶ **Lemma 3.** *Consider a non-blocked non-cut convex tile t in a hole-free polyomino P. The polyomino $P - t$ is decomposable if and only if P is decomposable.*

Proof. The first direction is trivial: if $P - t$ is decomposable, P is decomposable as well, because we can remove the non-blocked tile t first and afterwards use the existing decomposition sequence for $P - t$. The other direction requires some case distinctions. Suppose for contradiction that P is decomposable but $P - t$ is not, i.e., t is important for the later decomposition.

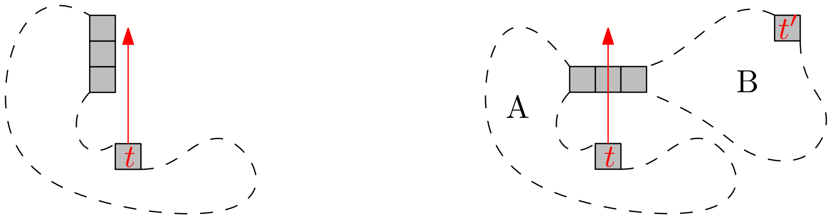
Consider a valid decomposition sequence for P and the first tile t' we cannot remove if we were to remove t in the beginning. W.l.o.g., let t' be the first tile in this sequence (removing all previous tiles obviously does not destroy the decomposability). When we remove t first, we are missing a tile, hence t' cannot be blocked but has to be a cut tile in the remaining polyomino $P - t$. The presence of t preserves connectivity, i.e., $\{t, t'\}$ is a minimal cut on P. Because P has no holes, then t and t' must be diagonal neighbors, sharing the neighbors a and b. Furthermore, by definition neither of t and t' is blocked in some direction. We make a case distinction on the relation of these two directions.



(a) If the unblocked directions of t and t' are orthogonal, one of the two adjacent tiles (w.l.o.g. a) cannot have any further neighbors. There can also be no tiles in the upper left corner, because the polyomino cannot cross the two free directions of t and t' (red marks).

(b) If the unblocked directions of t and t' are parallel, there is only the tile c for which something can change if we remove t before t'.

Figure 5 The red marks indicate that no tile is at this position; the dashed outline represents the rest of the polyomino.



(a) If the removal direction of t is not crossed, the last blocking tile has to be convex (and has to be removed before).

(b) If the removal direction of t crosses P, then P gets split into components A and B. Component B has a convex tile t' that needs to be removed before t.

Figure 6 Polyominoes for which no convex tile should be removable, showing the contradiction to t being the first blocked convex tile in P getting removed.

The directions are orthogonal (Fig. 5a). Either a or b is a non-blocked convex tile, because t and t' are both non-blocked; w.l.o.g., let this be a. It is easy to see that independent of removing t or t' first, after removing a we can also remove the other one.

The directions are parallel (Fig. 5b). This case is slightly more involved. By assumption, we have a decomposition sequence beginning with t'. We show that swapping t' with our convex tile t in this sequence preserves feasibility.

The original sequence has to remove either a or b before it removes t, as otherwise the connection between the two is lost when t' is removed first. After either a or b is removed, t becomes a leaf and can no longer be important for connectivity. Thus, we only need to consider the sequence until either a or b is removed. The main observation is that a and b block the same tiles as t or t', except for tile c as in Fig. 5b. However, when c is removed, it has to be a leaf, because a is still not removed and in the original decomposition sequence, t' has already been removed. Therefore, a tile $d \neq t'$ would have to be removed before c. Hence, the decomposition sequence remains feasible, concluding the proof. ◀

Next we show that such a convex tile always exists if the polyomino is decomposable.

▶ **Lemma 4.** *Let P be a decomposable polyomino. Then there exists a convex tile that is removable without destroying connectivity.*

Proof. We prove this by contradiction based on two possible cases.

Assume P to be a decomposable polyomino in which no convex tile is removable. Because P is decomposable, there exists some feasible decomposition sequence S. Let P_{convex} denote the set of convex tiles of P and let $t \in P_{\text{convex}}$ be the first removed convex tile in the decomposition sequence S. By assumption, t cannot be removed yet, so it is either blocked or a cut tile.

t is blocked. Consider the direction in which we would remove t. If it does not cut the polyomino, the last blocking tile has to be convex (and would have to be removed before t), see Fig. 6a. If it cuts the polyomino, the component cut off also must have a convex tile and the full component has to be removed before t, see Fig. 6b. This is again a contradiction to t being the first convex tile to be removed in S.

t is a cut tile. $P - t$ consists of exactly two connected polyominoes, P_1 and P_2. It is easy to see that $P_1 \cap P_{\text{convex}} \neq \emptyset$ and $P_2 \cap P_{\text{convex}} \neq \emptyset$, because every polyomino of size $n \geq 2$ has at least two convex tiles of which at most one becomes non-convex by adding t. (A polyomino of size 1 is trivial.) Before being able to remove t, either P_1 or P_2 has to be completely removed, including their convex tiles. This is a contradiction to t being the first convex tile in S to be removed. ◀

3.2 An Efficient Algorithm

An iterative combination of these two lemmas proves the correctness of greedily removing convex tiles. As we show in the next theorem, using a search tree technique allows an efficient implementation of this greedy algorithm.

▶ **Theorem 5.** *A hole-free polyomino can be checked for decomposability/constructibility in time $O(N \log N)$.*

Proof. Lemma 3 allows us to remove any convex tile, as long as it is not blocked and does not destroy connectivity. Applying the same lemma on the remaining polyomino iteratively creates a feasible decomposition sequence. Lemma 4 proves that this is always sufficient. If and only if we can at some point no longer find a matching convex tile (to which we refer as *candidates*), the polyomino cannot be decomposable.

Let B be the time needed to check whether a tile t is blocked. A naïve way of doing this is to try out all tiles and check if t gets blocked, requiring time $O(N)$. With a preprocessing step, we can decrease B to $O(\log N)$ by using $O(N)$ binary search trees for searching for blocking tiles and utilizing that removing a tile can change the state of at most $O(1)$ tiles. For every vertical line x and horizontal line y going through P, we create a balanced search tree, i.e., for a total of $O(N)$ search trees. An x-search tree for a vertical line x contains tiles lying on x, sorted by their y-coordinate. Analogously define a y-search tree for a horizontal line y containing tiles lying on y sorted by their x-coordinate. We iterate over all tiles $t = (x, y)$ and insert the tile in the corresponding x- and y-search tree with a total complexity of $O(N \log N)$. Note that the memory complexity remains linear, because every tile is in exactly two search trees. To check if a tile at position (x', y') is blocked from above, we can simply search in the $(x' - 1)$-, x'- and $(x' + 1)$-search tree for a tile with $y > y'$. We analogously perform search queries for the other three directions, and thus have 12 queries of total cost $O(\log N)$.

We now iterate on all tiles and add all convex tiles that are not blocked and are not a cut tile to the set F (cost $O(N \log N)$). Note that checking whether a tile is a cut tile can be done in constant time, because it suffices to look into the local neighborhood. While F is not empty, we remove a tile from F, from the polyomino, and from its two search trees in

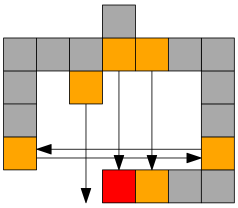


Figure 7 When removing the red tile, only the orange tiles can become unblocked or convex.

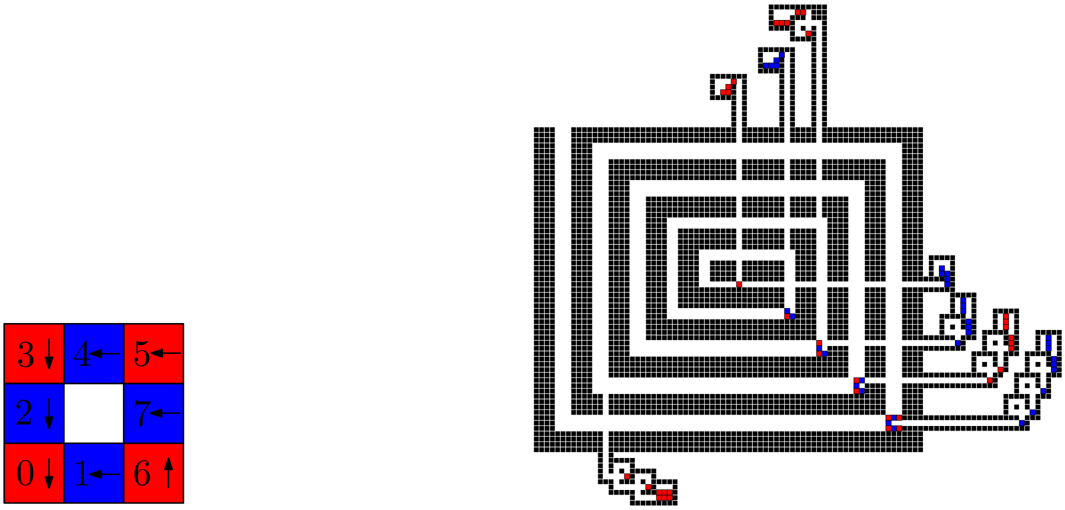


Figure 8 (Left) A polyomino P. Shown is the assembly order and the direction of attachment to the seed (tile 0). (Right) A maze environment for pipelined construction of the desired polyomino P. After the fourth cycle, each further cycle produces a new copy of P.

time $O(\log N)$. Next, we check the up to 12 tiles that are blocked first from the removed tile for all four orientations, see Fig. 7. Only these tiles can become unblocked or a convex tile. Those that are convex tiles, not blocked and no cut tile are added to F. All tiles behind those cannot become unblocked as the first tiles would still be blocking them. The cost for this is again in $O(\log N)$. This is continued until F is empty, which takes at most $O(N)$ loops each of cost $O(\log N)$. If the polyomino has been decomposed, the polyomino is decomposable/constructible by the corresponding tile sequence. Otherwise, there cannot exist such a sequence. By prohibiting to remove a specific tile, one can force a specific start tile. ◀

3.3 Pipelined Assembly

Given that a construction is always possible based on adding convex corners to a partial construction, we can argue that the idea of Manzoor et al. [15] for pipelined assembly can be realized for *every* constructible polyomino: We can transform the construction sequence into a spiral-shaped maze environment, as illustrated in Fig. 8. This allows it to produce D copies of P in $N + D$ cycles, implying that we only need $2N$ cycles for N copies. It suffices to use a clockwise order of four unit steps (west, north, east, south) in each cycle.

The main idea is to create a spiral in which the assemblies move from the inside to the outside. The first tile is provided by an initial south movement. After each cycle, ending with a south movement, the next seed tile of the next copy of P is added. For every direction corresponding to the direction of the next tile added by the sequence, we place a tile depot on the outside of the spiral, with a straight-line path to the location of the corresponding attachment.

▶ **Theorem 6.** *Given a construction sequence* $\sigma := ((d_1, l_1), \ldots, (d_{N-1}, l_{N-1}))$ *that constructs a polyomino* P*, we can construct a maze environment for pipelined tilt assembly, such that constructing* D *copies of* P *needs* $O(N + D)$ *unit steps. In particular, constructing one copy of* P *can be done in amortized time* $O(1)$*.*

A more detailed proof can be found in the full version of this paper [4].

4 Optimization Variants in 2D

For polyominoes that cannot be assembled, it is natural to look for a maximum-size subpolyomino that is constructible. As it turns out, this optimization variant is polyAPX-hard, i.e., we cannot hope for an approximation algorithm with an approximation factor within $\Omega(N^{\frac{1}{3}})$, unless P = NP.

▶ **Definition 7** (Maximum Tilt Assembly Problem)**.** Given a polyomino P, the Maximum Tilt Assembly Problem (MaxTAP) asks for a sequence of tiles building a cardinality-maximal connected subpolyomino $P' \subseteq P$.

▶ **Theorem 8.** MaxTAP *is polyAPX-hard, even for tree-shaped polyominoes, and cannot be approximated within a factor of* $\Omega(N^{\frac{1}{3}})$*.*

The proof is based on a reduction from Maximum Independent Set (MIS) to MaxTAP. See full version [4] for details. On the positive side, we can give an $O(\sqrt{N})$-approximation algorithm.

▶ **Theorem 9.** *The longest constructible path in a tree-shaped polyomino* P *is a* $\sqrt{N}$*-approximation for* MaxTAP*, and we can find such a path in polynomial time.*

Proof. Consider an optimal solution P^* and a smallest enclosing box B containing P^*. Then there must be two opposite sides of B having at least one tile of P^*. Consider the path S between both tiles. Because the area A_B of B is at least the number of tiles in P^*, $|S| \geq \sqrt{A_B}$ and a longest, constructible path in P has length at least $|S|$, we conclude that the longest constructible path is a $\sqrt{N}$-approximation.

To find such a path, we can search for every path between two tiles, check whether we can build this path, and take the longest, constructible path. ◀

Checking constructibility for $O(N^2)$ possible paths is rather expensive. However, we can efficiently approximate the longest constructible path in a tree-shaped polyomino with the help of *sequentially* constructible paths, i.e., the initial tile is a leaf in the final path.

▶ **Theorem 10.** *We can find a constructible path in a tree-shaped polyomino in* $O(N^2 \log N)$ *time that has a length of at least half the length of the longest constructible path.*

Proof. We only search for paths that can be built sequentially. Clearly, the longest such path is at least half as long as the longest path that can have its initial tile anywhere. We use the same search tree technique as before to look for blocking tiles. Select a tile of the polyomino as the initial tile. Do a depth-first search and for every tile in this search, check if it can be added to the path. If it cannot be added, skip all deeper tiles, as they also cannot be added. During every step in the depth-first search, we only need to change a single tile in the search trees, doing $O(1)$ updates with $O(\log N)$ cost. As we only consider $O(N)$ vertices in the depth-first search, this results in a cost of $O(N \log N)$ for a fixed start tile. It is trivial to keep track of the longest such constructible path. Repeating this for every tile results in a running time of $O(N^2 \log N)$. ◀

In tree-shaped polyominoes, finding a constructible path is easy. For simple polyominoes, additional arguments and data structures lead to a similar result.

▶ **Theorem 11.** *In simple polyominoes, finding the longest of all shortest paths that are sequentially constructible takes $O(N^2 \log N)$ time.*

See full version [4] for details.

5 Three-Dimensional Shapes

An interesting and natural generalization of TAP is to consider three-dimensional shapes, i.e., polycubes. As it turns out, the local considerations for simply connected two-dimensional shapes are no longer sufficient. In the following we show that deciding whether a polycube is constructible is NP-hard. Moreover, it is NP-hard to check whether there is a constructible path from a start cube s to an end cube t in a partial shape.

As a stepping stone, we start with a restricted version of the three-dimensional problem.

▶ **Theorem 12.** *It is* NP*-hard to decide if a polycube can be built.*

The proof is based on a reduction from 3SAT and omitted for lack of space; see full version [4] for details.

The difficulties of construction in 3D are highlighted by the fact that even identifying constructible connections between specific positions is NP-hard.

▶ **Theorem 13.** *It is* NP*-hard to decide whether a path from one tile to another can be built in a general polycube.*

The proof proceeds by a reduction from SAT; see full version [4] for details.

6 Conclusion/Future Work

We have provided a number of algorithmic results for Tilt Assembly. Various unsolved challenges remain. What is the complexity of deciding TAP for non-simple polyominoes? While Lemma 4 can be applied to all polyominoes, we cannot simply remove any convex tile. Clearly, TAP $\in$ NP if the polyomino is encoded tile by tile, which is usually done in practice. Can we find a constructible path in a polyomino from a given start and endpoint? This would help in finding a $\sqrt{N}$-approximation for non-simple polyominoes. How can we optimize the total makespan for constructing a shape? And what options exist for non-constructible shapes?

An interesting approach may be to consider *staged* assembly, as shown in Fig. 9, where a shape gets constructed by putting together subpolyominoes, instead of adding one tile at a time. This is similar to staged tile self-assembly [10, 11]. This may also provide a path to sublinear assembly times, as a hierarchical assembly allows massive parallelization. We conjecture that a makespan of $O(\sqrt{N})$ for a polyomino with N tiles can be achieved.

All this is left to future work.

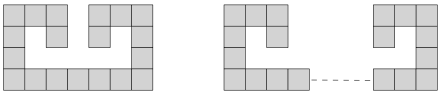


Figure 9 (Left) A polyomino that cannot be constructed in the basic TAP model. (Right) Construction in a staged assembly model by putting together subpolyominoes.

References

1 Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Golnaz Habibi, and James McLurkin. Reconfiguring massive particle swarms with limited, global control. In *Proc. Int. Symp. Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, pages 51–66, 2013.

2 Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, and James McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 6751–6756, 2014.

3 Aaron T. Becker, Chris Ertel, and James McLurkin. Crowdsourcing swarm manipulation experiments: A massive online user study with large swarms of simple robots. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2825–2830, 2014.

4 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces. *arXiv preprint arXiv:1709.06299*, 2017.

5 Aaron T. Becker, Ouajdi Felfoul, and Pierre E. Dupont. Simultaneously powering and controlling many actuators with a clinical MRI scanner. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 2017–2023, 2014.

6 Aaron T. Becker, Ouajdi Felfoul, and Pierre E. Dupont. Toward tissue penetration by MRI-powered millirobots using a self-assembled Gauss gun. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1184–1189, 2015.

7 Aaron T. Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 520–527, 2013.

8 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors). In *Proc. Int. Symp. on Theoretical Aspects of Computer Science(STACS)*, pages 172–184, 2013.

9 Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. *SIAM Journal on Computing*, 46(2):661–709, 2017.

10 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with O(1) glues. *Natural Computing*, 7(3):347–370, 2008.

11 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017.

12 Paul Seung Soo Kim, Aaron T. Becker, Yan Ou, Anak Agung Julius, and Min Jun Kim. Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control. *Journal of Nanoparticle Research*, 17(3):1–15, 2015.

13 Paul Seung Soo Kim, Aaron T. Becker, Yan Ou, Min Jun Kim, et al. Swarm control of cell-based microrobots using a single global magnetic field. In *Proc. Int. Conf. Ubiquitous Robotics and Ambient Intelligence (URAI)*, pages 21–26, 2013.

14 Arun V. Mahadev, Dominik Krupke, Jan-Marc Reinhardt, Sándor P. Fekete, and Aaron T. Becker. Collecting a swarm in a grid environment using shared, global inputs. In *Proc. IEEE Int. Conf. Autom. Sci. and Eng. (CASE)*, pages 1231–1236, 2016.

15 Sheryl Manzoor, Samuel Sheckman, Jarrett Lonsford, Hoyeon Kim, Min Jun Kim, and Aaron T. Becker. Parallel self-assembly of polyominoes under uniform control inputs. *IEEE Robotics and Automation Letters*, 2(4):2040–2047, 2017.

16 Hamed Mohtasham Shad, Rose Morris-Wright, Erik D. Demaine, Sándor P. Fekete, and Aaron T. Becker. Particle computation: Device fan-out and binary memory. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 5384–5389, 2015.

17 Shiva Shahrokhi and Aaron T. Becker. Stochastic swarm control with global inputs. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 421–427, 2015.

18 Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

A Simple Greedy Algorithm for Dynamic Graph Orientation*

Edvin Berglin[1] **and Gerth Stølting Brodal**[2]

1 **Department of Computer Science, Aarhus University, Aarhus, Denmark**
berglin@cs.au.dk

2 **Department of Computer Science, Aarhus University, Aarhus, Denmark**
gerth@cs.au.dk

Abstract

Graph orientations with low out-degree are one of several ways to efficiently store sparse graphs. If the graphs allow for insertion and deletion of edges, one may have to *flip* the orientation of some edges to prevent blowing up the maximum out-degree. We use arboricity as our sparsity measure. With an immensely simple greedy algorithm, we get parametrized trade-off bounds between out-degree and worst case number of flips, which previously only existed for amortized number of flips. We match the previous best worst-case algorithm (in $\mathcal{O}(\log n)$ flips) for general arboricity and beat it for either constant or super-logarithmic arboricity. We also match a previous best amortized result for at least logarithmic arboricity, and give the first results with worst-case $\mathcal{O}(1)$ and $\mathcal{O}(\sqrt{\log n})$ flips nearly matching degree bounds to their respective amortized solutions.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Dynamic graph algorithms, graph arboricity, edge orientations

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.12

1 Introduction

An important building block in algorithmic theory and practice is the ability to store graphs with low memory usage and fast query times. Classical storage methods are edge lists and adjacency matrix, but both have pitfalls for *sparse* graphs: adjacency matrices use too much memory, while edge lists can have slow adjacency queries and/or updates on high-degree vertices. Much research has been devoted to improving these simple methods. The graph parameter *arboricity* α is a well-known measure of a graph's sparsity, which captures the minimum number of forests the edges of a graph can be partitioned into. Kannan et al. [6] showed how to efficiently store static graphs with low arboricity and supporting fast ($\mathcal{O}(\alpha)$ time) adjacency queries in the worst case.

Brodal and Fagerberg [3] extended this idea to consider *dynamic* graphs, where edges may be arbitrarily inserted or deleted. If the arboricity of the graphs remains bounded by a constant α, the forest partitions may be forced to change due to the updates. The authors deal with this by considering the problem of orienting the edges of the dynamic graph as in [6], but by re-orienting ("flipping") edges as needed to maintain low out-degree. They gave a simple greedy algorithm and proved that its amortized number of flips was $\mathcal{O}(1)$-competitive to the number of flips made by any other algorithm – even if that other algorithm is afforded unlimited computational resources and knowledge of the entire sequence

* Work supported by the Danish National Research Foundation grant DNRF84 through the Center for Massive Data Algorithmics (MADALGO).

Table 1 Previous and new results for the dynamic edge orientation of dynamic graphs with bounded arboricity α. Flip bounds are either amortized (am.) or worst-case (w.c.) per update.

Reference	Out-degree	Flips	α known	Note
Brodal & Fagerberg [3]	$\mathcal{O}(\alpha)$	$\mathcal{O}(\log n)$ am.	yes	$\Omega(n)$ worst-case flips
Kowalik [8]	$\mathcal{O}(\alpha \log n)$	$\mathcal{O}(1)$ am.	yes	uses alg. from [3]
Kopelowitz et al. [7]	$\mathcal{O}(\alpha + \log n)$	$\mathcal{O}(\alpha + \log n)$ w.c.	no	
Kopelowitz et al. [7]	$\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$	$\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ w.c.	no	if $\alpha = \mathcal{O}\left(\sqrt{\log n}\right)$
He et al. [5]	$\mathcal{O}\left(\alpha \sqrt{\log n}\right)$	$\mathcal{O}\left(\sqrt{\log n}\right)$ am.	yes	uses alg. from [3]
He et al. [5]	$\mathcal{O}(\alpha \log n)$	$\mathcal{O}(\alpha \log n)$ w.c.	no	
New (Corollary 18)	$\mathcal{O}(\alpha + \log n)$	$\mathcal{O}(\log n)$ w.c.	no	
New (Corollary 19)	$\mathcal{O}(\alpha \log n)$	$\mathcal{O}\left(\sqrt{\log n}\right)$ w.c.	no	
New (Corollary 20)	$\mathcal{O}(\log n)$	$\mathcal{O}\left(\alpha \sqrt{\log n}\right)$ w.c.	yes	if $\alpha = \mathcal{O}\left(\sqrt{\log n}\right)$
New (Corollary 17)	$\mathcal{O}\left(\alpha \log^2 n\right)$	$\mathcal{O}(1)$ w.c.	no	
New (Corollary 16)	$\mathcal{O}\left(\frac{\alpha \log^2 n}{f(n)}\right)$	$\mathcal{O}(f(n))$ w.c.	no	if $f(n) = \mathcal{O}(\log n)$

of updates in advance. In this paper, we will use the term ‘offline strategy’ to describe such an algorithm. In particular, Brodal and Fagerberg showed how to maintain the out-degrees bounded by $\mathcal{O}(\alpha)$ with $\mathcal{O}(\log n)$ amortized flips, where n is the number of vertices in the graph. They also gave a lower bound of $\Omega(n)$ flips for maintaining the out-degrees bounded by α. It is not hard to see that this bound holds even for $\alpha = 1$.

Kowalik [8] gave another offline strategy and applied it to the algorithm by Brodal and Fagerberg, getting $\mathcal{O}(\alpha \log n)$ out-degree in constant amortized flips, demonstrating that a reasonable trade-off was possible. Both the algorithms of Brodal and Fagerberg [3] and Kowalik [8] need to know, and use as a parameter, a bound on the arboricity of the graph.

Kopelowitz et al. [7] later found a different algorithm, which came with slightly worse bounds but in the worst case rather than amortized. Their algorithm maintains $\mathcal{O}(\alpha + \log n)$ out-degree with $\mathcal{O}(\alpha + \log n)$ flips, without knowing α. However, if α is known, they give an alternate algorithm with somewhat faster running time but otherwise equal bounds. Also, if $\alpha = \mathcal{O}\left(\sqrt{\log n}\right)$, both bounds can be improved slightly to $\mathcal{O}(\log n / \log \log n)$ due to some freedom in setting the base of the logarithmic terms.

He et al. [5] gave a new offline strategy with a parametrized trade-off between out-degree and flips, generalizing the two strategies in [3] and [8]. When applied to the algorithm by Brodal and Fagerberg it achieves $\mathcal{O}\left(\alpha \sqrt{\log n}\right)$ out-degree with $\mathcal{O}\left(\sqrt{\log n}\right)$ amortized flips. They also give another algorithm with worst-case bounds, nearly matching those in [7] but with somewhat simpler pseudocode.

The problem was originally motivated by quick adjacency queries [6]. But rather than making an explicit dictionary data structure, we focus on the problem of dynamically flipping edges to guarantee low maximum out-degree. This allows us to ignore lower bounds for dictionary operations, and we deliberately omit comparisons of update time complexity as they might be skewed unfairly in our favor. It is straightforward to create such a data structure on top of our machinery, should one so desire, by extending our solution to report which edges are flipped. This allows programmers to tailor the balance between update and query time to suit their own needs.

Dynamic edge orientations have recently become a very popular building block in dynamic graph algorithms, especially for maintaining maximal matchings; see e.g. [1] [2] [10] [11] [12]. For an overview of other applications, we refer to the appendix in the full version of [7].

1.1 Our contribution

We present a new algorithm for maintaining an edge orientation of a dynamic graph, with a guarantee of low out-degree and worst-case number of flips. Like many previous solutions we relate the performance to the arboricity α of the dynamic graph, but unlike some previous works, ours does not require knowledge of the arboricity in the general case. Our algorithm is furthermore much simpler than previous ones, and uses queues as the only under-the-hood data structure. It owes its simplicity to the fact that it greedily chooses which edge to flip.

By controlling a run-time parameter, our algorithm allows a user-specified trade-off between the out-degree and the number of flips; this was previously only possible for algorithms with *amortized* number of flips. Depending on the choice of the parameter, the algorithm can maintain e.g. $\mathcal{O}(\alpha + \log n)$ out-degree with $\mathcal{O}(\log n)$ flips, or $\mathcal{O}\big(\alpha \log^2 n\big)$ out-degree with constant flips. Various other parameter settings are possible. We match or improve all known bounds with worst-case flips, except when the arboricity is within a specific, very narrow range.

2 Preliminaries

The *arboricity* of a graph G is the smallest number t such that the edges of G can be partitioned into t forests. Several equivalent definitions are used throughout the literature. We use arboricity(G) to denote the arboricity of G. A graph G with bounded arboricity arboricity$(G) \leq \alpha$ is *sparse*: any induced subgraph of G on $n' \leq n$ vertices contains at most $(n'-1)\alpha$ edges. Note that while bounded arboricity graphs have no dense neighbourhood they can still have vertices of arbitrarily high degree, e.g. stars have arboricity 1 but maximum degree $n-1$.

We say that $\mathcal{G} = G_0, G_1, G_2, \ldots, G_t$ is an *edit-sequence of graphs* if for each $i > 0$ there exists some edge (u,v) s.t. either $G_i = G_{i-1} \cup \{(u,v)\}$ (update i is an *insertion*) or $G_i = G_{i-1} \setminus \{(u,v)\}$ (update i is a *deletion*). We typically assume $G_0 = \emptyset$. We say that $\mathcal{G}$ has bounded arboricity (by a number α), or that arboricity$(\mathcal{G}) \leq \alpha$, if arboricity$(G_i) \leq \alpha$ for every i.

An *orientation* of a graph G is a directed graph $\overline{G}$ with the same vertex and edge sets as G, but where an undirected edge $(u,v) \in G$ exists as the directed edge (u,v) or (v,u) in $\overline{G}$. We use $\deg(\overline{G})$ to denote the maximum out-degree of $\overline{G}$; it is a c-orientation if $\deg(\overline{G}) \leq c$. Any graph G with arboricity$(G) \leq \alpha$ has an α-orientation; to see this, partition the edges into α forests, pick an arbitrary root in every tree, and direct every edge towards the root of its respective tree.

We say that $\overline{\mathcal{G}} = \overline{G}_0, \overline{G}_1, \ldots, \overline{G}_t$ is a *sequence of orientations* of $\mathcal{G}$ if every $\overline{G}_i$ is an orientation of G_i. Similarly, $\overline{\mathcal{G}}$ is a c-orientation if every $\overline{G}_i$ is a c-orientation. A *flip* is a triple (i, v, u) such that (v,u) is an edge in $\overline{G}_{i-1}$ and (u,v) is an edge in $\overline{G}_i$.

An *offline c-orientation strategy* κ is some method that takes $\mathcal{G}$ and produces a c-orientation $\overline{\mathcal{G}}$. By abusing notation we will also use κ to refer to the $\overline{G}$ produced by κ.

An *online c-orientation algorithm* $\mathcal{A}$ is analogous to the offline strategy, except that it receives $\mathcal{G}$ as a stream and has only a single $\overline{G}_i$ stored in memory at any time. Hence, upon receiving *update* i, it produces $\overline{G}_i$ as a function of G_i and $\overline{G}_{i-1}$ and then forgets $\overline{G}_{i-1}$. We also say that $\mathcal{A}$ *maintains* an online c-orientation of $\mathcal{G}$.

We say that κ or $\mathcal{A}$ makes σ flips (in the worst case) if the number of flips between any two updates $i, i+1$ is at most σ, and that it makes σ *amortized* flips if after any update i the total amount of flips is at most σi.

Note the difference in wording: a *strategy* has access to the whole sequence $\mathcal{G}$ and produces the entire c-orientation at once, possibly using brute force. The online *algorithm* instead sees $\mathcal{G}$ as a stream of unknown length and, after every update i, produces only a single "current" orientation.

3 The algorithm

The algorithm takes an edit-sequence of graphs $\mathcal{G}$ as an online stream, and a positive integer parameter k. Each vertex v maintains a standard FIFO queue Q_v which holds all of its out-edges. On an insertion (deletion) update, orient the new edge arbitrarily (delete the edge via object reference) and then k times pick a vertex v with maximum out-degree and flip the first edge in Q_v. The book-keeping of out-degrees is trivial by using e.g. a degree-indexed array and a pointer to the maximum degree. We do not explicitly support queries. See pseudocode below for ease of reading.

Algorithm 1 Greedy flipping algorithm

```
procedure INSERTION($(v, u)$)
    push $(v, u)$ to $Q_v$
    K-FLIPS
procedure DELETION($(v, u)$)
    remove $(v, u)$ from $Q_v$
    K-FLIPS

procedure K-FLIPS
    for $i = 1$ to $k$ do
        let $v$ be a max out-degree vertex
        pop an edge $(v, u)$ from $Q_v$
        push $(u, v)$ to $Q_u$
```

4 Analysis

To show the efficiency of Algorithm 1, we will prove that its out-degree is competitive to an unknown offline strategy. For given $\mathcal{G}$ and k, let δ, σ and ε be values satisfying the following conditions: (i) there exists an offline δ-orientation strategy κ of $\mathcal{G}$ making at most σ flips in the worst case, (ii) $0 < \varepsilon \leq 1$, and (iii) $k \geq 1 + 1/\varepsilon + 2\sigma$.

▶ **Theorem 1.** *Algorithm 1 maintains an online $\mathcal{O}(\delta + (\delta\varepsilon + 1)\log_2 n)$-orientation of $\mathcal{G}$ with k flips and in $\mathcal{O}(k)$ time.*

Note that Algorithm 1 is completely oblivious to the values of δ, σ and ε, as well as any graph properties of $\mathcal{G}$ itself. The number of flips, and hence the running time, in Theorem 1 is trivial from the pseudocode. The rest of this section is dedicated to proving the bound on the out-degree. While the proof is quite non-trivial, the roadmap thereof is easy. We will associate potentials on all edges, such that the potential of an edge depends on where it is stored. Then we show that the total potential cannot increase, unless the maximum out-degree is $\mathcal{O}(\delta)$ in which case the potentials do not matter. Finally we re-interpret the moving of potentials as a game, where even an adversary cannot concentrate more than $\mathcal{O}((\delta\varepsilon + 1)\log n)$ extra potential in any single vertex – this also (roughly) bounds the maximum out-degree.

For purposes of analysis, we consider each queue Q_v to be two queues, the *Front* F_v and *Back* B_v. Edges are always inserted into B_v, and extracted from F_v. If F_v is empty when an edge should be extracted from Q_v, simply swap the two queues (by renaming) and then continue. It should be trivially clear that this is equivalent to using a single queue. We say an edge was flipped *from* v and *to* u if it was removed from Q_v/F_v and inserted into Q_u/B_u.

Table 2 Potential by type and placement in queue.

	Front	Back
Good	$1+2\varepsilon$	$1-\varepsilon$
Bad (first 3δ)	1	$1+\varepsilon$
Bad (rest)	1	1

To bound the maximum out-degree, we introduce potentials on the edges. At update i, we say that an edge in $\overline{G}_i$ is *good* if it has the same orientation as in $\kappa(G_i)$ and *bad* otherwise. Good edges have $1+2\varepsilon$ potential if they are in a Front queue and $1-\varepsilon$ in a Back queue. Bad edges have potential 1, except for the first 3δ bad edges in any Back queue which have potential $1+\varepsilon$. Let $p(v)$ be the sum of potentials of all edges stored in Q_v, $\hat{p}(\overline{G}) = \max_v p(v)$ and $P(\overline{G}) = \sum_v p(v)$. When we need to differentiate the potential of a vertex in a specific orientation $\overline{G}_i$, we use $p_i(v)$ to denote $p(v)$ at the time that the algorithm was storing $\overline{G}_i$.

Since Algorithm 1 does not know the values of δ or ε, it cannot determine the exact potential of a vertex. But as the following lemma shows, the out-degree of a vertex is a close approximation of its potential. We will prove the theorem by bounding the maximum potential of any vertex, which then implies a bound on its degree.

▶ **Lemma 2.** *For any vertex v, $\deg(v) + 5\delta\varepsilon \geq p(v) \geq \deg(v) - \delta\varepsilon$.*

Proof. For the upper bound, all edges contribute a base 1 potential, accounting for the $\deg(v)$ term. Note that at most δ out-edges of v are good. If they are all placed in F_v, they contribute an extra $2\delta\varepsilon$. At most 3δ bad edges in B_v contribute an extra ε each, giving at most $5\delta\varepsilon$ extra potential in total.

For the lower bound, only good edges in B_v can contribute less than 1 potential. Again there are at most δ of these and they contribute ε less, giving at least $\deg(v) - \delta\varepsilon$ potential in the vertex. ◀

Let $\beta = 6\delta\varepsilon$ be the *resolution* of the system. The following states that the potential of the highest-degree vertex is not too far from the maximum potential of any vertex.

▶ **Lemma 3.** *Let u be some vertex with maximum potential, and let v be some maximum out-degree vertex. Then $p(u) - p(v) \leq \beta$.*

Proof. By Lemma 2, the potential of v is at least $p(v) \geq \deg(v) - \delta\varepsilon$ and the potential of u is at most $p(u) \leq \deg(u) + 5\delta\varepsilon \leq \deg(v) + 5\delta\varepsilon$. Rearranging we get $p(u) - p(v) \leq \deg(v) + 5\delta\varepsilon - (\deg(v) - \delta\varepsilon) = 6\delta\varepsilon = \beta$. ◀

▶ **Lemma 4.** *Assume a vertex v has an empty F_v and at least 4δ edges in B_v. Then swapping F_v and B_v does not increase $p(v)$.*

Proof. The Back queue contains at most δ good edges and at least 3δ bad edges, hence exactly 3δ bad edges carry an extra ε potential which is released when moving from B_v to F_v. This $3\delta\varepsilon$ potential is enough to raise the potential of all δ good edges from $1-\varepsilon$ to $1+2\varepsilon$. Any surplus potential is lost. ◀

▶ **Lemma 5.** *Let v have out-degree at least 4δ. Then flipping an edge from v releases at least ε potential.*

Proof. By Lemma 4 we can assume F_v is non-empty. Let (u, v) be the edge moved from F_v to B_u. Note that if the edge was previously good it is now bad, and vice versa. Hence its potential decreases either from $1+2\varepsilon$ to at most $1+\varepsilon$, or from 1 to $1-\varepsilon$. ◀

▶ **Lemma 6.** *Let S be any suffix of the sequence of flips performed by Algorithm 1 after some update. Let $d = \deg(\overline{G})$ at the start of S. Then $\deg(\overline{G}) \leq d + 1$ after S.*

Proof. Note that flips can increase the maximum degree only if there are at least two vertices u, v with maximum degree, and the algorithm flips an edge incident on both of them. As soon as some vertex reaches degree $d + 1$, it will be the only vertex of maximum degree and immediately fall down to degree d in the following flip. Consequently no sequence of flips can raise a second vertex to degree $d + 1$, which is a necessary condition for raising any vertex to degree $d + 2$. ◀

▶ **Lemma 7.** *Let v be a vertex that had an edge flipped from it on update i. Then $\deg_{\overline{G}_i}(v) \geq \deg(\overline{G}_i) - 2$.*

Proof. Take the suffix S of flips that begins with the last flip from v. Before S, v had maximum out-degree d. After S, $d - 1 \leq \deg(v)$ and $\deg(\overline{G}_i) \leq d + 1$ by Lemma 6. ◀

Consider the algorithm as it receives an update i. We say that the currently stored graph $\overline{G}_{i-1}$ has *sufficient degree* if each of the k flips associated with update i is from a vertex with out-degree at least 4δ. Conversely, we say the graph $\overline{G}_{i-1}$ has *insufficient degree* if at least one of the k flips is from a vertex with out-degree less than 4δ.

▶ **Lemma 8.** *If $\overline{G}_{i-1}$ has insufficient degree, then $\deg(\overline{G}_i) = \mathcal{O}(\delta)$.*

Proof. Since some edge was flipped from a vertex with out-degree $d < 4\delta$, it follows from Lemma 6 that $\deg(\overline{G}_i) \leq d + 1 \leq 4\delta$. ◀

▶ **Lemma 9.** *If $\overline{G}_{i-1}$ has sufficient degree, then $P(\overline{G}_i) \leq P(\overline{G}_{i-1})$.*

Proof. Assume update i is an insertion. The new edge is inserted into a Back queue, and adds at most $1 + \varepsilon$ potential. The offline strategy κ makes at most σ flips, which causes σ stored edges to swap their classification ("renaming") from good to bad or vice versa. A Front edge that was bad increases potential from 1 to $1 + 2\varepsilon$, and a Back edge that was good increases from $1 - \varepsilon$ to 1 or $1 + \varepsilon$. The renaming can therefore increase the total potential by at most $2\sigma\varepsilon$. Each flip frees ε potential by Lemma 5 and the assumption of sufficient degree, so the total potential does not increase as long as $k\varepsilon \geq 1 + \varepsilon + 2\sigma\varepsilon$. This is guaranteed by the choice of parameters.

If the update was instead a deletion, the flips still release $k\varepsilon$ potential while even less potential is inserted. ◀

Note that the potential of the system can increase on both insertion and deletion updates if the graph has insufficient degree, since we cannot rely on Lemma 4 to ensure that the potential of a vertex is well-behaved when flipping edges from it. Also note that if κ is *known* not to perform any flips on deletion updates, *no* potential gets added to the system and so our algorithm can also forgo flipping on deletions.

So far we have shown that either the maximum out-degree is $\mathcal{O}(\delta)$, or we have a non-increasing quantity of potential and the degree of each vertex is closely approximated by its own potential. We next bound the maximum out-degree via a *counter game*, disassociated from the actual graph orientation, played by an adversary whose goal it is to concentrate as much potential as possible in a single counter. Counter games have been explored previously, under various names, in e.g. [4] and [9]: typically they may be thought of as two-player games where the second player is benevolent. Our game is different because the lone player is instead restricted by the concept of resolution β.

Formally, the game is played by a single player on n counters $x_1, \ldots, x_n$. Each counter x_i will hold a non-negative real-valued *weight* $|x_i|$, and the sum of weights is a constant $\sum_i |x_i| = X$. Any such distribution of X on the n counters is called a *game configuration* C. Let $\hat{x} = \max_i |x_i|$ be the maximum weight at any time. The player can perform arbitrarily many iterations of the following three-step operation: (i) pick a counter x_i and a $c > 0$ such that $|x_i| - c \geq \max(0, \hat{x} - \beta - 2)$, (ii) remove c weight from x_i and (iii) add positive weights whose sum is c to any set of counters.

The player is therefore allowed to redistribute weight *to* arbitrary counters, but must take it in not-too-large chunks *from* counters that are within the resolution (here $\beta + 2$) of the maximum counter. Before upper-bounding $\hat{x}$, we show that the player is powerful enough to simulate the movement of potentials by Algorithm 1. We say a game configuration C *dominates* a graph orientation $\overline{G}$ if $|x_j| \geq p(v_j)$ for every j.

▶ **Lemma 10.** *Let i be an update such that $\overline{G}_{i-1}$ has sufficient degree. Let C be a game configuration that dominates $\overline{G}_{i-1}$. Then the player can reach a game configuration C' that dominates $\overline{G}_i$.*

Proof. We need to show that if some vertex gains potential (so its corresponding counter no longer dominates it), then we can safely take enough weight from other counters to fill that 'gap'. Keep in mind that the total potential does not increase (Lemma 9). Since the player is allowed to redistribute weight *to* any counter, we let the gaps be filled in arbitrary order and only show that enough weight can be taken *from* other counters to make up the difference. If $\hat{x} > \hat{p}(\overline{G}_{i-1})$ then greedily take weight from all counters greater than $\hat{p}(\overline{G}_{i-1})$ to get $\hat{x} = \hat{p}(\overline{G}_{i-1})$.

Let v_j be a vertex that had an edge flipped from it. Then its resulting out-degree is $\deg_{\overline{G}_i}(v_j) \geq \deg(\overline{G}_i) - 2$ (Lemma 7) and its potential is $p_i(v_j) \geq \deg_{\overline{G}_i}(v_j) - \delta\varepsilon \geq \deg(\overline{G}_i) - 2 - \delta\varepsilon$ (Lemma 2). Also by Lemma 2 the maximum potential in the system is $\hat{p}(\overline{G}_i) \leq \deg(\overline{G}_i) + 5\delta\varepsilon$. Hence the final potential of v_j is within $6\delta\varepsilon + 2 = \beta + 2$ of the maximum potential. As the rules of the counter game allow us to take weight up to $\beta + 2$ from the maximum counter, then however much potential v_j lost we can take at least the same amount of weight from its corresponding counter x_j.

Conversely, if a vertex loses potential but its resulting potential is not at least $\hat{p}(\overline{G}_i) - \beta - 2$, it must have lost that potential due to deletion or renaming rather than flipping. Its counter can safely be left untouched and still dominate the potential of the vertex.

Since the sum of potential decreases (by flipping) is at least as large as the sum of increases (for any reason) (Lemma 9), and for any vertex that lost potential by flipping we can remove at least as much weight from its counter, then we can redistribute enough weight to raise the too-low counters to again dominate their respective vertex potentials. The updated counters form a game configuration that dominates $\overline{G}_i$. ◀

▶ **Lemma 11.** *Let $\overline{G}_a, \ldots, \overline{G}_b$ be any sequence of orientations such that $\overline{G}_i$ has sufficient degree for every $a \leq i \leq b$. Consider a game with starting configuration C_a that dominates $\overline{G}_a$, with $\hat{x} = \hat{p}(\overline{G}_a)$. Then the player can reach game configurations $C_a, \ldots, C_b$ where C_i dominates $\overline{G}_i$ for every $a \leq i \leq b$.*

Proof. For every $a < i \leq b$ iterate Lemma 10 on C_{i-1} to create C_i. ◀

We now let an adversary play the game, with the goal to increase $\hat{x}$ as much as possible. For simplicity we assume that every counter is raised to $\hat{x}$ as the starting configuration. For $j = -1, 0, 1, 2, \ldots$ let $\ell_j = X/n + j(\beta + 2)$ be *weight level* j. A counter x_i is *above* *level* j, or *above* ℓ_j, if $|x_i| \geq \ell_j$. Let $X_j = \sum_{i=1}^{n} \max(0, |x_i| - \ell_j)$ be the *weight above* ℓ_j, and $\overline{X}_j = X - X_j$ the *weight below* ℓ_j. We say a counter x_i *contributes* $\max(0, |x_i| - \ell_j)$ to X_j and $\min(|x_i|, \ell_j)$ to $\overline{X}_j$.

▶ **Lemma 12.** *Let j be a weight level such that $\ell_j \leq \hat{x}$. Let the player make any sequence of moves that maintain the condition $\ell_j \leq \hat{x}$. Then X_{j-1} does not increase.*

Proof. Note that any counter x_i contributes $\min(|x_i|, \ell_{j-1})$ to $\overline{X}_{j-1}$. By assumption there will always be a counter x_k with $\ell_j \leq |x_k|$. Hence the resolution rule prevents the player from making any counter contribute less to $\overline{X}_{j-1}$ than it already does. Since X is a constant and $\overline{X}_{j-1}$ is non-decreasing, $X_{j-1} = X - \overline{X}_{j-1}$ is non-increasing. ◀

Since $\ell_0 = X/n$ is the average weight of all counters, it must always be the case that $\hat{x} \geq \ell_0$ and $X_{-1} \leq n(\beta + 2)$.

▶ **Lemma 13.** *Let j be a weight level such that $\ell_j \leq \hat{x} \leq \ell_{j+1}$. Let the player make any sequence of moves that maintain the condition $\ell_j \leq \hat{x} \leq \ell_{j+1}$. Then $2X_j \leq X_{j-1}$.*

Proof. By Lemma 12, X_{j-1} is a non-increasing amount. Let x_i be any counter that will contribute some positive weight w to X_j. Since the player maintains that $\hat{x} \leq \ell_{j+1}$, no counter will be able to contribute more than $\ell_{j+1} - \ell_j = \beta + 2$ to X_j, i.e. $0 < w \leq \beta + 2$. Then x_i must contribute $w + \beta + 2$ to X_{j-1}. Hence any counter that contributes to X_j contributes at least twice as much to X_{j-1}, and $2X_j \leq X_{j-1}$. ◀

The player is therefore stuck in the following dilemma: once $\hat{x}$ reaches some level ℓ_j, only a bounded amount X_{j-1} of weight remains available to redistribute. But once $\hat{x}$ reaches ℓ_{j+1}, only the weight above ℓ_j will be possible to redistribute. Therefore, in order to concentrate as much weight as possible above ℓ_{j+2}, the player must first maximize X_j without any counter actually reaching above ℓ_{j+1}.

▶ **Lemma 14.** *The player cannot increase $\hat{x}$ to $\ell_{1+\log_2 n}$.*

Proof. Assume $\hat{x} \geq \ell_{\log_2 n}$. By alternatingly iterating Lemma 12 and Lemma 13, the weight above $\ell_{\log_2 n}$ is $X_{\log_2 n} \leq \left(\frac{1}{2}\right)^{1+\log_2 n} X_{-1} = \frac{1}{2n} X_{-1} \leq \frac{1}{2n} n(\beta + 2) < \beta + 2$. Since the weight is strictly less than $\beta + 2$, even concentrating all of it in a single counter is not enough to make that counter reach $\ell_{1+\log_2 n}$. Hence $\hat{x} < \ell_{1+\log_2 n}$. ◀

We are now ready to prove the out-degree part of Theorem 1.

Proof of Theorem 1. Either the graph orientation has insufficient degree and maximum out-degree $\mathcal{O}(\delta)$ (Lemma 8) or it has non-increasing potential (Lemma 9) which is dominated by a counter game where the starting weight of any counter is $\mathcal{O}(\delta)$ (Lemma 11). By Lemma 14, the maximum counter is $\hat{x} < \ell_{1+\log_2 n} = \mathcal{O}(\delta) + (1 + \log_2 n)(\beta + 2)$. By Lemma 2, $\deg(v) \leq p(v) + \delta\varepsilon$, and therefore any vertex has out-degree bounded by $\mathcal{O}(\delta) + (1 + \log_2 n)(\beta + 2) + \delta\varepsilon = \mathcal{O}(\delta + (\delta\varepsilon + 1)\log_2 n)$. ◀

5 De-amortizing offline strategies

In the previous work by Brodal and Fagerberg [3], their amortized algorithm is shown competitive with an offline strategy with bounded amortized number of flips, and hence subsequently published strategies have focused on achieving good amortized bounds. However, for our algorithm analysis, we require an offline strategy with *worst-case* flips per update.

In this section we show one way to de-amortize offline strategies. Our technique does not generalize to every offline strategy, but relies on the special structure inherent to the strategies of both [7] and [5]. These strategies partition the edit-sequence into blocks of consecutive updates, with some length λ. No flips occur within a block, only in the seams between two blocks. The amortized flip complexity of these strategies is therefore simply the maximum number of flips between two blocks, divided by the length λ of the preceding block.

Since no flips are allowed within a block, the strategy is required to find an orientation of the union of all graphs $G_i, \ldots, G_{i+\lambda-1}$ within a block. The maximum out-degree of the entire strategy is therefore upper bounded by the maximum out-degree of any oriented union-graph. Higher λ gives a less sparse union-graph, necessitating higher out-degree, but also allows for a better amortized flip complexity. The following theorem shows a simple way of de-amortizing strategies with this structure, by taking all the flips between two blocks and spreading them evenly over the updates in the later block.

▶ **Theorem 15.** *Let κ be a δ-orientation strategy of $\mathcal{G}$ where, for arbitrary λ, any update with $\sigma\lambda$ flips is followed by at least $\lambda - 1$ updates with no flips. Then there exists a 2δ-orientation strategy of $\mathcal{G}$ making σ flips in the worst case.*

Note that if the last block of flips is not followed by $\lambda - 1$ updates due to $\mathcal{G}$ ending, then one can pad $\mathcal{G}$ to appropriate length by repeatedly inserting and removing a dummy edge after the end of $\mathcal{G}$. Also note that λ can vary within the same sequence – blocks do not need to be of uniform length.

Proof. Let i be an update where κ performs a set of $\lambda\sigma$ flips. Let F be the set of flipped edges. Let κ' be an offline strategy with the same edge orientations as κ except on updates $i, \ldots, i + \lambda - 1$. On any insertion update $i, \ldots, i + \lambda - 1$, let κ' orient the new edge in the same direction as κ. Furthermore, on each update $i, \ldots, i + \lambda - 1$, κ' takes σ arbitrary edges in F, removes them from F, and flips them.

Then F will be empty after update $i + \lambda - 1$, so $\kappa(G_{i+\lambda-1}) = \kappa'(G_{i+\lambda-1})$. At all times F forms a δ-orientation, since F is a subset of $\kappa(G_{i-1})$. Similarly, $\kappa'(G_j) \setminus F$ is δ-orientation for every $i \leq j \leq i + \lambda - 1$, since they are a subset of $\kappa(G_j)$. Hence κ' is a 2δ-orientation. Finally, κ' performs at most σ flips per update between updates i and $i + \lambda - 1$; exhaustively perform the same transformation on all of κ for σ flips on any update. ◀

6 Discussion

With our two theorems proven, we can relate the algorithm to known offline strategies and achieve the following corollaries. In all of the following, $\mathcal{G}$ is an arbitrary edit-sequence with arboricity$(\mathcal{G}) \leq \alpha$.

Kowalik [8] presents an offline $\mathcal{O}(\alpha \log n)$-orientation strategy making 1 amortized flip. Using Theorem 15 we can de-amortize it to an offline $\mathcal{O}(\alpha \log n)$-orientation strategy making 1 flip in the worst case, giving the following two corollaries.

▶ **Corollary 16.** *For a positive function $f(n) = \mathcal{O}(\log n)$, Algorithm 1 maintains an $\mathcal{O}\left(\frac{\alpha \log^2 n}{f(n)}\right)$-orientation with $k = 3 + \lceil f(n) \rceil$ flips.*

Proof. Let $\delta = \mathcal{O}(\alpha \log n)$, $\sigma = 1$ and $\varepsilon = 1/f(n)$. Then the algorithm maintains out-degree $\mathcal{O}\left(\alpha \log n + \left(\frac{\alpha \log n}{f(n)} + 1\right) \log n\right) = \mathcal{O}\left(\frac{\alpha \log^2 n}{f(n)}\right)$. ◀

▶ **Corollary 17.** *Algorithm 1 maintains an $\mathcal{O}\left(\alpha \log^2 n\right)$-orientation of $\mathcal{G}$ with $k = 4$ flips.*

Proof. Let $f(n) \equiv 1$ in Corollary 16. ◀

Corollary 17 is the first result with $\mathcal{O}(1)$ worst-case flips. Compared to [8] (with $\mathcal{O}(1)$ *amortized* flips), it incurs an extra $\mathcal{O}(\log n)$ factor on the out-degree, but avoids the $\Omega(n)$ worst-case flips which that algorithm can experience.

Brodal and Fagerberg [3] give an offline $\mathcal{O}(\alpha)$-orientation strategy with $\mathcal{O}(\log n)$ flips in the worst case. It only makes flips on insertion updates.

▶ **Corollary 18.** *Algorithm 1 maintains an $\mathcal{O}(\alpha + \log n)$-orientation of $\mathcal{G}$ with $k = \mathcal{O}(\log n)$ flips.*

Proof. Let $\delta = \mathcal{O}(\alpha)$, $\sigma = \mathcal{O}(\log n)$ and $\varepsilon = 1/\log n$. Then the algorithm maintains out-degree $\mathcal{O}\Big(\alpha + \Big(\frac{\alpha}{\log n} + 1\Big)\log n\Big) = \mathcal{O}(\alpha + \log n)$. ◀

He et al. [5] give an offline $\mathcal{O}\big(\alpha\sqrt{\log n}\big)$-orientation strategy making $\mathcal{O}\big(\sqrt{\log n}\big)$ amortized flips, which we de-amortize using Theorem 15.

▶ **Corollary 19.** *Algorithm 1 maintains an $\mathcal{O}(\alpha \log n)$-orientation of $\mathcal{G}$ with $k = \Theta(\sqrt{\log n})$ flips.*

Proof. Let $\delta = \mathcal{O}\big(\alpha\sqrt{\log n}\big)$, $\sigma = \mathcal{O}\big(\sqrt{\log n}\big)$ and $\varepsilon = 1/\sqrt{\log n}$. Then the algorithm maintains out-degree $\mathcal{O}\left(\alpha\sqrt{\log n} + \left(\frac{\alpha\sqrt{\log n}}{\sqrt{\log n}} + 1\right)\log n\right) = \mathcal{O}(\alpha \log n)$. ◀

▶ **Corollary 20.** *Algorithm 1 maintains an $\mathcal{O}(\log n)$-orientation of $\mathcal{G}$ with $k = \mathcal{O}\big(\alpha\sqrt{\log n}\big)$ flips, if $\alpha = \mathcal{O}\big(\sqrt{\log n}\big)$.*

Proof. Let $\delta = \mathcal{O}\big(\alpha\sqrt{\log n}\big)$, $\sigma = \mathcal{O}\big(\sqrt{\log n}\big)$ and $\varepsilon = 1/\alpha\sqrt{\log n}$. Then the algorithm maintains out-degree $\mathcal{O}\left(\alpha\sqrt{\log n} + \left(\frac{\alpha\sqrt{\log n}}{\alpha\sqrt{\log n}} + 1\right)\log n\right) = \mathcal{O}(\log n)$. ◀

Corollary 19 is an improvement over [7] in the flip complexity for edit-sequences with arboricity bounded by a constant. For $\alpha = \mathcal{O}\big(\sqrt{\log n}/\log\log n\big)$, Corollary 20 matches or improves the flip complexity from [7], albeit with a slightly worse degree bound, and only if α is known. If α is both $\mathcal{O}\big(\sqrt{\log n}\big)$ and $\omega(\sqrt{\log(n)}/\log\log n)$ we are narrowly outperformed by [7], by no more than an $\mathcal{O}(\log\log n)$ factor.

Corollary 19 also nearly matches the degree bound in [5] but with worst-case flips instead of amortized. Corollary 18 matches the bounds in [7] for general arboricity and improves on their flip complexity if $\alpha = \omega(\log n)$. Furthermore, if $\alpha = \Omega(\log n)$, Corollary 18 matches the amortized bounds from [3].

6.1 Reverse trade-off

Compared to an offline strategy, our analysis lends itself to a trade-off in one direction, getting (at most) an $\mathcal{O}(\log n)$ factor on the out-degree for a constant factor on the number of flips. It allows us to perform much fewer flips than in [7] at the price of weaker degree bounds. A trade-off in the opposite direction would also be highly desirable, achieving out-degree (closer to) $\mathcal{O}(\delta)$ by making $\Omega(\sigma)$ flips. We have only found a very weak such trade-off:

▶ **Lemma 21.** *Algorithm 1 can maintain an $\mathcal{O}(\alpha)$-orientation of $\mathcal{G}$ with $k = \mathcal{O}(\alpha n)$ flips.*

Proof. Let $\delta = \mathcal{O}(\alpha)$ and $\varepsilon = 1$ (the value of σ is inconsequential). Then each edge holds between 0 and 3 potential. And since any G_i has at most αn edges (by definition of arboricity), the total potential is between 0 and $3\alpha n$. Furthermore each flip releases 1 potential from the system, contingent on the graph having sufficient degree (Lemma 5). Hence after performing at most $3\alpha n$ flips on any starting orientation $\overline{G}_i$, we must reach a state where the next flip does not release potential, contradicting Lemma 5, and so by Lemma 8 the graph has out-degree at most $4\delta = \mathcal{O}(\alpha)$ after all flips. ◀

Lemma 21 only matches the worst-case bound of the algorithm in [3], which has drastically better amortized performance. Hence it should not be used in practice. Still, we believe a stronger reverse trade-off is possible and conjecture the following:

▶ **Conjecture 22.** *For some function f, Algorithm 1 maintains an online $\mathcal{O}\left(\delta + \frac{\sigma+1}{f(k)}\delta \log n\right)$-orientation of $\mathcal{G}$ with k flips and in $\mathcal{O}(k)$ time.*

6.2 Dynamic arboricity

Throughout the paper we have done all our performance analysis against a static arboricity bound, i.e. a bound on the greatest arboricity seen anywhere in the edit-sequence. An interesting issue arises if the sequence contains contiguous sub-sequences, of non-trivial length, with higher or lower arboricity than elsewhere in the sequence. Some previous algorithms, e.g. one of the algorithms in [7] and the non-amortized algorithm in [5], adapt to increasing and decreasing arboricity automatically.

Our analysis immediately adapts to sequences with increasing arboricity, since the analysis can be performed on any prefix (or contiguous sub-sequence) of $\mathcal{G}$. In the case of periods with lower arboricity than earlier in the sequence, our algorithm obeys the new arboricity if the maximum out-degree is already within that new bound. In other words, if the maximum out-degree is already bounded relative to the new arboricity, then it will remain so. However, if the arboricity falls enough that the current maximum out-degree *breaks* the new bounds, our analysis does not require the maximum out-degree to decrease accordingly. Intuitively, using a k strictly larger than $1 + 1/\varepsilon + 2\sigma$ (thus experiencing a net loss of total potential with every update) should force the maximum out-degree to tend towards the updated degree bounds, similar to the proof of Lemma 21. However, we do not have a formal argument for this.

6.3 Open problems

For all known strategies that maintain out-degree δ with σ (amortized) flips, it holds that $\delta\sigma = \Omega(\alpha \log n)$ and most achieve $\delta\sigma = \Theta(\alpha \log n)$. Can one design a strategy with $\delta\sigma = o(\alpha \log n)$?

References

1 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. *arXiv preprint arXiv:1506.07076*, 2015.

2 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 692–711. Society for Industrial and Applied Mathematics, 2016.

3 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In *Proceedings 6th International Workshop on Algorithms and Data Structures (WADS)*, volume 1663 of *Lecture Notes in Computer Science*, pages 342–351. Springer, 1999.

4 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372. ACM, 1987.

5 Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Proceedings 25th International Symposium on Algorithms and Computation (ISAAC)*, volume 8889 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2014.

6 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

7 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Proceedings 41st International Colloquium Automata, Languages, and Programming (ICALP), Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014.

8 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Information Processing Letters*, 102(5):191–195, 2007.

9 Christos Levcopoulos and Mark H. Overmars. A balanced search tree with $O(1)$ worst-case update time. *Acta Informatica*, 26(3):269–277, 1988.

10 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

11 David Peleg and Shay Solomon. Dynamic (1+ ε)-approximate matchings: a density-sensitive approach. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 712–729. Society for Industrial and Applied Mathematics, 2016.

12 Shay Solomon. Fully dynamic maximal matching in constant update time. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 325–334. IEEE, 2016.

Crossing Number for Graphs with Bounded Pathwidth[*][†]

Therese Biedl[‡][1], Markus Chimani[§][2], Martin Derka[¶][3], and Petra Mutzel[‖][4]

1 **Dept. of Computer Science, University of Waterloo, Canada**
biedl@uwaterloo.ca
2 **Dept. of Computer Science, Universität Osnabrück, Germany**
markus.chimani@uni-osnabrueck.de
3 **Dept. of Computer Science, University of Waterloo, Canada**
mderka@uwaterloo.ca
4 **Dept. of Computer Science, Technische Universität Dortmund, Germany**
petra.mutzel@cs.tu-dortmund.de

Abstract

The crossing number is the smallest number of pairwise edge crossings when drawing a graph into the plane. There are only very few graph classes for which the exact crossing number is known or for which there at least exist constant approximation ratios. Furthermore, up to now, general crossing number computations have never been successfully tackled using bounded width of graph decompositions, like treewidth or pathwidth.

In this paper, we for the first time show that crossing number is tractable (even in linear time) for maximal graphs of bounded pathwidth 3. The technique also shows that the crossing number and the rectilinear (a.k.a. straight-line) crossing number are identical for this graph class, and that we require only an $O(n) \times O(n)$-grid to achieve such a drawing.

Our techniques can further be extended to devise a 2-approximation for general graphs with pathwidth 3, and a $4\mathbf{w}^3$-approximation for maximal graphs of pathwidth $\mathbf{w}$. This is a constant approximation for bounded pathwidth graphs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Crossing Number, Graphs with Bounded Pathwidth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.13

1 Introduction

The crossing number $cr(G)$ is the smallest number of pairwise edge-crossings over all possible drawings of a graph G into the plane. Despite decades of lively research, see e.g. [25, 26], even most seemingly simple questions, such as the crossing number of complete or complete bipartite graphs, are still open, cf. [23]. There are only very few graph classes, e.g., Petersen graphs $P(3, n)$ or Cartesian products of small graphs with paths or trees, see [4, 20, 24], for which the crossing number is known or can be efficiently computed.

[*] This work has been started at the Crossing Number Workshop 2016 in Strobl (Austria)
[†] A full version of the paper is available at [1], https://arxiv.org/abs/1612.03854.
[‡] Research of T.B. supported by NSERC.
[§] Research of M.C. partially supported by the German Science Foundation (DFG), project CH 897/2-1.
[¶] Research of M.D. supported by NSERC Vanier CSG.
[‖] Research of P.M. partially supported by the DFG within the SFB 876 (project A6).

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 13; pp. 13:1–13:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Considering approximations, we know that computing $cr(G)$ is APX-hard [5], i.e., there does not exist a PTAS (unless P = NP). The best known approximation ratio for general graphs with bounded maximum degree is $\tilde{O}(n^{0.9})$ [10]. We only know constant approximation ratios for special graph classes. In fact, all known constant approximation ratios are based on one of three concepts: *Topology-based* approximations require that G can be embedded without crossings on a surface of some fixed or bounded genus [13, 16, 17]. *Insertion-based* approximations assume that there is only a small (i.e., bounded size) subset of graph elements whose removal leaves a planar graph [6–9]. In either case, the ratios are constant only if we further assume bounded maximum degree. Finally, some approximations for the crossing number exist if the graph is *dense* [12].

While treewidth and pathwidth have been very successful tools in many graph algorithm scenarios, they have only very rarely been applied to crossing number: Since general crossing number seems not to be describable with second order monadic logic, Courcelle's result [11] regarding treewidth-based tractability can only be applied if cr itself is bounded [14, 18]. The related strategy of "planar decompositions" lead to linear crossing number bounds [27].

Contribution. In this paper, we for the first time show that such graph decompositions, in our case pathwidth, *can* be used for computing crossing number. We show for maximal graphs G of pathwidth 3 (see Section 3):

- We can compute the *exact* crossing number $cr(G)$ in linear time.
- The topological $cr(G)$ equals the *rectilinear* crossing number $\overline{cr}(G)$, i.e., the crossing number under the restriction that all edges need to be drawn as straight lines.
- We can compute a drawing realizing $\overline{cr}(G)$ on an $O(n) \times O(n)$-grid.

We then generalize these techniques to show:

- A 2-approximation for $cr(G)$ and $\overline{cr}(G)$ for general graphs of pathwidth 3, see Section 4.
- A $4\mathbf{w}^3$-approximation for $cr(G)$ for maximal graphs of pathwidth $\mathbf{w}$, see Section 5. This can be achieved by placing vertices and bend points on a $4n \times \mathbf{w}n$ grid.

Observe that in contrast to most previous results, these approximation ratios are *not* dependent on the graph's maximum degree. As a complementary side note, we show (in the full version of the paper, see [1]) that the *weighted* (possibly rectilinear) crossing number is weakly NP-hard already for maximal graphs with pathwidth 4.

Focusing on graphs with bounded pathwidth may seem very restrictive, but in some sense these are the most interesting graphs for crossing minimization because Hliněný showed that crossing-number critical graphs have bounded pathwidth [15].

2 Preliminaries

We always consider a simple undirected graph G with n vertices as our input. A drawing of G is a mapping φ of vertices and edges to points and simple curves in the plane, respectively. The curve $\varphi(e)$ of an edge $e = (u, v)$ does not pass through any point $\varphi(w)$, $w \in V(G)$, but has its ends at $\varphi(u)$ and $\varphi(v)$. When asking for a crossing minimum drawing of G, we can restrict ourselves to *good* drawings, which means that adjacent edges do not cross, non-adjacent edges cross at most once, and no three edges cross at the same point of the drawing. For other drawings, straightforward redrawing arguments, see e.g. [25], show that the crossing number can never increase when establishing these properties.

A *clique* is a complete graph and a *biclique* is a complete bipartite graph. While the exact crossing number is unknown for general cliques and bicliques, there are upper bound constructions, conjectured to attain the optimal value. In particular the old construction due to Zarankiewicz, attaining $\lfloor\frac{n_1}{2}\rfloor\lfloor\frac{n_1-1}{2}\rfloor\lfloor\frac{n_2}{2}\rfloor\lfloor\frac{n_2-1}{2}\rfloor$ crossings for K_{n_1,n_2}, is known to give the optimum for $n_1 \leq 6$ [19].

A prominent variant of the traditional ("topological") crossing number $cr(G)$ is the *rectilinear* crossing number $\overline{cr}(G) \geq cr(G)$, sometimes also known as geometric or straight-line crossing number. Thereby, edges are required to be drawn as straight line segments without any bends. Interestingly, while we know $\overline{cr}(G) > cr(G)$ in general (e.g., already for complete graphs), Zarankiewicz's construction is a straight-line drawing, suggesting that maybe $cr(G) = \overline{cr}(G)$ for bicliques.

Alternating path decompositions and clusters. There are several equivalent definitions of pathwidth; we use here the one based on tree decompositions, see e.g. [21]. A *path decomposition* $\mathcal{P}$ of a connected graph G consists of a finite set of *bags* $\{X_i \mid 1 \leq i \leq \xi \in \mathbb{N}\}$, where each bag is a subset of the vertices of G, such that for every edge (v, w) at least one bag contains both v and w, and for every vertex v of G the set of bags containing v forms an interval (i.e., the underlying graph formed by the bags is a path). The indexing of the bags gives a total ordering and we may speak of *first*, *last*, *preceding*, and *succeeding* bags. The *width* of a path decomposition is the maximum cardinality of a bag minus one, i.e., $\max_{1\leq i\leq\xi} |X_i| - 1$. The *pathwidth* $\mathbf{w} := \mathbf{w}(G)$ of G is the smallest width that can be achieved by a path decomposition of G. A *maximal pathwidth-*$\mathbf{w}$ *graph* is a graph of pathwidth $\mathbf{w}$ for which adding any edge increases its pathwidth. In particular, this implies that the vertices in each bag form a clique. We assume that $n > \mathbf{w} + 1$; otherwise G is a clique and the crossing number is 0 for $\mathbf{w} = 3$ and easily approximated within a factor of $O(1)$ for bigger $\mathbf{w}$ (e.g., via the crossing lemma [22]).

Several additional constraints can be imposed on the bags and the path decomposition without affecting the required width. We use a variant of a *nice* path decomposition that we call an *alternating* path decomposition (see Fig. 1); one can easily show that such a decomposition exists:

- There are exactly $\xi = 2n - 2\mathbf{w} - 1$ bags.
- $|X_i| = \mathbf{w} + 1$ if i is odd and $|X_i| = \mathbf{w}$ if i is even.
- For any even $1 < i < \xi$, we have $X_{i-1} \supset X_i \subset X_{i+1}$.

Note that for any odd i there is exactly one vertex v that is in X_i but not in bag X_{i+1}. We say that v is *forgotten* by bag X_{i+1}. Similarly, bag X_i contains exactly one vertex v that was not in bag X_{i-1}. We say that v is *introduced* by bag X_i. We define the *age-order* $\{v_1, \ldots, v_n\}$ of the vertices of G as follows: v_1 is forgotten by X_2; $v_2, \ldots, v_{\mathbf{w}+1}$ are the other vertices of bag X_1 in arbitrary order. The order of the remaining vertices corresponds to the order of the bags by which they are introduced. We say that v_i is *older* than v_j if $i < j$, so the three oldest vertices are v_1, v_2, v_3. Note that we can choose v_2, v_3 arbitrarily among $X_1 - \{v_1\}$. In particular, if two vertices $p, q \in X_1$ are specified, then we can ensure that they are among the three oldest; this will be exploited in Section 4.2.

In our algorithms and proofs, we will work with special subsets of bags called *clusters*. Let G be a connected graph of pathwidth 3 with an alternating path decomposition $\mathcal{P} = \{X_i\}_{1\leq i\leq\xi}$. Consider a set of three vertices Y that constitute at least one bag (this bag has an even index). There can be several such bags with exactly those vertices, but all bags containing Y are consecutive. For any such Y, we define a *cluster* C as the maximal consecutive set of bags that all contain Y. We say that $T(C) := Y$ is the *anchor-triplet* of C. Any cluster has at least 3 bags. They alternate between size 4 and 3, starting and ending with size-4 bags. Two consecutive clusters overlap in exactly one bag (which consequently has size 4). The order of the bags induces a unique order of the clusters $\{C_1, \ldots, C_\kappa\} =: \mathcal{C}$.

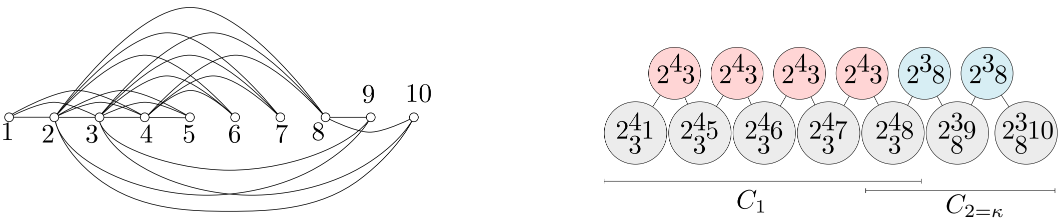


Figure 1 (left) A graph, with vertices in age order according to $\mathcal{P}$. (right) Its alternating path decomposition $\mathcal{P}$ of width 3, with two clusters: C_1 has $T(C_1) = \{2, 3, 4\}$, and consists of all bags containing this anchor-triplet. Analogously, we have $T(C_2) = \{2, 3, 8\}$. In C_1, the lost vertex is $x_1^- = 1$ and the emerging vertex is $x_1^+ = 8$.

Note that a cluster C can be described as a set of bags, or by its anchor-triplet. Denote the vertices that appear in the union of bags of C by $V(C)$, and let $n(C) := |V(C)|$. The following observation is trivial (because any vertex of the anchor-triplet of C belongs to all bags of C) but crucial for our analysis.

▶ **Observation 1.** *Let G be a maximal pathwidth-3 graph and let C be a cluster. Then the graph induced by $V(C)$ consists of the triangle induced by $T(C)$ and (edge-disjoint) a biclique $K_{3,n(C)-3}$ with one partition being $T(C)$.*

We define the *emerging vertex* of C_i, denoted by x_i^+, as the vertex introduced by the last bag of C_i. Note that x_i^+ belongs to the anchor-triplet of the next cluster C_{i+1} if $i < \kappa$. We define the *lost vertex* of C_i, denoted by x_i^-, as the vertex that was forgotten by the second bag of C_i. Note that x_i^- belongs to the anchor-triplet of the previous cluster C_{i-1} if $i > 1$, but not to the anchor-triplet of C_i. Observe that $x_1^- = v_1$, $x_\kappa^+ = v_n$, $x_{i-1}^+ \neq x_i^-$ and $T(C_i) = T(C_{i-1}) \cup \{x_{i-1}^+\} \setminus \{x_i^-\}$ for all $2 \leq i \leq \kappa$. For notational simplicity, we define $x_0^+ := v_2$. Any vertex x that belongs to C_i but is not in $T(C_i) \cup \{x_i^+, x_i^-\}$ is called a *singleton* of C_i. Vertex x belongs to a "middle" bag of C_i and only appears in this bag; it belongs to no cluster other than C_i. See Fig. 1 for an example.

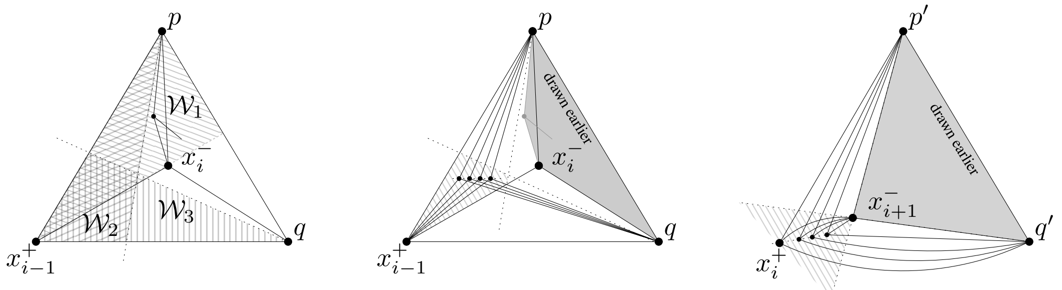
3 Exact Algorithm for Maximal Pathwidth-3 Graphs

Let G be a maximal pathwidth-3 graph and fix an alternating path decomposition of width 3. By maximality, all bags form cliques, and in particular, each anchor-triplet induces a triangle in the graph, called *anchor triangle* consisting of *anchor edges*.

The general idea to draw G is to iterate through the clusters $C_1, \ldots, C_\kappa$. When considering cluster C_i, its first bag will already be drawn and the anchor triangle will form the outer face of the current drawing. About half of the vertices introduced by C_i will be drawn inside the anchor triangle while the other half will be drawn outside, mimicking Zarankiewicz' construction locally. The number of crossings that these vertices add will be exactly the minimum number of crossings needed to draw the biclique $K_{3,n(C_i)-3}$ of cluster C_i, hence leading to an optimal drawing.

We start with drawing bag $X_1 = \{v_1, v_2, v_3, v_4\}$ as a planar drawing of K_4 with the vertices $T(C_1) = X_2 = \{v_2, v_3, v_4\}$ on the outer face. Now we iterate over all clusters C_i, $1 \leq i \leq \kappa$, drawing their bags with the following invariants:

- The drawing is good and straight-line.
- Before drawing C_i, the outer face contains the three vertices $T(C_i)$.
- For any $j \leq i$, the anchor edges of C_j are drawn without crossings.



(a) Wedges $\mathcal{W}_1$, $\mathcal{W}_2$, and $\mathcal{W}_3$. **(b)** Adding ℓ_1 vertices inside. **(c)** Adding ℓ_2 vertices outside.

Figure 2 Drawing maximal pathwidth-3 graphs. For ease of legibility we draw some edges in (c) slightly curved. Dotted lines mark boundaries of the regions defined in the text.

Let ℓ be the number of singleton vertices in C_i (possibly $\ell = 0$). We need to place the ℓ singletons and the emerging vertex x_i^+. We will add $\ell_1 := \lfloor(\ell+1)/2\rfloor \leq \ell$ vertices into an inner face of the current drawing and $\ell_2 = \lceil(\ell+1)/2\rceil \geq 1$ vertices on the outside. Note that $\ell_1 + \ell_2 = \ell + 1$.

Placement on the inside. By the invariant the outer face consists of the edges connecting $T(C_i) = \{x_{i-1}^+, p, q\}$ for some p, q. W.l.o.g. assume that x_{i-1}^+, p, and q occur in clockwise order walking along the outer face. By maximality, and because x_{i-1}^+ has just been introduced, x_{i-1}^+ has degree 3 in the current graph, and its neighbors are p, q, x_i^-.

Let $\mathcal{R}$ be the open region obtained by the intersection of three open "wedges" $\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3$ defined as follows: Wedge $\mathcal{W}_1$ emanates from x_{i-1}^+ between edges (x_{i-1}^+, p) and (x_{i-1}^+, x_i^-) in the interior of the triangle induced by $T(C_i)$. Wedge $\mathcal{W}_2$ ($\mathcal{W}_3$) emanates at p (q) inside of $T(C_i)$ and runs along edge (p, x_{i-1}^+) ((q, x_{i-1}^+), respectively) with a sufficiently small angle such that it crosses only edges incident to x_{i-1}^+. Any point inside $\mathcal{R}$ can be connected to all of p, q, x_{i-1}^+ with straight lines and a single crossing (with edge (x_{i-1}^+, x_i^-)).

Consider a straight line s through $\mathcal{R}$ but not through any of p, q, x_{i-1}^+. Place ℓ_1 vertices (for ℓ_1 singletons of C_i) along s within $\mathcal{R}$, and connect each of them to all of p, q, x_{i-1}^+. All generated crossings are with edge (x_{i-1}^+, x_i^-) or among the added edges. The drawing is straight-line and good (no three edges cross in a point), and the number of added crossings is $\ell_1 + \binom{\ell_1}{2} = \frac{1}{2}\ell_1(\ell_1 + 1)$.

Placement on the outside. The outer face of the drawing is still formed by the edges connecting $T(C_i)$, since all vertices from the paragraph above were added inside $\mathcal{R}$ and thus in the interior of $T(C_i)$. We know that the vertex x_{i+1}^- in $T(C_i)$ will be lost in the next cluster C_{i+1} (if there is any); it will play a prominent role now. Since we may or may not have $x_{i+1}^- = x_{i-1}^+$, we label the vertices of $T(C_i)$ afresh as $\{x_{i+1}^-, p', q'\}$.

Define an open wedge $\mathcal{W}$ in the exterior of $T(C_i)$ emanating from x_{i+1}^- between the extensions of the edges (p', x_{i+1}^-) and (q', x_{i+1}^-) beyond x_{i+1}^-. Any point inside $\mathcal{W}$ can be connected via straight lines to all of p', q', x_{i+1}^- without any crossings. Consider a straight line s' through $\mathcal{W}$, not through any of x_{i+1}^-, p', q', and crossing (p', q'). Now place ℓ_2 vertices along s' within $\mathcal{W}$, and connect all of them to all of x_{i+1}^-, p', q' via straight lines. All generated crossings are among the added edges. The drawing is still straight-line and good, and the number of added crossings is $\binom{\ell_2}{2}$. The outer face of the resulting drawing is again a triangle with two corners being p' and q' and the third corner being a vertex that was added on s'. We assign this latter vertex the role of the emerging vertex x_i^+; the other inserted vertices are the necessary singletons. With this, the invariant holds since $T(C_{i+1}) = T(C_i) \cup \{x_i^+\} \setminus \{x_{i+1}^-\}$.

This finishes the description of the drawing algorithm. We claim that the final drawing has the minimum possible number of crossings: We first give an upper bound on the number of crossings that we achieve, and then show that any drawing requires this number.

▶ **Lemma 2.** *The above algorithm produces at most $\sum_{i=1}^{\kappa} \lfloor \frac{1}{2}(n(C_i) - 3) \rfloor \lfloor \frac{1}{2}(n(C_i) - 4) \rfloor$ crossings.*

Proof. The algorithm started with a planar drawing of K_4. We argued above that the i-th iteration (drawing C_i, which contains ℓ singletons) added

$$\frac{1}{2}\ell_1(\ell_1 + 1) + \frac{1}{2}\ell_2(\ell_2 - 1) = \lfloor \frac{1}{2}(\ell + 1) \rfloor \lfloor \frac{1}{2}(\ell + 2) \rfloor$$

crossings, where $\ell_1 = \lfloor (\ell + 1)/2 \rfloor$ and $\ell_2 = \lceil (\ell + 1)/2 \rceil$. Finally, observe that $\ell = n(C_i) - 5$ since all vertices of C_i except $T(C_i) \cup \{x_i^+, x_i^-\}$ are singletons. ◀

▶ **Lemma 3.** *Any good drawing of G requires at least $\sum_{i=1}^{\kappa} \lfloor \frac{1}{2}(n(C_i) - 3) \rfloor \lfloor \frac{1}{2}(n(C_i) - 4) \rfloor$ crossings.*

Proof. From Observation 1 we know that each cluster C_i contains a biclique $B(C_i) := K_{3,n(C_i)-3}$. By Zarankiewicz' formula, $K_{3,m}$ needs $\lfloor m/2 \rfloor \lfloor (m-1)/2 \rfloor$ crossings in any drawing. Thus, within each cluster we only introduce the optimal number of crossings.

However, we must argue that it is impossible for one crossing to belong to two or more clusters in an optimal drawing. This holds because nearly all of $V(C_i)$ does not belong to other clusters. More precisely, assume some other cluster C_j shares vertices with C_i; we may assume $j < i$. Then all common vertices must appear in the first bag $X = T(C_i) \cup \{x_i^-\}$ of C_i. However, only three edges of those induced by X are in $B(C_i)$, and all three of them are incident to x_i^-. Since adjacent edges do not cross in a good drawing, no crossing can be shared between $B(C_i)$ and $B(C_j)$. ◀

▶ **Theorem 4.** *There is a linear time algorithm to compute the exact crossing number $cr(G)$ of any maximal pathwidth-3 graph G. Furthermore, $cr(G) = \overline{cr}(G)$, and the algorithm gives rise to a straight-line drawing where the anchor edges are not crossed.*

Proof. Optimality follows from Lemmas 2 and 3. The second part of the claim follows from the first and third invariant in the above algorithmic description. It remains to argue linear running time. Computing a path decomposition of width 3 (if it exists) can be done in linear time [2,3]. This path decomposition can be turned into an alternating path decomposition in linear time as well. On it we compute $cr(G)$ as the sum in Lemma 2 in linear time. ◀

Assume we are interested in the drawing achieving this solution. The drawing algorithm uses $O(n)$ operations, but this does not immediately imply linear time, since coordinates may become very small. We also cannot list all crossings, as there can be $\Theta(n^2)$ many. If, however, we are careful about how to place anchor-triplets, then singletons can be inserted while keeping all vertices at grid-points of an $O(n) \times O(n)$-grid, and thus we require only linear time to compute and output the drawing. Details are given in the full version of the paper [1, Appendix B]. We summarize:

▶ **Theorem 5.** *Every maximal pathwidth-3 graph on n vertices has a crossing-minimum drawing that is good, straight-line, and lies on a $28n \times 29n$-grid. It can be found in $O(n)$ time.*

4 Approximation Algorithm for Pathwidth-3 Graphs

We now give an algorithm that draws graphs of pathwidth 3 (not necessarily maximal) such that the number of crossings is within a factor of 2 of the optimum. Roughly speaking, if the graph is 3-connected (technically, we will define a slightly weaker assumption 3-*traceable*), then the algorithm for maximal pathwidth-3 graphs is applied, and the number of crossings is within a factor of 2. If the graph is not 3-traceable, then it can be split and the arising subdrawings can be "glued" together without increasing the approximation ratio.

4.1 3-traceable graphs

We first analyze graphs that satisfy a condition that is weaker than 3-connectivity. Define a *non-anchor vertex* to be a vertex that occurs in exactly one bag. Those are exactly v_1, v_n, and all the singletons defined earlier.

▶ **Definition 6** (3-traceable graph)**.** A graph G with an alternating path decomposition $\mathcal{P}$ of width 3 is 3-*traceable* if every non-anchor vertex has degree at least 3, and for all $1 \leq i \leq \kappa$, edge (x^+_{i-1}, x^-_i) exists.

Assume we are given a 3-traceable graph G with an alternating path decomposition $\mathcal{P}$ of width 3. We can first maximize G (obtaining G') by adding all edges that have both ends in one bag, but are not in G' yet. We then apply the algorithm described in Section 3 to G', and finally delete the temporarily added edges again. We will show:

▶ **Lemma 7.** *Let G be a 3-traceable graph. Then the algorithm of Theorem 4 gives a drawing of G with at most $2cr(G)$ crossings.*

We first give a sketch of the proof. The main challenge is that a cluster C now does not necessarily contain a biclique $K_{3,n(C)-3}$. However, we can argue that G contains a subdivision of $K_{3,n(C)-3}$ that uses mostly vertices of C, but "borrows" a non-anchor vertex each (to play the role of x^-_i and x^+_i) from the nearest preceding and succeeding cluster that has such vertices. This subdivided $K_{3,n(C)-3}$ requires $cr(K_{3,n(C)-3})$ crossings. The main work is then in arguing that these subdivided bicliques cannot overlap much, or more precisely, that any crossing can belong to at most 2 of them. Lemma 7 then follows by applying the upper bound given in Lemma 2.

As before, let $C_1, \dots, C_\kappa$ be the clusters of G with anchor-triplets $T(C_1), \dots, T(C_\kappa)$, and recall that we have an age-order $\{v_1, \dots, v_n\}$.

There are three types of edges in G. Type I are edges that are incident to non-anchor vertices. Type II are edges that have the form (x^+_{i-1}, x^-_i) for some $2 \leq i \leq \kappa$. Finally, Type III are the remaining edges (they connect vertices of some anchor-triplet $T(C_i)$, $1 \leq i \leq \kappa$).

▶ **Observation 8.** *Consider a 3-traceable graph. For any $1 \leq i < j \leq \kappa$, there are three vertex-disjoint paths $\Pi_{i,j}$ from $T(C_i)$ to $T(C_j)$ that are either single vertices or consist exactly of the Type II edges (x^+_{k-1}, x^-_k) for $i < k \leq j$. Every non-anchor vertex attaches to the three different paths $\Pi := \Pi_{1,\kappa}$.*

Proof. For any $1 \leq i < \kappa$, we have $T(C_{i+1}) = T(C_i) \cup \{x^+_i\} \setminus \{x^-_{i+1}\}$. By 3-traceability of G, edge (x^-_{i+1}, x^+_i) exists and $\Pi_{i,i+1}$ consists of two paths of length 0 (the common vertices of the triplets) and the third path being this edge. We obtain arbitrary $\Pi_{i,j}$ by extending $\Pi_{i,i+1}$ via $\Pi_{i+1,j}$. Since G is 3-traceable, the non-anchor vertices have degree 3 and are adjacent to the vertices of the anchor-triplet of their unique cluster; those lie on distinct paths of Π. ◀

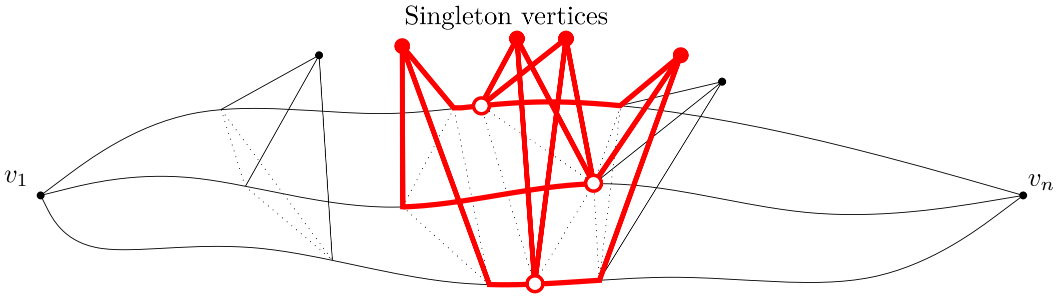


Figure 3 The structure of a 3-traceable graph. Dotted triangles mark anchor-triples with at least one adjacent singleton. In bold, we show one cluster biclique: the anchor vertices depicted as circles form one partition side. The left- and rightmost bold singleton is "borrowed" from the preceding and succeeding singleton-containing cluster, respectively.

This shows that G has $K_{3,n'}$ as a minor, where n' is the number of non-anchor vertices. Unfortunately this is not sufficient for crossing number arguments as contracting edges may increase the crossing number. Instead, we will use the above structure to extract a subdivision of $K_{3,n(C)-3}$ for each cluster C in such a way that these bicliques do not overlap "much."

▶ **Definition 9.** Let C_i, $1 \leq i \leq \kappa$, be a cluster with at least one singleton. The *cluster biclique of* C_i, denoted $\mathcal{B}(C_i)$, is a subdivision of $K_{3,n(C_i)-3}$ obtained as follows, cf. Fig. 3:

(a) The 3-side is formed by the three vertices of $T(C_i)$.

(b) Every singleton w that belongs to C_i (there are $n(C_i) - 5$ of them) is one of the vertices on the side that will have $n(C_i) - 3$ vertices. We know that $\deg(w) = 3$ by 3-traceability, and it is adjacent to all of $T(C_i)$ as required for the biclique.

(c) Let $i_- < i$ ($i_+ > i$) be maximal (minimal) such that cluster C_{i_-} (C_{i_+}, respectively) has a non-anchor vertex; among its non-anchor vertices, let w_- (w_+) be the youngest (oldest, respectively). If $i = 1$, we simply set $w_- := v_1$; if $i = \kappa$, we set $w_+ := v_n$. By Observation 8, we can establish three disjoint paths from w_- and w_+ to $T(C_i)$. Hence, add w_- and w_+ to the "big" side of $\mathcal{B}(C_i)$. Observe that in either case, w_- and w_+ are distinct from the the singletons of C_i and their paths to $T(C_i)$.

▶ **Lemma 10.** *Let e_1, e_2 be two edges of G without common endpoint. There are at most two cluster bicliques that contain both e_1 and e_2.*

Proof. We are done if at least one of e_1 and e_2 is of Type III, because then it belongs to no cluster biclique at all. Assume that one of e_1 and e_2 is of Type II, say $e_1 = (x^+_{i-1}, x^-_i)$ for some $2 \leq i \leq \kappa$. Edge e_1 may be used only for the cluster bicliques $\mathcal{B}(C_{j^-})$ and $\mathcal{B}(C_{j^+})$ where $j^- < i$ ($j^+ \geq i$) is the maximal (minimal) index such that cluster C_{j^-} (C_{j^+}, respectively) has singletons. The fact that e_1 belongs to at most two cluster bicliques proves the claim.

Finally, assume that both e_1 and e_2 are of Type I, i.e., incident to distinct non-anchor vertices, say $y_1 \in C_i$ and $y_2 \in C_{i'}$. Let $\mathcal{C}' \subseteq \mathcal{C}$ be the ordered subsequence of clusters that have at least one non-anchor vertex. A non-anchor vertex x can belong to at most three cluster bicliques, refer to Definition 9: the one of its "own" cluster $C \in \mathcal{C}'$, and those of the directly preceding and succeeding cluster in $\mathcal{C}'$. Assume that y_1 and y_2 are in three cluster bicliques. If $i = i'$, y_1 and y_2 are singletons of different age in C_i, and the two clusters directly preceding and succeeding C_i would have chosen distinct singletons of C_i, a contradiction. If $i \neq i'$, any overlap of three-element subsequences of $\mathcal{C}'$ with distinct middle clusters has size at most 2, a contradiction. ◀

Proof of Lemma 7. We know from Lemma 2 that the algorithm of Theorem 4 gives a drawing with at most $\sum_{C\in\mathcal{C}}\lfloor\frac{1}{2}(n(C)-3)\rfloor\lfloor\frac{1}{2}(n(C)-4)\rfloor$ crossings. We need to consider only clusters C that have at least one singleton; for any other cluster we have $n(C)=5$ and therefore its summand is 0. For any cluster C that has a singleton, we have $\mathcal{B}(C)$, a subdivision of $K_{3,n(C)-3}$, which requires at least $\lfloor\frac{1}{2}(n(C)-3)\rfloor\lfloor\frac{1}{2}(n(C)-4)\rfloor$ crossings in any good drawing $\mathcal{D}$ of G. Any crossing in $\mathcal{D}$ is created by two edges without common endpoints, and by Lemma 10, any such pair belongs to at most two cluster bicliques. Hence any drawing of G has at least $\frac{1}{2}\sum_{C\in\mathcal{C}}\lfloor\frac{1}{2}(n(C)-3)\rfloor\lfloor\frac{1}{2}(n(C)-4)\rfloor$ crossings, yielding the 2-approximation. ◀

4.2 General pathwidth-3 graphs

A pair of vertices $\{u,v\}$ of a 2-connected graph G is called a *separation pair* if $G-\{u,v\}$ is not connected. Assume that the pathwidth-3 graph G is 2-connected but not 3-traceable. We will show that we can split the graph at separation pairs within anchor-triplets, draw the cut-components recursively, and merge them without introducing additional crossings. We start with a more general auxiliary statement whose proof is in [1, Appendix C].

▶ **Lemma 11.** *Let G be a 2-connected graph with a separation pair $\{u,v\}$. Consider a partition of G into two edge-disjoint connected subgraphs H_1,H_2 with $H_1\cap H_2=\{u,v\}$. Define $H_i^+=H_i\cup\{(u,v)\}$ for $i=1,2$. Then $cr(H_1^+)+cr(H_2^+)\leq cr(G)$.*

We will draw cut-components inside triangles bounded by their three oldest vertices.

▶ **Lemma 12.** *Let G be a 2-connected graph with an alternating path decomposition $\mathcal{P}$ of width 3. Then there exists an algorithm to create a straight-line drawing of G with at most $2cr(G)$ crossings. All anchor-edges are drawn without crossings, and the three oldest vertices $\{v_1,v_2,v_3\}$ form the corners of the triangular convex hull of the drawing.*

Proof. We prove the result by induction on the structure and size of the graph.

Base case: G is 3-traceable or a K_4. If $G=K_4$, the claim is obvious. Otherwise, we apply Lemma 7. However, the algorithm of Theorem 4 used therein grows the drawing "outwards", while we would now like the oldest vertices to form the outer triangle. Thus we apply the algorithm for the reverse path decomposition; this makes (by suitably placing the last vertex) $T(C_1)=\{v_1,v_2,v_3\}$ the outer face and draws it as a triangle.

Induction Step: G is neither 3-traceable nor a K_4. For every non-anchor vertex $w\neq v_1$ of degree 2, let p_w,q_w be its adjacent anchor vertices. We can temporarily remove w from G, ensure that the reduced graph contains edge (p_w,q_w), draw the reduced graph, and—since (p_w,q_w) will be drawn crossing free by the induction hypothesis—reinsert each w with $(p_w,w),(w,q_w)$ crossing-free close to the drawing of (p_w,q_w). Similarly, we can remove v_1 if it has degree 2: We can choose an age-order of the reduced graph G' such that the neighbors of v_1 are among the three oldest vertices of G' and hence draw G' such that the neighbors of v_1 are on the outer-triangle; then v_1 can be reinserted on the outside to form the desired outer triangle. If the graph became 3-traceable by these operations, we are done (base case). Otherwise, we can now assume that all non-anchor vertices have degree 3.

Since G is not 3-traceable, $(x_{i-1}^+,x_i^-)\notin G$ for some $2\leq i\leq\kappa$. There exists a unique bag X_j, the common bag of C_{i-1} and C_i, that contains both x_{i-1}^+ and x_i^-. Let p,q be the two other vertices in this bag, and observe that $T(C_{i-1})=\{p,q,x_i^-\}$ while $T(C_i)=\{p,q,x_{i-1}^+\}$. Let G_ℓ be the graph induced by all vertices that appear in bags $\mathcal{P}_\ell:=[X_1,X_{j-2}]$, and let G_r be the graph induced by all vertices that appear in bags

$\mathcal{P}_r := [X_{j+2}, X_\xi]$. Any edge of G appears in G_ℓ or G_r, since $\{x_i^-, x_{i-1}^+\}$ is the only vertex-pair that existed in bags of $\mathcal{P}$, but neither of $\mathcal{P}_\ell$ nor $\mathcal{P}_r$. Clearly, $\{p, q\}$ is a separation pair with $G_\ell \cap G_r = \{p, q\}$.

Define $G_\ell^+ = G_\ell \cup \{(p, q)\}$ and $G_r^+ = G_r \cup \{(p, q)\}$. By the addition of edge (p, q) (if it did not already exist), both graphs are 2-connected. Apply induction to G_r^+ (with path decomposition $\mathcal{P}_r$) and G_ℓ^+ (with the path decomposition $\mathcal{P}_\ell$). Since p, q belong to the first bag of $\mathcal{P}_r$, we can ensure that they are among the three oldest vertices of G_r^+. We obtain two drawings $\mathcal{D}_1^+, \mathcal{D}_2^+$ in both of which (p, q) is not crossed. We can insert (affinely transformed) $\mathcal{D}_2^+$, which has (p, q) on its bounding triangle, along (p, q) in $\mathcal{D}_1^+$ without additional crossings. Finally, we remove edge (p, q) from the resulting drawing if $(p, q) \notin E(G)$.

By induction hypothesis, $cr(\mathcal{D}_\ell^+) \leq 2cr(G_\ell^+)$ and $cr(\mathcal{D}_r^+) \leq 2cr(G_r^+)$. By Lemma 11, $cr(G_\ell^+) + cr(G_r^+) \leq cr(G)$ and since the gluing gave no new crossings, the claim follows. ◀

We are now ready to establish the theorem for general pathwidth-3 graphs.

▶ **Theorem 13.** *Let G be any pathwidth-3 graph. We have $\overline{cr}(G) \leq 2cr(G)$, and a linear time algorithm to create a good straight-line drawing of G with at most $2cr(G)$ crossings.*

Proof. (Sketch) If G is 2-connected, then the result holds by Lemma 12. It is well known that $cr(G)$ is additive over the 2-connected components of G. When gluing at cut-vertices, the cut-vertex must be on the outer face of the drawing to be inserted into the other. We can achieve this while maintaining a straight-line drawing by choosing appropriate path decompositions; see [1, Appendix D]. The running time follows as in Theorem 4. ◀

5 Approximation Algorithm for Graphs of Higher Pathwidth

We now study the crossing number of graphs that have pathwidth $\mathbf{w} \geq 4$, and are maximal within this class. We give an algorithm to draw such graphs, and show that the number of crossings in the resulting drawing is within a factor of $4\mathbf{w}^3$ of the crossing number. As opposed to Section 3, the drawings we create here are not straight-line drawings.

As before we assume that we have an alternating path decomposition $\mathcal{P} = \{X_i\}_{1 \leq i \leq \xi}$ of width $\mathbf{w}$. We again use the *age-order* $\{v_1, \dots, v_n\}$ of the vertices of G. Define G_i to be the graph induced by vertices $v_1, \dots, v_i$, and use $\deg_{G_i}(v)$ to denote the number of neighbors that v has within graph G_i. For any $1 \leq i \leq n$, let the *predecessors* of vertex v_i be those neighbors that are older. We will only use this concept for $i \geq \mathbf{w} + 1$, which implies that v_i has exactly $\mathbf{w}$ predecessors by maximality of G. We enumerate them as $\{p_1^i, \dots, p_{\mathbf{w}}^i\}$ in age-order, with p_1^i the oldest.

Drawing algorithm. We create a drawing of G by starting with $G_{\mathbf{w}+1}$ (the graph induced by $v_1, \dots, v_{\mathbf{w}+1}$) and then iteratively adding vertex v_i. We maintain the following invariants for the drawing of G_i (see also Figure 4):

- Vertex v_j is drawn at $(j, 0)$ for all $1 \leq j \leq i$.
- The drawing is contained in the half-space $\{(x, y) : x \leq i\}$.
- All vertices w in the bag introducing v_i are *bottom-visible*, i.e., the vertical ray downward from w does not intersect any edge.

We start by placing $v_1, \dots, v_{\mathbf{w}+1}$ at their specified coordinates, and draw the edges between them as half-circles above the x-axis. This satisfies the above invariants and gives rise to $\binom{\mathbf{w}+1}{4}$ crossings since crossings are in 1-to-1-correspondence with subsets of 4 vertices.

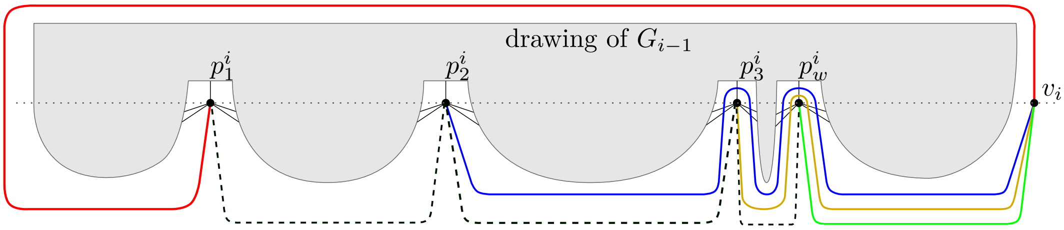


Figure 4 The construction for higher pathwidth: edge routings when adding vertex v_i.

Assume G_{i-1} is drawn and consider v_i, for $i \geq \mathbf{w}+2$. Place v_i as specified, i.e., to the right of all previous vertices and edges. Let $p_1^i, \ldots, p_{\mathbf{w}}^i$ be the predecessors of v_i, all of which are bottom-visible by the invariant. We draw the edges to them using two different methods (and then redraw previous edges as a third step for each i). See also Figure 4.

- The edge to p_1^i (the oldest predecessor) is routed counterclockwise around the drawing of G_{i-1} until it is below but slightly to the left of p_1^i, from where it connects to p_1^i. We need no crossings, and all predecessors remain bottom-visible.
- All other $\mathbf{w}-1$ edges incident to v_i are routed together as a bundle from v_i leftward below the drawing of G_{i-1}. This allows v_i to be bottom-visible. Whenever the bundle is slightly to the right of some p_k^i, $\mathbf{w} \geq k \geq 2$, one of the bundle's lines (the lowest one) connects to p_k^i. The remaining bundle lines go counterclockwise around p_k^i, in its direct vicinity, until they are to the left of p_k^i and below G_{i-1}. The bundle hence crosses every edge incident to p_k^i in G_{i-1}, but no other edges, and p_k^i remains bottom-visible. This drawing scheme continues until the last bundle line connects to p_2^i.
- Finally, we redraw the edges (p_{k-1}^i, p_k^i) for $3 \leq k \leq \mathbf{w}$; they exist by maximality. Both ends of any such edge are bottom-visible, so we can redraw it without crossing below the entire drawing, including the newly drawn edges from v_i. We remove the previous drawings of these edges and retain bottom-visibility of the vertices in the current bag.

In the full paper [1, Appendix E] we analyze of the number of crossings and obtain:

▶ **Theorem 14.** *Let G be a maximal graph of pathwidth $\mathbf{w} \geq 4$. The described algorithm runs in linear time and finds a drawing of G with at most $2(\mathbf{w}-1)(\mathbf{w}-2)(2\mathbf{w}-4)cr(G) \leq 4\mathbf{w}^3 cr(G)$ crossings. In particular, for any constant pathwidth $\mathbf{w}$, we have an $O(1)$-approximation of the crossing number. The drawing is poly-line on a $4n \times \mathbf{w}n$ grid.*

6 Conclusions and Open Questions

We have shown that the path decomposition of a graph can be used to efficiently compute or bound the crossing number of a graph. This is the first successful use of such graph decomposition for crossing numbers (besides the use of a tree decomposition in the special case that $cr(G)$ is bounded by a constant [14, 18]). Several interesting questions remain:

- Can we attain stronger approximation results for general pathwidth-3 graphs? The proven ratio of 2 may simply be due to a too weak lower bound, and we, in fact, do currently not know an instance where the algorithm does not obtain the optimum.
- Can we approximate $cr(G)$ for arbitrary (not maximal) pathwidth-$\mathbf{w}$-graphs?
- In [1] we only showed weak NP-completeness for the weighted crossing number version on pathwidth-restricted graphs. Can this be strengthened to unweighted graphs?

Finally, there is of course the question whether we can use the stronger tool of tree decompositions, instead of path decompositions, to achieve crossing number results.

References

1 T. Biedl, M. Chimani, M. Derka, and P. Mutzel. Crossing number for graphs with bounded pathwidth. *CoRR*, abs/1612.03854, 2016. URL: `http://arxiv.org/abs/1612.03854`.

2 H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. `doi:10.1137/S0097539793251219`.

3 H.L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996. `doi:10.1006/jagm.1996.0049`.

4 D. Bokal. On the crossing numbers of cartesian products with paths. *J. Comb. Theory Ser. B*, 97(3):381–384, May 2007. `doi:10.1016/j.jctb.2006.06.003`.

5 S. Cabello. Hardness of approximation for crossing number. *Discrete & Computational Geometry*, 49(2):348–358, 2013. `doi:10.1007/s00454-012-9440-6`.

6 S. Cabello and B. Mohar. Crossing number and weighted crossing number of near-planar graphs. *Algorithmica*, 60(3):484–504, 2011. `doi:10.1007/s00453-009-9357-5`.

7 M. Chimani and P. Hliněný. A tighter insertion-based approximation of the crossing number. *Journal of Combinatorial Optimization*, pages 1–43, 2016. `doi:10.1007/s10878-016-0030-z`.

8 M. Chimani and P. Hliněný. Inserting multiple edges into a planar graph. In *SoCG 2016*, pages 30:1–30:15. LIPIcs, 2016. `doi:10.4230/LIPIcs.SoCG.2016.30`.

9 M. Chimani, P. Hliněný, and P. Mutzel. Vertex insertion approximates the crossing number for apex graphs. *European Journal of Combinatorics*, 33:326–335, 2012.

10 J. Chuzhoy. An algorithm for the graph crossing number problem. In *STOC '11*, pages 303–312. ACM, 2011.

11 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. `doi:10.1016/0890-5401(90)90043-H`.

12 J. Fox, J. Pach, and A. Suk. Approximating the rectilinear crossing number. In *GD 2016*, LNCS 9801, pages 413–426. Springer, 2016. `doi:10.1007/978-3-319-50106-2_32`.

13 I. Gitler, P. Hliněný, J. Leanos, and G. Salazar. The crossing number of a projective graph is quadratic in the face-width. *Electronic Journal of Combinatorics*, 15(1):#R46, 2008.

14 M. Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004.

15 P. Hliněný. Crossing-number critical graphs have bounded path-width. *J. Comb. Theory, Ser. B*, 88(2):347–367, 2003. `doi:10.1016/S0095-8956(03)00037-6`.

16 P. Hliněný and M. Chimani. Approximating the crossing number of graphs embeddable in any orientable surface. In *SODA '10*, pages 918–927, 2010.

17 P. Hliněný and G. Salazar. Approximating the crossing number of toroidal graphs. In *ISAAC '07*, LNCS 4835, pages 148–159. Springer, 2007.

18 K-I. Kawarabayashi and B. Reed. Computing crossing number in linear time. In *STOC '07*, pages 382–390, 2007.

19 D.J. Kleitman. The crossing number of $K_{5,n}$. *J. of Comb. Theory*, 9(4):315–323, 1970. `doi:10.1016/S0021-9800(70)80087-4`.

20 M. Klešč and J. Petrillová. The crossing numbers of products of path with graphs of order six. *Discussiones Mathematicae Graph Theory*, 33(3):571–582, 2013.

21 T. Kloks. *Treewidth, Computations and Approximations*. LNCS 842. Springer, 1994. `doi:10.1007/BFb0045375`.

22 F.T. Leighton. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-exchange Graph and Other Networks*. MIT Press, Cambridge, MA, USA, 1983.

23 S. Pan and R.B. Richter. The crossing number of K_{11} is 100. *Journal of Graph Theory*, 56(2):128–134, 2007. `doi:10.1002/jgt.20249`.

24 R.B. Richter and G. Salazar. The crossing number of $P(N,3)$. *Graphs and Combinatorics*, 18(2):381–394, 2002. `doi:10.1007/s003730200028`.

25 M. Schaefer. The graph crossing number and its variants: A survey. *Electronic Journal of Combinatorics*, #DS21, May 15, 2014.

26 I. Vrt'o. Crossing numbers of graphs: A bibliography. `ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf`, 2014.

27 D.R. Wood and J.A. Telle. Planar decompositions and the crossing number of graphs with an excluded minor. *New York J. Math.*, 13:117–146, 2007.

An Improved Algorithm for Computing All the Best Swap Edges of a Tree Spanner*

Davide Bilò[1], Feliciano Colella[2], Luciano Gualà[3], Stefano Leucci[4], and Guido Proietti[5]

1 Università di Sassari, Italy
davide.bilo@uniss.it

2 Gran Sasso Science Institute, L'Aquila, Italy
feliciano.colella@gssi.it

3 Università di Roma "Tor Vergata", Italy
guala@mat.uniroma2.it

4 ETH Zürich, Switzerland
stefano.leucci@inf.ethz.ch

5 Università degli Studi dell'Aquila, and Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma, Italy
guido.proietti@univaq.it

Abstract

A *tree σ-spanner* of a positively real-weighted n-vertex and m-edge undirected graph G is a spanning tree T of G which approximately preserves (i.e., up to a multiplicative *stretch factor* σ) distances in G. Tree spanners with provably good stretch factors find applications in communication networks, distributed systems, and network design. However, finding an optimal or even a good tree spanner is a very hard computational task. Thus, if one has to face a *transient* edge failure in T, the overall effort that has to be afforded to rebuild a new tree spanner (i.e., computational costs, set-up of new links, updating of the routing tables, etc.) can be rather prohibitive. To circumvent this drawback, an effective alternative is that of associating with each tree edge a best possible (in terms of resulting stretch) *swap edge* – a well-established approach in the literature for several other tree topologies. Correspondingly, the problem of computing *all* the best swap edges of a tree spanner is a challenging algorithmic problem, since solving it efficiently means to exploit the structure of shortest paths not only in G, but also in all the scenarios in which an edge of T has failed. For this problem we provide a very efficient solution, running in $O(n^2 \log^4 n)$ time, which drastically improves (almost by a quadratic factor in n in dense graphs!) on the previous known best result.

1998 ACM Subject Classification G.2.2 Graph Theory, Graph algorithms, Trees

Keywords and phrases Transient edge failure, Swap algorithm, Tree spanner

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.14

1 Introduction

The problem of computing *all the best swap edges* (ABSE) of a tree has a long and rich algorithmic tradition. Basically, let $G = (V(G), E(G), w)$ be an n-vertex and m-edge 2-edge-connected undirected graph, with edge-weight function $w : E(G) \to \mathbb{R}^+$, and assume we are given a spanning tree T of G, which was computed by addressing some criterion (i.e.,

[*] A full version of the paper is available at https://arxiv.org/abs/1710.01516.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 14; pp. 14:1–14:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

objective function) ϕ. Then, the problem is that of computing a BSE for every edge $e \in E(T)$, namely an edge $f \in E(G) \setminus E(T)$ such that the swap tree $T_{e/f}$ obtained by swapping e with f in T optimizes some objective function ϕ' out of all possible swap trees. Quite reasonably, the function ϕ' must be related (if not coinciding at all) with ϕ.

The first immediate motivation for studying an ABSE problem comes from the edge fault-tolerance setting – a commonly accepted framework. Broadly speaking, the algorithmic question here is to design *sparse* subgraphs that guarantee a proper level of functionality even in the presence of an edge failure. In such a context, the rationale of an ABSE-based solution is the following: operations are normally performed on a (possibly optimal) spanning tree, and whenever an edge failure takes place, a corresponding BSE is plugged in. This way, the connectivity is reestablished in the most prompt and effective possible way (see also [13, 19] for some additional practical motivations).

Besides their practical relevance, ABSE problems have also an interesting theoretical motivation. Indeed, swapping can be reviewed as an exploration of the space of the perturbed (w.r.t. an edge swap) solutions to a given spanning tree optimization problem. Thus, the algorithmic challenge of solving efficiently an ABSE problem is related with the understanding of the structure of this space of perturbed solutions. And this is exactly why each ABSE problem has its own combinatorial richness, and thus requires a specific approach to be solved efficiently. Then, different ABSE problems have required the use of completely different approaches and methods in order to obtain efficient solutions. For instance, the most famous and studied ABSE problem comes when T is a *minimum spanning tree* (MST) of G. In this case, a best swap is of course a swap edge minimizing the *cost* (i.e., sum of the edge weights) of the swap tree, i.e., a swap edge of minimum weight (and we know this produces a MST of the perturbed graph). This problem is also known as the MST *sensitivity analysis* problem, and can be solved in $O(m \log \alpha(m, n))$ time [18], where α denotes the inverse of the Ackermann function, by using an efficient data structure, namely the *split-findmin* [11]. This was improving on another efficient solution given by Tarjan [21], running in $O(m\,\alpha(m, n))$ time and making use of the *transmuter*, namely a compact way of representing the cycles of a graph. Other data structures which revealed their usefulness to solve efficiently ABSE problems include *kinetic heaps* [6], *top trees* [3], *mergeable heaps* [17], and many others.

In this paper, we focus on the ABSE problem on the elusive spanning tree structure, namely the *tree spanner* (`ABSE-TS` problem in the following). A tree spanner is built with the aim of preserving node-to-node distances in G. Indeed, the *stretch factor* σ of a spanning tree T of G is defined as the *maximum*, over all the pairs $u, v \in V(G)$, of $d_T(u, v)/d_G(u, v)$, where d_T and d_G denote distances in T and G, respectively. Correspondingly, an *optimal* tree spanner has minimum stretch out of all the spanning trees of G. Unfortunately, finding an optimal tree spanner is notoriously an APX-hard problem, with no known $o(n)$-approximation. Hence, once a given solution undergoes a transient edge failure, the recomputation from scratch of a new (near) optimal solution is computationally unfeasible. Thus, swapping in a tree spanner is even more attractive than in general, and indeed the `ABSE-TS` problem was studied in [9], where the authors devised two solutions for both the weighted and the unweighted case, running in $O(m^2 \log n)$ and $O(n^3)$ time, respectively, and using $O(m)$ and $O(n^2)$ space, respectively. However, there the authors assume that a BSE is an edge minimizing the stretch of the swap tree w.r.t. distances in the *original* graph G, and not in the graph G deprived of e, say $G - e$. This contrasts with the general assumption (and the intuition) that the quality of a swap tree should be evaluated in the surviving graph. Hence, in [3] the authors resorted to such a standard setting, and provided two efficient linear-space solutions for both the weighted and the unweighted case, running in $O(m^2 \log \alpha(m, n))$ and

$O(mn \log n)$ time, respectively, and both using linear space. Notice that from a computational point of view, as shown in [3], the two settings are substantially equivalent, so our solutions can be used to improve the results given in [9] as well.

1.1 Our result

In this paper, we present a new algorithm that solves the ABSE-TS problem in $O(n^2 \log^4 n)$ time and $O(n^2 + m \log^2 n)$ space. Thus, our solution improves on the running time of both the algorithms provided in [3], for weighted and unweighted graphs, respectively, whenever $m = \Omega(n \log^3 n)$. Most remarkably, for dense weighted graphs, the improvement is almost quadratic in n.

To put into focus our result, it is worth noticing that, as observed in [9], the estimation of the stretch of the swap tree induced by a *single* swap edge f for a given failing edge e, would in principle ask for the evaluation of the stretch of $O(m)$ relevant pairs of nodes in G, namely the endvertices of all the non-tree edges that may serve as swap edge for e besides f. And in fact, a *critical edge* for f is the one whose endvertices maximize such a stretch out of these non-tree edges, and two swap edges will be essentially compared on the basis of their stretch w.r.t. their critical edge. This is basically the reason why both previous approaches take $\Omega(m^2)$ time. Thus, to avoid such a bottleneck, we drastically reduce, on the one hand, the number of candidate best swap edges, and on the other hand, the number of potential critical edges that need to be checked. More precisely, for each of the $n - 1$ considered edges in T, we succeed in reducing to $O(n \log n)$ the number of best swap edge candidates, and for each one of them we just need to check $O(\log^2 n)$ possible critical edges. The key ingredients to reach such a goal are the following:

- A *centroid decomposition* of T, which consists of a log-depth hierarchical decomposition of the vertices in T; a careful use of such a decomposition, combined with a set of preprocessing steps that associate various information with the tree nodes, allows us to reduce the number of candidate BSEs and of their corresponding candidate critical edges. As far as we know, this is the first time that such a decomposition is used to solve an ABSE problem, and we believe it will possibly be useful in other contexts as well.
- The second ingredient is given by the dynamic maintenance of the *upper envelopes* of a set of linear functions. Each of these functions is associated with a non-tree edge, and whenever the failure of a given tree edge is considered, it expresses the stretch such a non-tree edge induces w.r.t. a variable candidate BSE. This way, when we have to find a critical edge for a given candidate BSE f, we have to select the *maximum* out of all the functions once they are evaluated in f. In geometric terms, this translates into the maintenance of the upper envelope of a set of functions, with the additional complication that, for consistency reasons, this set of functions must be suitably partitioned into groups according to the underlying centroid decomposition, and moreover these groups are dynamic, since they depend on the currently considered tree edge.

1.2 Related work

The research on tree spanners is very active, also due to the strong relationship with the huge literature on *spanners*, where distances in G are approximately preserved through a *sparse* spanning subgraph. As mentioned before, finding an optimal tree spanner is a quite hard problem. More precisely, on weighted graphs, if G does not admit a tree 1-spanner (i.e., a spanning tree with $\sigma = 1$, which can be established in polynomial time [8]), then the problem is not approximable within any constant factor better than 2, unless P=NP [15]. In terms of approximability, no non-trivial upper bounds are known, except for the $O(n)$-approximation factor returned by a *minimum spanning tree* (MST) of G. If G is *unweighted*, things go slightly better. More precisely, in this case the problem becomes $O(\log n)$-approximable, while unless P=NP, the problem is not approximable within an additive term of $o(n)$ [10]. Moreover, the corresponding decision problem of establishing whether G admits a tree spanner with stretch σ is NP-complete for every fixed $\sigma \geq 4$ (for $\sigma = 2$ it is polynomial-time solvable [8], while for $\sigma = 3$ the problem is open). Finally, it is known that constant-stretch tree spanners can be found for several special classes of (unweighted) graphs, like strongly chordal, interval, and permutation graphs (see [7] and the references therein).

Concerning the problem of swapping in spanning trees, this has received a significant attention from the algorithmic community. There is indeed a line of papers that address ABSE problems starting from different types of spanning trees. Just to mention a few, besides the MST, we recall the *minimum diameter spanning tree* (MDST), the *minimum routing-cost spanning tree* (MRCST), and the *single-source shortest-path tree* (SPT). Concerning the MDST, a best swap is instead an edge minimizing the *diameter* of the swap tree [12, 16], and the best solution runs in $O(m \log \alpha(m, n))$ time [6]. Regarding the MRCST, a best swap is clearly an edge minimizing the *all-to-all routing cost* of the swap tree [22], and the fastest solution for solving this problem has a running time of $O\left(m2^{O(\alpha(n,n))} \log^2 n\right)$ [5]. Concerning the SPT, the most prominent swap criteria are those aiming to minimize either the maximum or the average distance from the root, and the corresponding ABSE problems can be addressed in $O(m \log \alpha(m, n))$ time [6] and $O(m\, \alpha(n, n) \log^2 n)$ time [20], respectively. Recently, in [4], the authors proposed two new criteria for swapping in a SPT, which are in a sense related with this paper, namely the minimization of the maximum and the average stretch factor from the root, for which they proposed an efficient $O(mn + n^2 \log n)$ and $O(mn \log \alpha(m, n))$ time solution, respectively.

Finally, for the sake of completeness, we mention that for the related concept of *average* tree σ-spanners, where the focus is on the average stretch w.r.t. all node-to-node distances, it was shown that every graph admits an average tree $O(1)$-spanner [1].

1.3 Preliminary definitions

Let $G = (V(G), E(G), w)$ be a 2-edge-connected, edge-weighted, and undirected graph with cost function $w : E(G) \to \mathbb{R}^+$. We denote by n and m the number of vertices and edges of G, respectively. If $X \subseteq V(G)$, let $E(X)$ be the set of edges incident to at least one vertex in X. When $X = \{v\}$, we may write $E(v)$ instead of $E(\{v\})$. Given an edge $e \in E(G)$, we will denote by $G - e$ the graph obtained from G by removing edge e. Similarly, given a vertex $v \in V(G)$, we will denote by $G - v$ the graph obtained from G by removing vertex v and all its incident edges. Given an edge $e \in E(T)$, we let $S(e)$ be the set of all the *swap edges* for e, i.e., all edges in $E(G) \setminus \{e\}$ whose endpoints lie in two different connected components of $T - e$. We also define $S(e, X) = S(e) \cap E(X)$, and $S(e, X, Y) = S(e) \cap E(X) \cap E(Y)$. When $X = \{v\}$, we will simply write $S(e, v)$ in lieu of $S(e, \{v\})$. For any $e \in E(T)$ and $f \in S(e)$, let $T_{e/f}$ denote the *swap tree* obtained from T by replacing e with f.

Given two vertices $x, y \in V(G)$, we denote by $d_G(x, y)$ the *distance* between x and y in G. We define the *stretch factor* of the pair (x, y) w.r.t. G and T as $\sigma_G(T, x, y) = \frac{d_T(x,y)}{d_G(x,y)}$. Accordingly, the stretch factor $\sigma_G(T)$ of T w.r.t. G is defined as $\sigma_G(T) = \max_{x,y \in V(G)} \sigma_G(T, x, y)$.

▶ **Definition 1** (Best Swap Edge). An edge $f^* \in S(e)$ is a *best swap edge* (BSE) for e if $f^* \in \arg\min_{f \in S(e)} \sigma_{G-e}(T_{e/f})$.

In the sequel, in order to solve the ABSE-TS problem, we will show how to efficiently find a BSE for every edge e of a tree spanner T of G.

Algorithm: ABSE-TS(G, T)

```
$\mathcal{T} \leftarrow$ Centroid decomposition of $T$;
foreach $e \in E(T)$ in postorder do                                  // $n-1$ phases
    $U_e \leftarrow$ vertices of the component of $T - e$ that contains the root of $T$;
    $f^* \leftarrow \perp$;                                          // Current BSE for $e$
    foreach $v \in U_e$ do                                           // $O(n)$ sub-phases
        compute a $v$-BSE $f$ for $e$ and the corresponding stretch factor;   // This takes
          $O(\log^4 n)$ time by using $\mathcal{T}$ and the dynamic maintenance of the upper
          envelopes associated with the swap edges, as shown in Section 3
        if $\sigma_{G-e}(T_{e/f}) < \sigma_{G-e}(T_{e/f^*})$ then $f^* \leftarrow f$;
    return $f^*$ as BSE for $e$ and continue with the next phase.
```

2 High-level description of the algorithm

It is useful to consider the tree T as rooted at any fixed vertex, and to assume, w.l.o.g., that T is binary. Indeed, if T is not binary, then it is possible to transform G and T into an equivalent graph G' and a corresponding binary spanning tree T', with $|V(G')| = \Theta(n)$ and $|E(G')| = \Theta(m)$, and such that a BSE for any edge of T is univocally associated with a BSE for a corresponding edge of T'. This transformation uses standard techniques and can be performed in linear time.

As a preprocessing step, we compute a *centroid decomposition* of T. A *centroid* of an n-vertex tree is a vertex whose removal splits T into subtrees of size at most $n/2$ [14]. A centroid decomposition of T can be computed in $O(n \log n)$ time, and can be represented by a tree $\mathcal{T}$ of height $O(\log n)$, whose nodes are actually subtrees of T. $\mathcal{T}$ is recursively defined as follows: the root of $\mathcal{T}$ is T. Then, let τ be a node of $\mathcal{T}$ (i.e., a subtree of T) such that τ contains more than one vertex, and let c be a centroid of τ. Since T is binary, the forest $\tau - c$ contains at most 3 trees, that we call τ_c^1, τ_c^2, and τ_c^3 (if $\tau - c$ generates less than 3 subtrees, we allow some τ_c^i to be the empty tree). Moreover, let τ_c^0 be the subtree of T containing the sole vertex c. Then, τ will have in $\mathcal{T}$ a child for each of the subtrees $\tau_c^i, i = 0, \dots, 3$ (see Figure 1 (a)). Since a centroid on a n-vertex tree can be found in linear time, the whole procedure requires $O(n \log n)$ time, and it is easy to see that the height of $\mathcal{T}$ is $O(\log n)$.

Our solution (see Algorithm ABSE-TS) works in $n - 1$ phases, one for each tree edge as considered in preorder w.r.t. T, and at the end of each phase returns a BSE for that edge. Let $e \in E(T)$ be the currently considered edge, and let U_e (resp. D_e) be the set of vertices that belong to the connected component of $T - e$ that contains (resp. does not contain) the root of T. We break down each of these phases into $O(n)$ additional sub-phases: when edge e is failing, we consider all the vertices in U_e and, for each such vertex v, we solve a restricted version of the ABSE-TS problem where we compute: (i) a *v-restricted best swap edge* (v-BSE for short), i.e., an edge $f \in \arg\min_{f \in S(e,v)} \sigma_{G-e}(T_{e/f})$, and (ii) the corresponding stretch factor $\sigma_{G-e}(T_{e/f})$. To simplify handling of special cases, whenever $S(e, v) = \emptyset$, we assume that $f = \perp$ and that $\sigma_{G-e}(T_{e/f}) = +\infty$. As we will see in the rest of the paper, the core of our algorithm is exactly the efficient computation of these v-BSEs and of their stretch factors. This is done trough a clever selection of a small set of *candidate* v-BSEs, as we will discuss in more detail in the next section. Once all the v-BSEs for e are computed, a BSE for e can be found as the one minimizing the associated stretch factor.

3 Computing efficiently a v-BSE

To show how a v-BSE for e can be computed efficiently, we need some preliminary definitions:

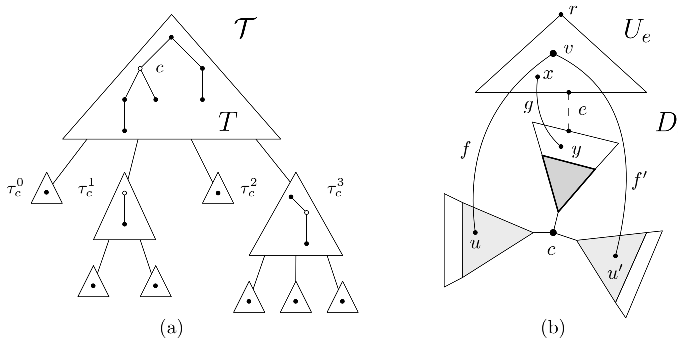


Figure 1 (a) An example of centroid decomposition of the tree T (which corresponds to the first vertex of $\mathcal{T}$). (b) and (c): Two of the four possible cases situation illustrated in Lemma 5. The subtree $\widehat{T}$ is represented by the three gray triangles along with the vertex c. f is a swap edge for e that minimizes $w(f) + d_T(u, c)$, and g is its corresponding critical edge. The (c, y)-*tree* of $\widehat{T}$ is drawn in bold. Notice that f and g do not need to be incident to $\widehat{T}$.

▶ **Definition 2** (Critical Edge). Given $e \in E(T)$ and a swap edge $f = (v, u) \in S(e, v)$, a *critical edge*[1] for f is an edge $g = (x, y) \in S(e)$ maximizing $\phi(f, g) := \dfrac{d_T(x, v) + w(f) + d_T(u, y)}{w(g)}$.

▶ **Definition 3** (Best Cut Edge). A *v-best cut edge* for e (v-BCE) is an edge $f \in S(e, v)$ minimizing $\varphi_e(f) = \max_{g \in S(e)} \phi(f, g)$.

Then, we will make use of the following property, which was given in [3]:

▶ **Proposition 4.** *Every v-BCE for e is a v-BSE for e.*

Let us first provide a high-level description of how we compute a v-BCE (i.e., a v-BSE) for e. The algorithm will compute $O(\log n)$ v-BCE *candidates*, the best of which will be a v-BCE for e. Informally speaking, each candidate f will be a swap edge close to the centroid of a certain subtree Λ of T. Depending on the position of a critical edge for f, the algorithm will recurse on a subtree of Λ and it will look for the next candidate. Thanks to the centroid decomposition of T, the number of recursions/candidates will then be $O(\log n)$.

The key ingredient for the correctness of our algorithm is the next lemma. Given a subtree $\widehat{T}$ of T, a vertex $c \in V(\widehat{T})$, and a vertex $y \in V(T)$, consider the first vertex z of the unique path from y to c in T that also belongs to $V(\widehat{T})$. The $(c, y)-$*tree* of $\widehat{T}$ is defined as follows: (1) if $z = c$, then it is the empty tree; otherwise (2) it is the tree of the forest $\widehat{T} - c$ that contains z. Then, the following holds (see also Figure 1 (b) and (c)):

▶ **Lemma 5.** *Let $\widehat{T}$ be a subtree of T such that $V(\widehat{T}) \subseteq D_e$, and let $c \in V(\widehat{T})$. Moreover, let $f = (v, u) \in S(e, v)$ be a swap edge for e that minimizes $w(f) + d_T(u, c)$, and let $g = (x, y)$ be a critical edge for f. Assume that $S(e, v, V(\widehat{T}))$ contains a v-BCE for e. If f is not a v-BCE for e, then $S(e, v, V(T'))$ contains a v-BCE for e, where T' is the (c, y)-tree of $\widehat{T}$.*

[1] Notice that this definition does not contain $d_{G-e}(x, y)$ at the denominator, as expected, since it already incorporates the property stated in the forthcoming Proposition 4.

Procedure FindBCE(Λ)

```
if $|V(\Lambda)| = 0$ then return $(\perp, \perp)$ ;
$c \leftarrow$ Centroid of $\Lambda$;
if $c \in U_e$ then
    $\tau \leftarrow$ unique child of $\Lambda$ in $\mathcal{T}$ that contains all the vertices in $V(\Lambda) \cap D_e$;
    return FindBCE ($\tau$);
else                                                                         // $c \in D_e$
    Compute an edge $f = (v,u) \in \arg\min_{(v,u) \in S(e,v)}\{w(v,u) + d_T(u,c)\}$ $g_1 = (x,y) \leftarrow$
     FindCritical($f, T$);              // Compute a critical edge for $f$ (see Sec. 3.1)
    $\tau \leftarrow (c,y)$-tree of $\Lambda$;            // Either $\tau$ is empty or it is a child of $\Lambda$ in $\mathcal{T}$
    $(f', g_2) \leftarrow$ FindBCE ($\tau$);
    if $\phi(f, g_1) \leq \phi(f', g_2)$ then return $(f, g_1)$; else return $(f', g_2)$;
```

Proof. Suppose that f is not a v-BCE for e, we will show that no swap edge $f' = (v, u') \in S(e, v)$ with $u' \notin V(T')$ can be a v-BCE for e. Indeed:

$$\begin{aligned}
\varphi(f') \geq \phi(f', g) &= \frac{d_T(x,v) + w(f') + d_T(u',y)}{w(g)} \\
&= \frac{d_T(x,v) + w(f') + d_T(u',c) + d_T(c,y)}{w(g)} \\
&\geq \frac{d_T(x,v) + w(f) + d_T(u,c) + d_T(c,y)}{w(g)} \geq \phi(f,g) = \varphi(f),
\end{aligned}$$

where we used the fact that $d_T(u', y) = d_T(u', c) + d_T(c, y)$ as either $u' = c$ or u' and y are in two different connected components of $T - c$. ◀

Lemma 5 allows us to design a recursive algorithm for computing a v-BCE for e, whose key steps are highlighted in Procedure `FindBCE` (notice that v and e are fixed). More precisely, the algorithm takes a tree Λ of the centroid decomposition $\mathcal{T}$ such that $V(\Lambda) \cap D_e \neq \emptyset$, and it computes a pair (f^*, g^*) such that if $S(e, v, V(\Lambda) \cap D_e)$ contains a v-BCE for e, then f^* is a v-BCE for e, and g^* is its critical edge. Procedure `FindBCE` makes use of an additional function `FindCritical`(f, T) that returns a critical edge for f w.r.t. the failure of e. The initial call will be `FindBCE`(T). In order to handle base cases, we assume $\phi(\perp, \perp) = +\infty$.

We now prove the correctness of the procedure:

▶ **Lemma 6.** *Procedure* `FindBCE`(T) *computes a v-BCE for e.*

Proof. Consider an invocation of the procedure and let Λ and (f^*, g^*) be its parameter and the edges it returns, respectively. We prove the following claim by induction on the cardinality of $V(\Lambda)$: if $S(e, v, V(\Lambda) \cap D_e)$ contains a v-BCE for e, then f^* is a v-BCE for e and g^* is a critical edge for f^*.

If $|V(\Lambda)| = 0$, then the claim trivially holds. Otherwise, $|V(\Lambda)| > 0$, and we distinguish two cases depending on the position of the centroid c of Λ. If $c \in U_e$, then there is only one child τ_c^j of Λ in $\mathcal{T}$ that contains all the vertices in $V(\Lambda) \cap D_e$, as otherwise the vertices in D_e would be disconnected in Λ. Hence, if $S(e, v, V(\Lambda) \cap D_e)$ contains a v-BCE for e, then $S(e, v, V(\tau_c^j) \cap D_e)$ also contains a v-BCE for e, and the claim follows by the inductive hypothesis (as $|V(\tau_c^j)| < |V(\Lambda)|$). The remaining case is the one in which $c \in D_e$, here the claim follows from Lemma 5 (where now $\widehat{T}$ is the subtree of T induced by $V(\Lambda) \cap D_e$) together with the inductive hypothesis. ◀

Next lemma provides an upper bound to the running time of the procedure:

▶ **Lemma 7.** *Procedure `FindBCE`(T) requires $O((\Gamma_f + \Gamma_{\texttt{FC}})\log n)$ time, where Γ_f and $\Gamma_{\texttt{FC}}$ is the time required to perform Steps 7 and 7, i.e., the time to find edge f, and to execute Procedure `FindCritical`, respectively.*

Proof. First of all, notice that Step 4 can be performed in $O(1)$ time, after a $O(\log n)$ preprocessing time in which we mark all the nodes of $\mathcal{T}$ on the path between the leaf of $\mathcal{T}$ containing the lower vertex of e (which clearly belongs to D_e) and the root of $\mathcal{T}$. Then, we only need to bound the depth of the recursion of the call `FindBCE`(T). Observe that each time Procedure `FindBCE`(Λ) recursively invokes itself on a tree Λ', we have that Λ' is a child of Λ in $\mathcal{T}$. The claim follows since the height of $\mathcal{T}$ is $O(\log n)$. ◀

Actually, the time to execute Step 7 is $O(\log n)$, after a preprocessing time and space of $O(n^2)$, by making use of *top-trees* [2]. Due to space limitations, the discussion of this result will appear in the full version of the paper. On the other hand, Procedure `FindCritical` will require $O(\log^3 n)$ time and $O(m \log^2 n)$ space, as we will show in the next two subsections.

3.1 Computing a critical edge for f

We will compute $O(\log^2 n)$ critical edge candidates for f and we will show that a critical edge for f will be one of them. More precisely, we look at $O(\log n)$ subtrees of the centroid decomposition $\mathcal{T}$ and, for each such subtree Λ, we will consider $O(\log n)$ subtrees Ψ to find a critical edge candidate having one endpoint in Ψ and the other in Λ. The choice of the $O(\log^2 n)$ pairs of trees is guided by the position of f, while the computation of a candidate for a given pair (Ψ, Λ) is the core of the procedure and is described in the next subsection.

▶ **Definition 8** ((Ψ, Λ)-Critical Edge)**.** Given a failing edge e and a swap edge $f = (v, u) \in S(e, v)$, and given two trees Ψ, Λ of the centroid decomposition $\mathcal{T}$, a (Ψ, Λ)-*critical edge* for f is an edge $g = (x, y) \in \arg\max_{g' \in S(e, V(\Psi) \cap U_e, V(\Lambda) \cap D_e)} \phi(f, g')$. When $\Psi = T$ we will refer to a (Ψ, Λ)-critical edge as a Λ-critical edge.

Let $f = (v, u) \in S(e, v)$ and let Λ be a tree of the centroid decomposition $\mathcal{T}$ such that $u \in V(\Lambda)$. Procedure `FindCritical` returns a Λ-critical edge for f, when edge e fails (such an edge always exists as f has one endpoint in U_e and the other in $V(\Lambda) \cap D_e$). Notice that the call `FindCritical`(f, T) in Procedure `FindBCE` computes a critical edge for f, since a T-critical edge for f is actually a critical edge for f.

Procedure `FindCritical` uses as a subroutine Procedure `FindCriticalCandidate`$(f$, Ψ, $\Lambda)$, which for the sake of clarity will be described in the next subsection. For the moment, it suffices to know that `FindCriticalCandidate` receives three inputs, i.e., edge $f = (v, u)$ and two subtrees Ψ, Λ of the centroid decomposition $\mathcal{T}$ such that $v \in \Psi$ and, either $u \notin V(\Lambda)$ or Λ is the tree containing the sole vertex u, and it returns a (Ψ, Λ)-critical edge for f. If no such edge exists, then `FindCriticalCandidate` returns $\perp$ and we assume that $\phi(f, \perp) = -\infty$.

▶ **Lemma 9.** *Let $f = (v, u) \in S(e, v)$, and let Λ be a tree of the centroid decomposition $\mathcal{T}$ such that $u \in V(\Lambda)$. Procedure `FindCritical`(f, Λ) returns a Λ-critical edge for f.*

Proof. The proof is by induction on the cardinality of $V(\Lambda)$.

If $|V(\Lambda)| = 1$, then the only vertex in Λ must be u and Procedure `FindCritical` invokes Procedure `FindCriticalCandidate`(f, T, Λ). Hence, assuming such a procedure is correct, it returns a (T, Λ)-critical edge, i.e., a Λ-critical edge. If $|V(\Lambda)| > 1$ then we distinguish two cases, depending on the position of the centroid c of Λ.

If $c \in D_e$ it is sufficient to notice that a Λ-critical edge for f must be incident to a tree τ_c^i for some $i = 0, 1, 2, 3$. Let j be the unique index in $\{0, 1, 2, 3\}$ such that $u \in V(\tau_c^j)$. If $j \neq i$

```
Procedure FindCritical($f = (v, u), \Lambda$)
if $V(\Lambda) = \{u\}$ then return FindCriticalCandidate($f$, $T$, $\Lambda$) ;
$c \leftarrow$ Centroid of $\Lambda$;
Let $j$ be the unique index in $\{0, 1, 2, 3\}$ such that $u \in V(\tau_c^j)$;
if $c \in U_e$ then return FindCritical($f, \tau_c^j$) ;
$\mathcal{G} \leftarrow$ {FindCriticalCandidate($f, T, \tau_c^i$) $: i = 0, 1, 2, 3 \wedge i \neq j\}$;          // Here $c \in D_e$
$g_1 \leftarrow \arg\max_{g \in \mathcal{G}} \phi(f, g)$;
$g_2 \leftarrow$ FindCritical($f, \tau_c^j$);
return $\arg\max_{g \in \{g_1, g_2\}} \{\phi(f, g)\}$;
```

then, assuming Procedure FindCriticalCandidate is correct, it returns a (T, Λ)-critical edge g_1 (and hence a Λ-critical edge) for f. Procedure FindCritical then returns either g_1 or another edge g such that $\phi(f, g) = \phi(f, g_1)$. If $j = i$, the algorithm is recursively invoked and, since $|V(\tau_c^i)| < |V(\Lambda)|$ we know, by the induction hypothesis, that it correctly returns a τ_c^i-critical edge for f, which is also Λ-critical edge for f.

If $c \in U_e$, then we know that there is at most one τ_c^i containing one or more vertices in D_e (as otherwise the vertices in $V(\Lambda) \cap D_e$ would be disconnected in Λ, a contradiction). Moreover, since $u \in V(\Lambda) \cap D_e$, there is exactly one such tree τ_c^i, namely τ_c^j. The algorithm recursively invokes itself on τ_c^j and, since $|V(\tau_c^j)| < |V(\Lambda)|$, we know, by induction hypothesis, that it returns a τ_c^j-critical edge for f, which is also Λ-critical edge for f. ◀

▶ **Lemma 10.** *Procedure FindCritical(f, Λ) requires $O(\Gamma_{\mathsf{FCC}} \cdot \log n)$ time, where Γ_{FCC} is the time required by an invocation of Procedure FindCriticalCandidate.*

Proof. Notice that Procedure FindCritical performs exactly one recursive invocation for each vertex of the tree $\mathcal{T}$ on the unique path between the root of $\mathcal{T}$ and u in $\mathcal{T}$. The claim follows since the height of $\mathcal{T}$ is $O(\log n)$. ◀

In the next subsection, we show that $\Gamma_{\mathsf{FCC}} = O(\log^2 n)$, and then we give our final result.

3.2 Procedure FindCriticalCandidate

In this subsection, we describe the core of the procedure that computes a critical edge for f. Let us first describe informally the main idea of this part. Let $b \in U_e$ and $c \in D_e$, and consider any two edges $f = (v, u), g = (x, y) \in S(e)$ such that b (resp. c) is on the unique path from x to v (resp. from y to u) in T (see Figure 2). It turns out that the stretch factor of *any* f w.r.t. a *given* g can be though as a linear function $\Phi_{b,c,g}(t) = \alpha_{b,c}(g) \cdot t + \beta_{b,c}(g)$, where $\alpha_{b,c}(g)$ and $\beta_{b,c}(g)$ only depend on g. More precisely, we will have that $\phi(f, g) = \Phi_{b,c,g}(t_{b,c}(f))$, for a suitable value $t_{b,c}(f)$ which only depends on f. Hence, whenever we look for a critical edge for f, we can ask for a corresponding function $\Phi_{b,c,g}(t)$ with maximum value on $t_{b,c}(f)$. Since we do not know a priori the edge f for which we need to compute a critical edge, we will maintain this information as the *upper envelope* of a suitable set of functions. Let us make this idea more precise.

▶ **Definition 11** (Upper Envelope). Let $\mathcal{F} = \{\Phi_1, \Phi_2, \ldots, \Phi_\ell\}$ be a finite set of functions, where $\Phi_i : \mathbb{R} \to \mathbb{R}$ for every $i = 1, 2, \ldots, \ell$. The upper envelope of $\mathcal{F}$ is defined as $\mathrm{UE}_{\mathcal{F}} : t \in \mathbb{R} \mapsto \arg\max_{\Phi \in \mathcal{F}} \Phi(t) \in 2^{\mathcal{F}}$.

Let $b \in U_e$ and $c \in D_e$. Given an edge $f = (v, u)$, define $t_{b,c}(f)$ as the quantity $d_T(b, v) + w(f) + d_T(u, c)$. Given an edge $g = (x, y)$, define $\alpha_{b,c}(g) = \frac{1}{w(g)}$ and $\beta_{b,c}(g) = \frac{d_T(x,b)+d_T(c,y)}{w(g)}$. Notice how, once b and c are fixed, $t_{b,c}(f)$ only depends on f while $\alpha_{b,c}(g)$ and $\beta_{b,c}(g)$ only depend on g. Let $\Phi_{b,c,g}(t) = \alpha_{b,c}(g) \cdot t + \beta_{b,c}(g)$.

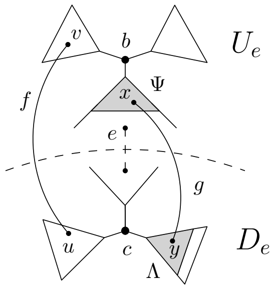


Figure 2 Illustration of Lemma 14. f is a swap edge for e, Ψ and Λ are two trees of the centroid decomposition, and b and c are their corresponding parent centroids. g is a potential (Ψ, Λ)-critical edge for f. Notice that the unique path from x to v (resp. from y to u) passes through b (resp. c).

▶ **Lemma 12.** *Let $f = (v, u) \in S(e, v)$. Let $b \in U_e$ and $c \in D_e$. Let X (resp. Y) be a set of vertices $x \in U_e$ (resp. $y \in D_e$) such that vertex b (resp. c) is on the unique path from x to v (resp. from y to u) in T. For every $g \in S(e, X, Y)$ we have $\phi(f, g) = \Phi_{b,c,g}(t_{b,c}(f))$.*

Proof. Let $g = (x, y)$. We have:

$$\begin{aligned}\phi(f,g) &= \frac{d_T(x,v) + w(f) + d_T(u,y)}{w(g)} = \frac{d_T(x,b) + d_T(b,v) + w(f) + d_T(u,c) + d_T(c,y)}{w(g)} \\ &= \frac{d_T(b,v) + w(f) + d_T(u,c)}{w(g)} + \frac{d_T(x,b) + d_T(c,y)}{w(g)} = \alpha_{b,c}(g) t_{b,c}(f) + \beta_{b,c}(g) \\ &= \Phi_{b,c,g}(t_{b,c}(f)).\end{aligned}$$ ◀

▶ **Definition 13** (Parent centroid). Let τ be a tree of the centroid decomposition $\mathcal{T}$. The *parent centroid* of τ is the centroid of the parent of τ in $\mathcal{T}$.

Lemma 12 is instrumental to proving the following (see Figure 2):

▶ **Lemma 14.** *Let $f = (v, u) \in S(e, v)$, and let Ψ, Λ be two trees of the centroid decomposition of T such that the following conditions hold: (i) $v \notin V(\Psi)$ or $V(\Psi) = \{v\}$, and (ii) $u \notin V(\Lambda)$ or $V(\Lambda) = \{u\}$. Let b (resp. c) be the parent centroid of Ψ (resp. Λ), and assume that $b \in U_e$ (resp. $c \in D_e$). Then, an edge g is a (Ψ, Λ)-critical edge for f if and only if $\Phi_{b,c,g} \in \mathrm{UE}_{\mathcal{F}}(t_{b,c}(f))$ where $\mathcal{F} = \{\Phi_{b,c,g'} : g' \in S(e, V(\Psi) \cap U_e, V(\Lambda) \cap D_e)\}$.*

Proof. First of all we show the following property of the centroid decomposition $\mathcal{T}$: let $p, q \in V(T)$, and suppose that the unique path in $\mathcal{T}$ between the leaf nodes associated with p and q contains a node whose corresponding centroid is z. Then, the unique path between p and q in T contains z. Indeed, if z is either p or q, the property is trivially true. On the other hand, suppose that $z \notin \{p, q\}$, and let τ be the subtree of T associated with z in $\mathcal{T}$. Then, let τ_z^i be the child subtree of τ containing p. Observe that q is not in τ_z^i. Moreover, by construction, each path from a node of τ_z^i, and in particular from p, to any node outside τ_z^i, and in particular to q, must pass through z.

We now prove the claim. If $V(\Psi) = \{v\}$ (resp. $V(\Lambda) = \{u\}$) then it follows from Lemma 12 by choosing $X = \{v\}$ and $Y = V(\Lambda) \cap D_e$ (resp. $X = V(\Psi) \cap U_e$ and $Y = \{u\}$). The complementary case is the one in which $v \notin V(\Psi)$ and $u \notin V(\Lambda)$. Consider the vertices v and b (resp. u and c) in $\mathcal{T}$ and notice that v (resp. u) cannot be an ancestor of b (resp. c). Indeed, if that were the case, then the subtree of T induced by the vertices in $V(\Psi)$ (resp. $V(\Lambda)$) would contain b (resp. c) contradicting the hypothesis. Hence, the path from any vertex in $V(\Psi)$ to v (resp. $V(\Lambda)$ to u) traverses b (resp. c) in $\mathcal{T}$ and therefore the same holds in T. The claim follows by invoking Lemma 12 with $X = V(\Psi) \cap U_e$ and $Y = V(\Lambda) \cap D_e$. ◀

Procedure FindCriticalCandidate($f = (v, u), \Psi, \Lambda$)

```
if $V(\Lambda) \cap D_e = \emptyset$ then return $\perp$;
if $V(\Psi) = \{v\}$ then return $\mathcal{Q}_e(f, \Psi, \Lambda)$;
$b \leftarrow$ Centroid of $\Psi$;
Let $j$ be the unique index in $\{0, 1, 2, 3\}$ such that $v \in V(\tau_b^j)$;
if $b \in D_e$ then return FindCriticalCandidate($f, \tau_b^j, \Lambda$);
$\mathcal{G} \leftarrow \{\mathcal{Q}_e(f, \tau_b^i, \Lambda) : i = 0, 1, 2, 3 \wedge i \neq j\}$;          // Here $b \in U_e$
$g_1 \leftarrow \arg\max_{g \in \mathcal{G}} \phi(f, g)$;
$g_2 \leftarrow$ FindCriticalCandidate($f, \tau_b^j, \Lambda$);
return $\arg\max_{g \in \{g_1, g_2\}} \{\phi(f, g)\}$;
```

Lemma 14 allows us to design a recursive procedure to compute a (Ψ, Λ)-critical edge for f (see Procedure `FindCriticalCandidate`). To this aim we will make use of a data structure $\mathcal{Q}_e$ that, for each edge $f \in S(e)$, and for each pair of trees Ψ, Λ of the centroid decomposition, can perform a query operation that we name $\mathcal{Q}_e(f, \Psi, \Lambda)$. This query reports an edge whose function $\Phi_{b,c,g}$ is in $\mathrm{UE}_{\mathcal{F}}(t_{b,c}(f))$ where b and c are the parent centroids of Ψ and Λ, respectively, and $\mathcal{F} = \{\Phi_{b,c,g'} : g' \in S(e, V(\Psi) \cap U_e, V(\Lambda) \cap D_e)\}$.

Next two lemmas show the correctness and the running time of the procedure:

▶ **Lemma 15.** *Let be given an edge $f = (v, u) \in S(e, v)$ and two trees Ψ, Λ of the centroid decomposition such that: (i) $v \in V(\Psi)$, and (ii) $u \notin V(\Lambda)$ or $V(\Lambda) = \{u\}$. Then, Procedure FindCriticalCandidate(f, Ψ, Λ) computes a (Ψ, Λ)-critical edge for f.*

Proof. First of all notice that if $V(\Lambda) \cap D_e = \emptyset$, then the algorithm correctly returns $\perp$. We now prove the claim by induction on $|V(\Psi)|$. If $|V(\Psi)| = 1$, then the only vertex in Ψ must be v and Procedure `FindCriticalCandidate` queries $\mathcal{Q}_e$ for $\mathcal{Q}_e(f, \Psi, \Lambda)$. By Lemma 14, the returned edge is a (Ψ, Λ)-critical edge for f. If $|V(\Psi)| > 1$ then we distinguish two cases, depending on the position of the centroid b of Ψ. If $b \in U_e$ it is sufficient to notice that a (Ψ, Λ)-critical edge for f must be incident to a tree τ_b^i for some $i = 0, 1, 2, 3$. Let j be the unique index in $\{0, 1, 2, 3\}$ such that $v \in V(\tau_b^j)$. If $j \neq i$ then, by Lemma 14, the query $\mathcal{Q}_e(f, \tau_b^i, \Lambda)$ returns a (τ_b^i, Λ)-critical edge g' (and hence g' is also a (Ψ, Λ)-critical edge) for f. Procedure `FindCritical` then returns either g' or another edge g such that $\phi(f, g) = \phi(f, g')$. If $j = i$, the algorithm is recursively invoked and, since $|V(|\tau_b^i|)| < |V(\Psi)|$ we know, by the induction hypothesis, that it returns a (τ_b^i, Λ)-critical edge for f, which is also (Ψ, Λ)-critical edge for f. If $b \in D_e$, then there is at most one τ_b^i containing at least one vertex in U_e (as the converse would imply that the vertices in $V(\Psi) \cap U_e$ are disconnected in Ψ, a contradiction). Moreover, since $v \in V(\Psi) \cap U_e$, there is exactly one such tree τ_b^i, namely τ_b^j. The algorithm recursively invokes itself on τ_b^j and we know, by induction hypothesis, that it returns a τ_b^j-critical edge for f, which is also (Ψ, Λ)-critical edge for f. ◀

▶ **Lemma 16.** *Procedure FindCriticalCandidate(f, Ψ, Λ) requires $O(\Gamma_{\mathcal{Q}_e} \cdot \log n)$ time, where $\Gamma_{\mathcal{Q}_e}$ is the time required by a query on $\mathcal{Q}_e$.*

Proof. Notice that Procedure `FindCriticalCandidate` performs exactly one recursive invocation for each vertex of the tree $\mathcal{T}$ on the unique path between the root of $\mathcal{T}$ and u in $\mathcal{T}$. The claim follows since the height of $\mathcal{T}$ is $O(\log n)$. ◀

Thus, to get the promised running time of $O(\log^2 n)$ for $\Gamma_{\mathtt{FCC}}$, we are left to prove that $\Gamma_{\mathcal{Q}_e} = O(\log n)$. Actually, such a bound can be obtained by suitably implementing $\mathcal{Q}_e$ in such a way that all the underlying upper envelope functions are efficiently maintained. Due to space limitation, this technical part will appear in the full version of the paper. By combining all the lemmas, and by observing that we need $O(n^2)$ space to handle the top-trees, and $O(m \log^2 n)$ space to implement each $\mathcal{Q}_e$, we eventually can give the following:

▶ **Theorem 17.** *The* ABSE-TS *problem can be solved in* $O(n^2 \log^4 n)$ *time and* $O(n^2 + m \log^2 n)$ *space.*

References

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In *Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511, 2007.

2 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

3 Davide Bilò, Feliciano Colella, Luciano Gualà, Stefano Leucci, and Guido Proietti. A faster computation of all the best swap edges of a tree spanner. In *Proc. of the 22nd Intl. Colloquium Structural Information and Communication Complexity*, pages 239–253, 2015. doi:10.1007/978-3-319-25258-2_17.

4 Davide Bilò, Feliciano Colella, Luciano Gualà, Stefano Leucci, and Guido Proietti. Effective edge-fault-tolerant single-source spanners via best (or good) swap edges. In *Proc. of the 24th Intl. Colloquium Structural Information and Communication Complexity*, in press.

5 Davide Bilò, Luciano Gualà, and Guido Proietti. Finding best swap edges minimizing the routing cost of a spanning tree. *Algorithmica*, 68(2):337–357, 2014. doi:10.1007/s00453-012-9674-y.

6 Davide Bilò, Luciano Gualà, and Guido Proietti. A faster computation of all the best swap edges of a shortest paths tree. *Algorithmica*, 73(3):547–570, 2015. doi:10.1007/s00453-014-9912-6.

7 Andreas Brandstädt, Victor Chepoi, and Feodor F. Dragan. Distance approximating trees for chordal and dually chordal graphs. *J. Algorithms*, 30(1):166–184, 1999. doi:10.1006/jagm.1998.0962.

8 Leizhen Cai and Derek G. Corneil. Tree spanners. *SIAM J. Discrete Math.*, 8(3):359–387, 1995. doi:10.1137/S0895480192237403.

9 Shantanu Das, Beat Gfeller, and Peter Widmayer. Computing all best swaps for minimum-stretch tree spanners. *J. Graph Algorithms Appl.*, 14(2):287–306, 2010.

10 Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM J. Comput.*, 38(5):1761–1781, 2008. doi:10.1137/060666202.

11 Harold N. Gabow. A scaling algorithm for weighted matching on general graphs. In *Proc. of the 26th Annual Symposium on Foundations of Computer Science*, pages 90–100, 1985. doi:10.1109/SFCS.1985.3.

12 Giuseppe F. Italiano and Rajiv Ramaswami. Maintaining spanning trees of small diameter. *Algorithmica*, 22(3):275–304, 1998. doi:10.1007/PL00009225.

13 Hiro Ito, Kazuo Iwama, Yasuo Okabe, and Takuya Yoshihiro. Polynomial-time computable backup tables for shortest-path routing. In *Proc. of the 10th Intl. Colloquium Structural Information and Communication Complexity*, pages 163–177, 2003.

14 Camille Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math*, 70(185):81, 1869.

15 Christian Liebchen and Gregor Wünsch. The zoo of tree spanner problems. *Discrete Applied Mathematics*, 156(5):569–587, 2008. doi:10.1016/j.dam.2007.07.001.

16 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Inf. Process. Lett.*, 79(2):81–85, 2001. doi:10.1016/S0020-0190(00)00175-7.

17 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Nearly linear time minimum spanning tree maintenance for transient node failures. *Algorithmica*, 40(2):119–132, 2004. doi:10.1007/s00453-004-1099-9.

18 Seth Pettie. Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time. In *Proc. of the 16th Intl. Symposium on Algorithms and Computation*, pages 964–973, 2005. doi:10.1007/11602613_96.

19 Guido Proietti. Dynamic maintenance versus swapping: An experimental study on shortest paths trees. In *Proc. of the 4th Intl. Workshop on Algorithm Engineering*, pages 207–217, 2000. doi:10.1007/3-540-44691-5_18.

20 Aleksej Di Salvo and Guido Proietti. Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. *Theor. Comput. Sci.*, 383(1):23–33, 2007. doi:10.1016/j.tcs.2007.03.046.

21 Robert Endre Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Inf. Process. Lett.*, 14(1):30–33, 1982. doi:10.1016/0020-0190(82)90137-5.

22 Bang Ye Wu, Chih-Yuan Hsiao, and Kun-Mao Chao. The swap edges of a multiple-sources routing tree. *Algorithmica*, 50(3):299–311, 2008. doi:10.1007/s00453-007-9080-z.

Decomposing a Graph into Shortest Paths with Bounded Eccentricity

Etienne Birmelé[1], Fabien de Montgolfier[2], Léo Planche[3], and Laurent Viennot[4]

1 **MAP5, UMR CNRS 8145, Univ. Sorbonne Paris Cité, France**
`etienne.birmele@parisdescartes.fr`

2 **IRIF, UMR CNRS 8243, Univ. Sorbonne Paris Cité, France**
`fm@liafa.univ-paris-diderot.fr`

3 **MAP5, UMR CNRS 8145, Univ. Sorbonne Paris Cité and IRIF, Paris, France**
`Leo.Planche@liafa.univ-paris-diderot.fr`

4 **INRIA and IRIF, Paris, France**
`Laurent.Viennot@inria.fr`

Abstract

We introduce the problem of hub-laminar decomposition which generalizes that of computing a shortest path with minimum eccentricity (MESP). Intuitively, it consists in decomposing a graph into several paths that collectively have small eccentricity and meet only near their extremities. The problem is related to computing an isometric cycle with minimum eccentricity (MEIC). It is also linked to DNA reconstitution in the context of metagenomics in biology. We show that a graph having such a decomposition with long enough paths can be decomposed in polynomial time with approximated guaranties on the parameters of the decomposition. Moreover, such a decomposition with few paths allows to compute a compact representation of distances with additive distortion. We also show that having an isometric cycle with small eccentricity is related to the possibility of embedding the graph in a cycle with low distortion.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph Decomposition, Graph Clustering, Distance Labeling, BFS, MESP

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.15

1 Introduction

The goal of this paper is to extend the MESP (Minimum Eccentricity Shortest Path) Problem from Dragan and Leitert [5] and the related problem of recognizing k-laminar graphs from Völkel *et al.* [16]. Both consist in finding a shortest path (in the sense that no path joining the same endpoints is shorter) k-dominating a graph (every vertex is at distance at most k from that path). The k-laminar problem additionally requires that path to be a diameter (there is no longer shortest path in the graph). Relationships between the two parameters are derived in [4].

To generalize this problem to more complex underlying structures, we introduce the problem of decomposing a graph into subgraphs with bounded shortest-path eccentricity. More precisely, we introduce the hub-laminar decomposition as a set of paths that k-dominates the graph and meet only near their extremities. To formalize this property, we introduce the notion of hub, that is a ball with fixed radius r centered at a path endpoint. The laminar associated to a path is the set of nodes k-dominated by the path. Our definition requires that an edge between two nodes belonging to two different laminars must also belong to a

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 15; pp. 15:1–15:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

hub. The degree of a hub is then the number of laminars that meet in the hub. The main result of the paper is that computing such a decomposition becomes tractable when hub centers are far enough one from another, or equivalently when paths are long enough. The MESP problem is equivalent to a hub-laminar decomposition with one laminar.

Such a generalization is naturally interesting in networks where one might want to identify a set of speedy linear routes that are "highly accessible" with applications in communication networks, transportation planning and water resource management. It is also motivated by DNA assembly in biology. DNA sequencing proceed through the reading of DNA fragments that must be assembled. When a single DNA strand is sequenced, comparison of fragments leads to a graph with "laminar" structure [16] that is with large diameter and small shortest path eccentricity. In the context of metagenomics, several DNA strands are sequenced together and more complex structures appear (see Figure 1 in [16]). Identifying the laminar structures of such graphs is typically encountered in metagenomic approaches for evolution questions (see e.g. [13]). The problem of the assembly (gluing DNA fragments to reconstruct a DNA strand) is then mixed with that of binning (sort DNA strands into groups that represent an individual genome or genomes from closely related organisms). See [14] for a presentation of assembly and binning problems in the context of metagenomics. Efficient decomposition of a graph into laminars could thus enhance the techniques for assembly and binning in this context.

The problem of decomposing a graph into λ laminars that k-cover the graph is not well defined as there may be several trade-offs of parameters λ and k. However, we show that when laminars are long enough compared to parameters r and k, then all (r, k)-hub-laminar decompositions are equivalent (same global structure) and have closely located hubs (except for hubs of degree two that do not affect the global structure). This implies for example that the positions of the extremities of the minimum eccentricity shortest path (MESP) can be approximated within $O(k)$ distance when the diameter of a graph is large with respect to the eccentricity k of the MESP.

From a graph perspective, a very natural generalization of MESP is the problem of finding a minimum eccentricity isometric cycle (MEIC), that is a cycle preserving distances that has minimum eccentricity k. Note that such a cycle can be seen as a hub-laminar decomposition with two laminars and two hubs with degree two. An important motivation for the MESP problem is its relationship with embedding a graph into the line with small multiplicative distortion [5]. We similarly show that the MEIC problem is related to embedding a graph into a circle with low multiplicative distortion, i.e. such that distances in the circle are within a constant factor of distances in the graph. Note that circle distortion is bounded by line distortion as a line segment can isometrically be embedded in a sufficiently long circle. (However, line distortion can be much larger than circle distortion.) Graph embedding in classical metrics is a well studied problem [9, 10]. Another related subject with abundant literature is that of compactly representing the distances of a graph [15, 12]. We show that a decomposition with few laminars ensures a compact representation of distances with bounded additive distortion.

Related works

Finding a MESP is NP-complete but can be approximated within a constant factor [5]. Better trade-off between computation time and approximation factor for MESP is obtained in [4]. The problem of efficiently representing the distances in a graph encompasses a vast literature dating from metric embedding [1]. Approximating embedding with low distortion is introduced in [2] where some results are provided in the case of the line. The case of embedding the metric induced by an unweighted graph is studied in [3]. Embedding a graph metric into the line with minimum distortion is NP-complete but fixed parameter tractable with respect to distortion [6]. Approximate distance oracles, i.e. compact data-structures for representing an approximation of distances, are investigated in [15]. A particular approach introduced by Peleg [12] resides in assigning a label to each node of a graph such that the distance between two nodes can be estimated from their labels. Several results exist about the trade-off between label size and approximation quality. Exact distance estimation is investigated in [8] and requires $\Omega(n)$ bits labels for general graphs. Approximation with a constant factor and sub-linear label size is derived in [15]. Some results concern additive approximation such as [7] in the case of hyperbolic graphs. A longest isometric cycle can be found in polynomial time [11].

2 Definitions

We consider finite, undirected and *connected* graphs (the connectivity is always assumed within the paper). Given a graph G, with vertex set $V(G)$ and edge set $E(G)$, we let $d_G(u,v)$ denote the *distance* between two vertices, i.e. the length of a shortest path from u to v. When the graph G is clear from the context, we omit the G subscript and simply write $d(u,v)$. Let $B(u,r) = \{v \in V(G) \mid d(u,v) \leq r\}$ denote the *ball* of radius r centered at u. Given a set of vertices U we set $B(U,r) = \cup_{u \in U} B(u,r)$. Given two sets U and W of vertices, we say that U *k-dominates* W when every vertex in W is at distance at most k from some vertex in U, i.e. $W \subseteq B(U,k)$. We say that U has *eccentricity* k, denoted $ecc(U) = k$, when k is the smallest integer such that $B(U,k) = V(G)$. A path P in G is a sequence of nodes such that any two consecutive nodes are linked by an edge of G. We consider only simple paths: a node appears at most once in the sequence. The first node of the sequence and the last one are called the *endpoints* of P. For the simplicity of notations, we also let P denote the set of nodes appearing in the sequence. For any vertices u and v on P, we denote by P_{uv} the subpath of P having u and v as endpoints.

2.1 Hub-laminar decomposition

▶ **Definition 1** (Hub-laminar decomposition)**.** Consider a connected undirected graph G, two positive integers r and k, $H = \{h_1, \ldots, h_q\}$ a set of vertices of G called *hub centers*, and $\mathcal{P} = \{P_1, \ldots, P_p\}$ a set of paths of G called *laminar paths*. A ball $B(h,r)$ with $h \in H$ is called a *hub*, and a set $B(P,k)$ with $P \in \mathcal{P}$ is called a *laminar*. $(H, \mathcal{P})$ is an (r,k)-*hub-laminar decomposition* of G if the following conditions are satisfied:

1. each laminar links two hubs centers: the endpoints h, h' of any $P \in \mathcal{P}$ belong to H and for every other hub $h'' \in H \setminus \{h, h'\}$, $B(P,k) \cap B(h'', r+1) = \emptyset$,
2. the laminars and the hubs dominate G: $V(G) \subseteq \bigcup_{h \in H} B(h,r) \cup \bigcup_{P \in \mathcal{P}} B(P,k)$,
3. each laminar path is locally a shortest path: any path $P \in \mathcal{P}$ with endpoints h and h' is a shortest path of the graph $G[B(P,k) \cup B(h,r) \cup B(h',r)]$,
4. laminars meet at hubs only: for all $i \neq j$ and $uv \in E(G)$ such that $u \in B(P_i,k)$ and $v \in B(P_j,k)$, there is a hub center $h \in H$ such that P_i and P_j both have h as endpoint and $u, v \in B(h,r)$.

The *minimal laminar length* of a decomposition $(H, \mathcal{P})$, denoted l, is the minimal length of the paths in $\mathcal{P}$. Its *laminar size*, denoted λ, is the number of paths in $\mathcal{P}$.

A hub-laminar decomposition $(H, \mathcal{P})$ with $l \geq 2r+1$ forms a partition of the edges of G in the following sense: each edge is either inside exactly one hub (possibly touching many

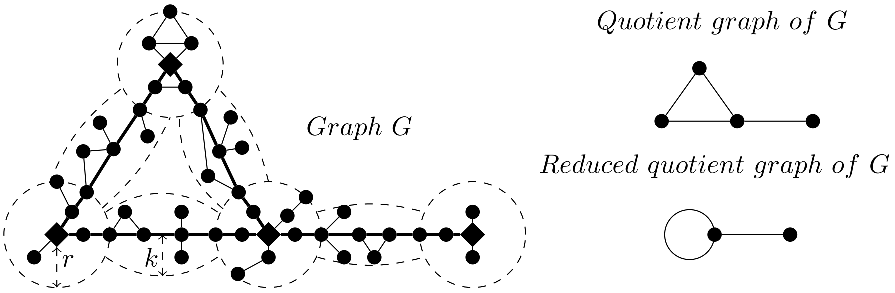


Figure 1 Illustration of an hub-laminar decomposition with $r = 2, k = 1$. Every vertex is at distance r from a hub center (diamond vertices) or at distance k from a laminar path (bold paths between hub centers).

laminars ending in that hub), i.e $\exists! h \in H$ s.t. $u, v \in B(h, r)$; or, else, inside a unique laminar (possibly touching one hub extremity of that laminar), i.e, $\exists! P \in \mathcal{P}$ s.t. $u, v \in B(P, k)$.

Figure 1 illustrates this definition and the notion of quotient graph that we define next. This definition basically defines a decomposition into k-neighborhoods of internally far apart shortest paths. It may seem a bit involved, but we think it expresses in a minimalist way what we mean by "internally far apart" with Item 4. Items 1 and 2 indicate that the graph is decomposed into laminars which are k-neighborhoods of certain paths and hubs which are balls centered at the extremities of those paths. Item 3 requires path to be shortest in the induced graph (rather than in G), to allow laminars with different length.

2.2 Quotient graph and equivalence between decompositions

As previously mentioned, the hub-laminar decomposition gives naturally raise to a skeleton, which can be simplified into a *quotient graph*.

▶ **Definition 2** (quotient graph and reduced quotient). Given a graph G and an (r, k)-hub-laminar decomposition $(H, \mathcal{P})$ of G, the *quotient* of this decomposition is an edge-labeled multigraph with vertex-set H and for each $P \in \mathcal{P}$ with endpoints h, h' there is an edge hh' whose label is the length of P.

The *degree of a hub* denotes the degree of the corresponding vertex in the quotient graph, or equivalently the number of laminar paths its center is the endpoint of.

The *reduced quotient graph* of a decomposition $(H, \mathcal{P})$ is the multigraph obtained from its quotient graph by repeatedly removing degree 2 nodes: for every vertex u of the quotient incident with exactly two edges uv and uw with respective labels a and b, u and both edges are removed and a new edge vw is added with label $a + b$. (It is a loop when $v = w$.)

When the quotient is not a cycle (a case specifically adressed by MEIC, see Section 3) the reduced quotient is well defined and unique (recall that graphs are supposed connected).

▶ **Definition 3** (equivalence between decompositions). Two hub-laminar decomposition of a given graph G, possibly with different parameters r, k, are *D-equivalent* if they have the same reduced quotient graph, up to an isomorphism ϕ of vertex-sets such that $d(h, \phi(h)) \le D$ (d is the distance between hub centers in G, not in the reduced quotient).

2.3 Isometric cycle, circle embedding and distance labeling

A cycle C in a graph G is *isometric* if it preserves distances, i.e. $d_C(u,v) = d(u,v)$ for all $u, v \in V(C)$. In other words, for any pair u, v of nodes on the cycle, one of the two path linking u and v in the cycle is a shortest path in the graph. Note that an isometric cycle is necessarily an induced cycle. The MEIC problem consists in finding an isometric cycle with minimum eccentricity. It can be shown to be NP-complete following a similar proof as [5] for the NP-completeness of MESP problem.

A *circle embedding* of a graph G is a mapping $f : V(G) \to C$ where C is a circle of given length c. It has distortion γ if $d(u,v) \leq d_C(f(u), f(v)) \leq \gamma d(u,v)$ for all u, v in $V(G)$. The *circle distortion* $cd(G)$ of G is the minimum distortion of a circle embedding of G.

A distance labeling of a graph G consists in assigning a label L_u to each node $u \in V(G)$ together with a distance estimation function f that outputs an estimation of $d(u,v)$ when given L_u and L_v as input. It has additive distortion α if $d(u,v) \leq f(L_u, L_v) \leq d(u,v) + \alpha$ for all u, v in G.

3 Main results

Obviously, the reduced quotient graph of a graph having a (r,k)-hub-laminar decomposition follows the following trichotomy: it is either a path, a cycle or has a degree three node. We treat separately the three cases.

In the first case, the graph has a shortest path with eccentricity $\max\{3k, 2r\}$ and can be recognized through an approximate MESP algorithm such as [4]. (The $\max\{3k, 2r\}$ bound is a consequence of Lemma 12 given in Section 4.) In the second case, the graph has an isometric cycle with eccentricity at most $\max\{3k, 2r\}$. To recognize such graphs, we propose an approximate MEIC algorithm:

▶ **Theorem 4.** *Given a graph containing a K-dominating isometric cycle with length ℓ, a $6K$-dominating isometric cycle can be computed in $O(n^{4.752}\log(n))$ time. Moreover, the computed cycle is indeed $3K$-dominating when $\ell \geq 12K + 2$.*

We obtain therefore an algorithm for approximating circle embedding with low distortion.

▶ **Proposition 5.** *If a graph has circle distortion γ, it is possible to embed it in a circle with distortion $O(\gamma^2)$ in polynomial time.*

Recognizing the general case of decomposition is not a well defined problem as several decompositions may yield different trade-offs of the parameters. However, when laminars are long enough, all (r,k)-hub-laminar decompositions are indeed $O(k)$ equivalent. This can be seen as a consequence of the following recognition result.

▶ **Theorem 6.** *Given a graph G having a (r,k)-hub-laminar decomposition $(H, \mathcal{P})$ of minimal laminar length $\ell \geq 8r + 60k + 4$ and integers K, R such that $K \geq 3k$, $R \geq 2K + 3r + 3k$ and $2R + 8K < \ell - 2r - 18k - 4$, it is possible to compute in $O(\min(n,\lambda)m)$ time a (K,R)-hub-laminar decomposition which is $(K + 2r)$-equivalent to $(H, \mathcal{P})$.*

From the graph metric point of view, we obtain then a compact representation of distances:

▶ **Proposition 7.** *Given a graph G having an (r,k)-hub-laminar decomposition with laminar size λ, it is possible to compute in polynomial time a $O(\max\{k, r\})$-additive distance labeling with $O(\lambda \log n)$ bit labels.*

Due to lack of space, the proofs of these theorems, and of the lemmas and propositions stated below, are put in Appendix.

4 Algorithms

4.1 Minimum Eccentricity Isometric Cycle

We propose to approximate the MEIC problem by computing a longest isometric cycle, that is an isometric cycle of G with maximum length. The following lemma shows that a longest isometric cycle $O(k)$-dominates any k-dominating isometric cycle.

▶ **Lemma 8.** *Let G be a graph with an isometric cycle $C = c_1, ...c_p$ k-dominating G, and let D be a longest isometric cycle of G. Every vertex of C is at distance at most $5k$ of D. Furthermore, if D has length more than $12k + 2$ then every vertex of C is at distance at most $2k$ of D.*

Consequently, a longest isometric cycle in a graph is a 6-approximation for the MEIC problem, and a 3-approximation when the graph has a diameter large enough. As shown in [11], a longest isometric cycle can be computed in $\mathcal{O}(n^{4.752} \log(n))$ time. Theorem 4 is thus a direct consequence of this and Lemma 8.

4.2 General case outline

Consider a graph G having a (r, k) hub-laminar decomposition $(H, \mathcal{P})$ of minimal laminar length ℓ and having at least one hub of degree at least 3. The underlying idea of the algorithm is to use BFS (Breadth-first search) to compute shortest paths and their K-neighborhoods, K being chosen large enough to dominate every laminar traversed by the considered shortest paths, but small enough compared to ℓ to detect all hubs of degree at least 3.

The first step, called $FindHubs$, consists in applying the procedure $NextHub$ described in section 4.4 until it discovers no new hubs. This step yields two sets of hub-centers A and B, respectively called *unmovable* and *movable* hub-centers, which will be used to determine the laminars. An unmovable hub center $a \in A$ corresponds to exactly one hub center $h \in H$ such that $d(a, h)$ is bounded. It will be shown that A contains exactly one such vertex for every hub-center of H which degree is not 2.

A movable hub center $b \in B$ will only be added by $NextHub$ in a configuration corresponding to a cycle in the quotient graph of $(H, \mathcal{P})$ containing only one hub of degree at least 3, like the three laminars on the left of Figure 1. This is called a *Problematic Configuration* We then know there exists at least a degree 2 hub $h \in H$ somewhere in that cycle, but if they are thin enough they may remain merged in the laminars and we can not bound $d(b, h)$, and b may be moved in the second step described bellow.

The laminars are determined in a second step by the $FindLaminars$ procedure, which links the hub-centers of the previous step by shortest paths. The only difficulty which has to be taken into account refers to hubs of degree 2 in $(H, \mathcal{P})$. Indeed, the BFS runs of the hub-detection step may have missed one of them because they K-dominated it, whereas the BFSs of second step don't. In that case, the set of hubs A is adapted by adding the new discovered hub, and if needed, the corresponding movable hub center is deleted from B.

Figure 2 gives a summary of the two steps by showing a possible outcome of the $FindHubs$ and $FindLaminars$ on an example. The $FindHubs$ procedure detects all hubs of degree different from 2 and some of those of degree 2. Moreover, it places a movable hub on each problematic configuration. $FindLaminars$ then computes the corresponding laminars, adding new hubs if a hub of degree 2 missed in the first step is detected. Some of them may however still be undetected, being replaced by a movable hub or just missing in the final

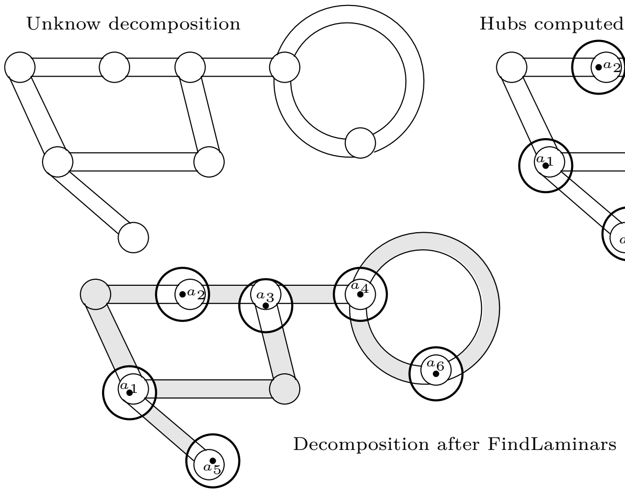


Figure 2 Illustration of the different steps of the algorithm. The $(H, \mathcal{P})$ decomposition is unknown (top left). Notice a Problematic Case on the right of the graph: a cycle of laminar with only one degree not 2 hub. First, hub centers are computed such that every hub $B(h, r), h \in H$ with degree different from 2 is covered by $B(a_i, R), a_i \in A$ (top right). Finally the laminars are computed (greyed, bottom) and some movable hubs may be moved (like b_1 moved into a_5). Some thin degree 2 hubs from H are not found but merged in the K-laminars. Fortunately the hub center a_5 was found by BFS in the second step, but we could also have output b_1 instead, or both a_5 and b_1, yielding in any case an equivalent reduced quotient.

decomposition. The quotient graphs of the decomposition supposed by Theorem 6 and that of the constructed decomposition may therefore be different, but their reduced quotients are equivalent.

4.3 Some rules to compute a decomposition

We rely on the following properties for running our algorithm. The first tool is used to identify hub centers. Fortunately there is a pattern that, when it occurs, signals that any hub-laminar decomposition must have a hub nearby:

▶ **Lemma 9** (Hub trigger). *Consider three numbers r, k and $K \geq 3k$. If there exists*

- *a shortest path Q from a to b*
- *a vertex $u \in V(Q)$ such that $d(a, u) > K + 6k$ and $d(b, u) > K + 6k$*
- *a vertex v such that $d(u, v) = K$*
- *a vertex w such that $vw \in E(G)$ and $d(Q, w) = K + 1$ and $d(u, w) = K + 1$,*

then any (r, k)-hub-laminar decomposition $(H, \mathcal{P})$ has a hub center $h \in H$ with $d(u, h) \leq K + r$.

Fig. 3 (right) illustrates this. This pattern, when found, allows to propose u as a hub. And it is very powerful, since every hub h of degree at least three of any (r, k)-hub-laminar decomposition shall trigger this pattern, for any shortest path Q passing close to h and long enough, as stated by the following lemma:

▶ **Lemma 10** (Degree ≥ 3 Hub Detection). *Consider a graph admitting an (r, k)-hub-laminar decomposition $(H, \mathcal{P})$, and having a hub $h \in H$ whose degree is at least 3, and consider $K \geq 3k$. For any shortest path Q and vertex $u \in V(Q)$ such that*

- *$d(u, h) \leq K$,*
- *u is at distance at most $r + 4K + 9k + 2$ of both endpoints of Q,*

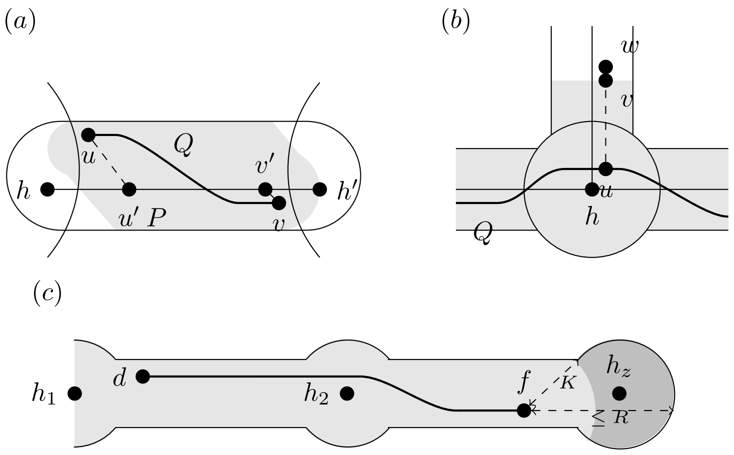


Figure 3 Illustration of the structural properties. (a) Lemma 12: The part of the laminar k-covered by $P_{u'v'}$ is K-covered by Q. (b) Lemma 9: If Q goes through a hub of degree ≥ 3, a triplet u, v, w can be found, and it is the only such case far apart the extremities of Q. (c) Lemma 11.

there exists

- *$x \in V(Q)$ such that x is at distance at least $K + 6k$ from both endpoints of Q, and*
- *$vw \in E(G)$ such that $d(x, v) = K$ and $d(Q, w) = K + 1$*

Notice that if a graph admits an (r, k)-hub-laminar decomposition $(H, \mathcal{P})$ where all hubs have degree at least three, then the pattern is enough to find all its hubs, or more exactly to compute a set of hubs H' which is in bijection ϕ with H, i.e. for all $h \in H$ $d(h, \phi(h)) \leq K + r$. Of course to do so in polynomial time we shall use a clever collection of paths Q to trigger all hubs. This is the idea developed in the algorithm. But before we shall explain how to deal with degree 1 hubs, the "dead end" hubs.

▶ **Lemma 11** (Dead-end hub). *Consider the graph G' induced by a sequence of incident hubs and laminar $H_1, L_1, H_2, ...H_z$, such that h_1 and h_z are at distance at least $2R + r + 2$. Suppose moreover that H_z is a hub of degree 1 and all other hubs but H_1 are of degree 2.*

Let d in L_1 be at distance at most $R + r$ of h_1 and f a vertex of G' the furthest from d. f is then at distance at most $2r + 2k$ from h_z.

This lemma allow to approximate h with f. As we have just seen, hubs with degree different from 2 are, in some sense, uniquely defined (up to a certain distance) in any hub-laminar decomposition of given parameters. Degree 2 hubs however may be added at discretion on any hub-laminar decomposition, in the middle of long laminar, so we cannot imagine a sufficient condition for detecting them. However, they are necessary in very few case, namely

- to dominate a vertex at distance more than k, but less than r, inside of a r-laminar (not k-laminar)
- for the Problematic Configuration, since a laminar must have two distinct extremities

The last property, proved in [4], deals no more with computing hub centers but with computing laminars. While it is is NP-hard to find a shortest path that k-dominates a k-laminar graph, any path $3k$-dominates a section of the laminar between its extremities. Stated more formally:

▶ **Lemma 12** (Path local dominating). *Consider a shortest path P (say, from h to h'). Let Q be a path from u to v contained in $B(P,k)$.*

Assume there exists $u' \in \mathcal{P}$ and $v' \in P$ such that $d(u,u') \leq k$ and $d(v,v') \leq k$.

Then every vertex of $P_{u'v'}$ is at distance at most $2k$ from Q.

Furthermore, every vertex of $B(P_{u'v'},k)$ is at distance at most $3k$ of Q.

Fig. 3 (left) illustrates this. We extensively use this lemma for designing an approximation algorithm: P is any laminar path, and Q is chosen to $3k$ dominate the middle of the laminar of P, i.e. all vertices far enough from P extremities (Lemma13 and 14 define "far enough" as $2R + 8K + 2r + 18k + 4$) and we therefore get a decomposition into $3k$-laminar graphs.

4.4 Finding hubs

In the following section, graphs are vertex-colored, with possibly some uncolored vertices.

The general idea of the algorithm is that starting with an uncolored graph, we end with a fully colored one, such that :

- Every vertex at distance less than $K + 1$ of a computed hub a is colored with color a.
- Other vertices of the graph are colored with color *lam*.

4.4.1 The StopBFS function

The **StopBFS** procedure, provided a vertex d and a color c, consists in running an usual Breadth-first search, starting at vertex d, with the following additional rules:

- only uncolored vertices are put in the BFS queue
- the BFS stops immediately if a vertex f is visited (i.e. extracted from BFS queue) and f has a colored neighbor whose color is not c.
- otherwise, if the BFS stops because its queue is empty, let f be the last visited vertex
- function $StopBFS(d,c)$ returns the BFS path P from d to f (which is a shortest path in the graph induced by G after removing c-colored vertices).

4.4.2 Finding a new hub: NextHub

Given a vertex s, typically corresponding to an already selected hub center, the **NextHub** procedure (see pseudo-code in Algorithm 1) detects new hubs: it colors $B(s,R)$ with a new color and runs a StopBFS procedure from its border. In the case of a not deep-enough tree, the discovered vertices are colored to not be reused during the hub discovery. Otherwise, it may either find a new hub of degree at least 3 by Lemma 10, find a new hub of degree 1 by Lemma 11, meet another hub and dominate a laminar by Lemma 12 or cycle and come back to hit $B(s,R)$. The later case indicates that the algorithm encountered the problematic configuration and induces the creation of a movable hub.

Given a path P, $r3K(P)$ denotes the subpath of P obtained by removing the $3K$ first and $3K$ last vertices of P. In the sequel, sets A and B respectively denote the unmovable and movable hub centers.

4.4.3 Finding all hubs : FindHub

The $FindHub$ simply consists in considering the initial uncolored graph G and to construct the sets A and B of unmovable and movable hub-centers by repeatedly applying $NextHub$ (see pseudo-code in Appendix).

Algorithm 1: NextHub

```
NextHub
    Input: A graph $G$ with possibly colored vertices, integers $R$ and $K$, hub-center
           sets $A$ and $B$, and a vertex $s$
    Output: Updated sets $A$, $B$ and vertex coloring
Color every vertex in $B(s, R)$ with a new color $col(s)$
Choose an uncolored vertex $d$ at distance $R + 1$ from $s$
Let $P = stopBFS(d, col(s))$ and $f$ the last vertex of $P$
If $P$ is of length less that $2R + 4K + 2$ then
    /* Not deep enough tree: no laminar is crossed                */
    Color all vertices visited by $stopBFS(d, col(s))$ with color $lam$
else if $\exists w, a$ s.t. $col(w) \neq col(s)$ and $h \in r3K(P)$ and $d(w, a) = K+1$ and
  $d(w, P) = K+1$ then
    /* A hub has been detected by Lemma 9 configuration            */
    Add to $A$ the first vertex $a$ of $r3K(P)$ satisfying the above
else if $f$ is at distance less than $2K$ of $B(s, R)$ then
    /* $P$ is deep, found no hub and came back near the root:
       problematic configuration                                   */
    Add to $B$ the vertex $b$ in the middle of $P$
    Color uncolored vertices in $B_{G\setminus\{B(d,R)\cup B(f,R)\}}(P, K)$ with color $lam$
else if $f$ is not adjacent to a colored vertex then
    /* $P$ is long, found no hub and doesn't come back: dead end    */
    Add $f$ to $A$
else
    /* $P$ links $B(s, R)$ to a vertex of a color different from $col(s)$.
       The dominated vertices correspond to a laminar.             */
    Color uncolored vertices in $B_{G\setminus\{B(d,R)\cup B(f,R)\}}(P, K)$ with color $lam$
```

For the first call, we first compute a long path Q using a double BFS. More precisely, starting at any s_0, we compute a furthest node s and then repeatedly apply $NextHub$ until a vertex a is added to A. If there is a unique hub of degree at least three, the fact that $\ell > 2R + 8K + 2r + 18k + 4$ ensures that the deepest vertex of any BFS is at distance greater than $R + (r + 4K + 9k + 2)$ of the hub center. If there are at least two hubs of degree at least three, $\ell > 2R + 8K + 2r + 18k + 4$ implies that any vertex is at distance greater than $\frac{\ell}{2} > R + (r + 4K + 9k + 2)$ of one of the two hub-centers. In any case, the $Next\_Hub$ function applied to s has to find the configuration from Lemma 9 at some point, ensuring that a first hub center $a \in A$ is found. We set $A = \{a\}$ and $B = \emptyset$, and uncolor the whole graph.

Once this first vertex of A has been found, $NextHub$ is run while there exists a hub center $a \in A$ having an uncolored vertex in its $R + 1$-neighborhood. If ℓ is large, the $FindHubs$ procedure finds all hubs of $(H, \mathcal{P})$ up to those of degree 2, as stated in the following lemma.

▶ **Lemma 13.** *Suppose that $(H, \mathcal{P})$ has at least a hub of degree 3, and $\ell(H, \mathcal{P}) > 2R + 8K + 2r + 18k + 4$. Then, for every vertex $a \in A$, there exists a vertex $h \in H$ such that their distance is at most $K + 2r$. Conversely, for every $h \in H$ of degree different from 2, such a vertex a is selected in A.*

4.5 Finding laminars

At this step we have a set of unmovable hubs including all hubs of degree 1 or at least 3, and potentially those of degree 2. Moreover, the set B of movable hub-centers indicates the places where problematic configuration occur. We have to identify the laminars and their paths, keeping in mind that some new hubs of degree 2 may be detected. Each path is found by a BFS starting at an hub center and ending at the first other hub center encountered.Then we remove from the graph the vertices from the laminar, but not the hubs. For each path P linking two hub centers h and h', the vertices from $B(P,k)-(B(h,R)\cup B(h',R)$ are removed from the graph. Hub center h is no more used when $B(h,R)$ becomes disconnected and the whole process ends when the graph consists in disconnected hubs only.

To prevent any difficulty arising from ending a shortest path with a movable hub $B(b,R)$, we start by those hubs to run BFSs. Indeed, such hubs correspond to a configuration where the quotient of the decomposition (the one supposed by Theorem 6 and the computed one, since they have the same reduced quotient) contains a cycle . If a movable hub has been used, it means that only one hub center $a\in A$ corresponding to hub-center $h\in H$ of degree at least 3 has been found, and that all other hubs are of degree 2 on the cycle and have been missed. Starting from b, the first element of $A\cup B$ which is hit is then a, whatever direction was followed from $B(b,R)$. Thus, two BFS from b to a are run and follow the ring in opposite directions. Either the two obtained paths K-dominate all vertices of the ring, in which case b is transferred to A and the two paths added to $\mathcal{Q}$; Or there exist a vertex in the ring which is not K-dominated. This vertex is then at distance at most $K+2r$ of some $h\in H$ (cf Appendix for a proof). It is thus added to A and b is deleted from B.

Once the movable centers have been considered, no other places with problematic configurations are left. One therefore just has to draw shortest paths between vertices of A, and Lemma 12 ensures that they cover the laminars of $(H,\mathcal{P})$. The only difficulty is again that a hub of degree 2 that had not been discovered by $FindHubs$ may this time be discovered by $FindLaminars$ because Lemma 9 configuration is encountered. In that case, this degree 2 hub center is added to A and a new BFS is run from it. See pseudo-code of $FindLaminars$ in Appendix.

▶ **Lemma 14.** *Suppose that $(H,\mathcal{P})$ has at least a hub of degree 3, and $\ell(H,\mathcal{P})>2R+8K+2r+18k+4$. Suppose that $FindLaminars$ is run on sets A and B returned by $FindHubs$. Then it ends with every vertex deleted or marked as undeletable.*

As shown in the appendix, Lemmas 13 and 14 imply Theorem 6.

5 Embedding and distance labeling

5.1 Circle embedding with bounded distortion

Proposition 5, stated in Section 3, is a consequence of Theorem 4 and the two following propositions.

▶ **Proposition 15.** *Any graph G having a circle embedding with distortion γ has a shortest path or an isometric cycle with eccentricity $\lfloor\gamma/2\rfloor$ at most.*

▶ **Proposition 16.** *Given a graph G and an isometric cycle with eccentricity k in G, an embedding of G in a circle with distortion $O(k\cdot cd(G))$ can be computed in polynomial time.*

Proof sketch. The construction of the embedding is similar to that of [5] with Euler tours of trees of depth k rooted in the cycle (see [5]). We then obtain an embedding of the graph in a cycle of length $2n$ at most that can be easily embedded in a circle with same length. The distortion of an edge uv of G is then at most twice the size of the union S of trees rooted on the shortest path of the cycle from the root u' of the tree of u to the root v' of the tree of v. As we have $d(u', v') \leq 2k + 1$, the diameter of S is at most $4k + 1$. Now consider an embedding of G in a circle C with distortion $\gamma = cd(G)$. Two nodes of S are embedded at distance at most $\gamma(4k + 1)$ in the circle and different nodes are at distance 1 at least. We thus have $|S| \leq \gamma(8k + 2)$, and our embedding has distortion $O(\gamma k)$. ◀

5.2 Distance labeling for general hub-laminar decomposition

A hub-laminar decomposition of a graph G allows to compute a compact representation of distances in G with additive distortion. A distance labeling is said to be c-additive and have s bit labels when the label L_u assigned to a node u contains at most s bits and for all pairs of nodes u, v, a distance estimation $\widehat{d_{uv}}$ can be computed from L_u and L_v such that $d(u, v) \leq \widehat{d_{uv}} \leq d(u, v) + c$. Proposition 7 is a consequence of Theorem 6 and the following proposition.

▶ **Proposition 17.** *Given a (r, k)-hub-laminar decomposition $(H, \mathcal{P})$ with λ laminars of a graph G, a $\max(4k, 2r)$-additive distance labeling with $O(\lambda \log n)$ bit labels can be computed in polynomial time.*

Acknowledgments. The authors thank Michel Habib for inspiring discussions about k-laminar graphs, and Eric Bapteste, Philippe Lopez and Chloé Vigliotti for raising the problem of identifying complex laminar structures in biological graphs.

References

1 Patrice Assouad. étude d'une dimension métrique liée à la possibilité de plongements dans $\mathbf{R}^n$. *C. R. Acad. Sci. Paris Sér. A-B*, 288(15):A731–A734, 1979.

2 Mihai Badoiu, Julia Chuzhoy, Piotr Indyk, and Anastasios Sidiropoulos. Low-distortion embeddings of general metrics into the line. In *SToC 2005*, pages 225–233. ACM, 2005. doi:10.1145/1060590.1060624.

3 Mihai Badoiu, Kedar Dhamdhere, Anupam Gupta, Yuri Rabinovich, Harald Räcke, R. Ravi, and Anastasios Sidiropoulos. Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In *SODA 2005*, pages 119–128. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070449.

4 Etienne Birmelé, Fabien de Montgolfier, and Léo Planche. Minimum eccentricity shortest path problem: An approximation algorithm and relation with the k-laminarity problem. In *Combinatorial Optimization and Applications - 10th International Conference, COCOA 2016, Hong Kong, China, December 16-18, 2016, Proceedings*, pages 216–229, 2016. doi:10.1007/978-3-319-48749-6_16.

5 Feodor F. Dragan and Arne Leitert. On the minimum eccentricity shortest path problem. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 276–288, 2015. doi:10.1007/978-3-319-21840-3_23.

6 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Elena Losievskaja, Frances A. Rosamond, and Saket Saurabh. Distortion is fixed parameter tractable. *TOCT*, 5(4):16:1–16:20, 2013. doi:10.1145/2489789.

7 Cyril Gavoille and Olivier Ly. Distance labeling in hyperbolic graphs. In *ISAAC 2005*, volume 3827 of *Lecture Notes in Computer Science*, pages 1071–1079. Springer, 2005. doi:10.1007/11602613_106.

8 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004. doi:10.1016/j.jalgor.2004.05.002.

9 Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *FOCS 2001*, pages 10–33. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959878.

10 Piotr Indyk and Jiří Matoušek. Low-distortion embeddings of finite metric spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 177–196. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.ch8.

11 Daniel Lokshtanov. Finding the longest isometric cycle in a graph. *Discrete Applied Mathematics*, 12(157):2670–2674, 2009.

12 David Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000. doi:10.1002/(SICI)1097-0118(200003)33:3<167::AID-JGT7>3.0.CO;2-5.

13 Jimmy H et al. Saw. Exploring microbial dark matter to resolve the deep archaeal ancestry of eukaryotes. *Phil. Trans. R. Soc. B*, 370(1678):20140328, 2015.

14 Torsten Thomas, Jack Gilbert, and Folker Meyer. Metagenomics-a guide from sampling to data analysis. *Microbial informatics and experimentation*, 2(1):3, 2012.

15 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

16 Finn Völkel, Eric Bapteste, Michel Habib, Philippe Lopez, and Chloe Vigliotti. Read networks and k-laminar graphs. arXiv:1603.01179, Mar 2016. URL: https://hal.inria.fr/hal-01282715.

Independent Feedback Vertex Set for P_5-free Graphs[*][†]

Marthe Bonamy[1], Konrad K. Dabrowski[2], Carl Feghali[3], Matthew Johnson[4], and Daniël Paulusma[5]

1 CNRS, LaBRI, Université de Bordeaux, France,
marthe.bonamy@u-bordeaux.fr
2 Durham University, Durham, UK,
konrad.dabrowski@durham.ac.uk
3 IRIF & Université Paris Diderot, France,
feghali@irif.fr
4 Durham University, Durham, UK,
matthew.johnson2@durham.ac.uk
5 Durham University, Durham, UK,
daniel.paulusma@durham.ac.uk

Abstract

The NP-complete problem Feedback Vertex Set is to decide if it is possible, for a given integer $k \geq 0$, to delete at most k vertices from a given graph so that what remains is a forest. The variant in which the deleted vertices must form an independent set is called Independent Feedback Vertex Set and is also NP-complete. In fact, even deciding if an independent feedback vertex set exists is NP-complete and this problem is closely related to the 3-Colouring problem, or equivalently, to the problem of deciding if a graph has an independent odd cycle transversal, that is, an independent set of vertices whose deletion makes the graph bipartite. We initiate a systematic study of the complexity of Independent Feedback Vertex Set for H-free graphs. We prove that it is NP-complete if H contains a claw or cycle. Tamura, Ito and Zhou proved that it is polynomial-time solvable for P_4-free graphs. We show that it remains in P for P_5-free graphs. We prove analogous results for the Independent Odd Cycle Transversal problem, which asks if a graph has an independent odd cycle transversal of size at most k for a given integer $k \geq 0$.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases feedback vertex set, odd cycle transversal, independent set, H-free graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.16

1 Introduction

Many computational problems in the theory and application of graphs can be formulated as modification problems: from a graph G, some other graph H with a desired property must be obtained using certain permitted operations. The number of graph operations used (or some other measure of cost) must be minimised. The computational complexity of a graph modification problem depends on the desired property, the operations allowed and the possible inputs; that is, we can prescribe the class of graphs to which G must belong.

[*] This paper received support from EPSRC (EP/K025090/1), London Mathematical Society (41536), the Leverhulme Trust (RPG-2016-258) and Fondation Sciences Mathématiques de Paris.
[†] A full version of the paper is available at https://arxiv.org/abs/1707.09402.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 16; pp. 16:1–16:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A set S of vertices in a graph G is a *feedback vertex set* of G if removing the vertices of S results in an acyclic graph, that is, the graph $G - S$ is a forest. The Feedback Vertex Set problem asks if a graph has a feedback vertex set of size at most k for some integer $k \geq 0$ and is a well-known example of a graph modification problem: the desired property is that the obtained graph is acyclic and the permitted operation is vertex deletion. The directed variant was one of the original problems proven to be NP-complete by Karp. The proof of this implies NP-completeness of the undirected version even for graphs of maximum degree 4.

In this paper, we consider the problem where we require the feedback vertex set to be an independent set. We call such a set an *independent feedback vertex set.*

Independent Feedback Vertex Set
Instance: a graph G and an integer $k \geq 0$.
Question: does G have an independent feedback vertex set of size at most k?

Many other graph problems have variants with an additional constraint that a set of vertices must be independent. For example, see [6] for a survey on Independent Dominating Set, and [10] for Independent Odd Cycle Transversal, also known as Stable Bipartization. We survey results on Independent Feedback Vertex Set below.

Not every graph admits an independent feedback vertex set (consider complete graphs on at least four vertices). Graphs that do admit an independent feedback vertex set are said to be *near-bipartite*, and we can ask about recognising these graphs.

Near-Bipartiteness
Instance: a graph G.
Question: is G near-bipartite (that is, does G have an independent feedback vertex set)?

Near-Bipartiteness is NP-complete even for graphs of maximum degree 4 [15] or diameter 3 [3]. Hence, by setting $k = n$, we find that Independent Feedback Vertex Set is NP-complete for these two graph classes. The Independent Feedback Vertex Set problem is even NP-complete for planar bipartite graphs of maximum degree 4 (see [14]). As bipartite graphs are near-bipartite, this result shows that there are classes of graphs where Independent Feedback Vertex Set is harder than Near-Bipartiteness. To obtain tractability results for Independent Feedback Vertex Set, we need to make some further assumptions.

One way is to consider the problem from a parameterized point of view; see [1, 11] for FPT algorithms for Independent Feedback Vertex Set when parameterized by k. Another way to obtain tractability results is to restrict the input to special graph classes in order to determine graph properties that make the problem polynomial-time solvable. Tamura et al. [14] showed that Independent Feedback Vertex Set is polynomial-time solvable for chordal graphs, graphs of bounded treewidth and for cographs. The latter graphs are also known as P_4-free graphs (P_r denotes the path on r vertices and a graph is *H-free* if it has no induced subgraph isomorphic to H), and this strengthened a result of Brandstädt et al. [4], who proved that Near-Bipartiteness is in P for P_4-free graphs.

Our Contribution. The Independent Feedback Vertex Set problem is equivalent to asking for a (proper) 3-colouring of a graph such that one colour class has at most k vertices and the union of the other two induces a forest. We wish to compare the behaviour of Independent Feedback Vertex Set with that of 3-Colouring, which we observe is equivalent to the problem of deciding if a graph has an *independent odd cycle transversal*, that is, a set of vertices whose deletion makes the graph bipartite. However, so far very few graph classes are known for which Independent Feedback Vertex Set is tractable and our goal is to find more of them. For this purpose, we consider H-free graphs and extend the result [14] for P_4-free graphs in a systematic way.

In Section 2, we consider the cases where H contains a cycle or a claw. We first prove that Near-Bipartiteness, and thus Independent Feedback Vertex Set, is NP-complete on line graphs, which form a subclass of the class of claw-free graphs. We then prove that Independent Feedback Vertex Set is NP-complete for graphs of arbitrarily large girth. Together, these results imply that Independent Feedback Vertex Set is NP-complete for H-free graphs if H contains a cycle or claw. Hence, only the cases where H is a *linear forest*, that is, a disjoint union of paths, remain open. In particular, the case where H is a single path has not yet been resolved. Due to the result of [14] for P_4-free graphs, the first open case to consider is when $H = P_5$.

The class of P_5-free graphs is a well-studied graph class. For instance, Hoàng et al. [8] proved that for every integer k, k-Colouring is polynomial-time solvable for P_5-free graphs, whereas Golovach and Heggernes [7] showed that Choosability is fixed-parameter tractable for P_5-free graphs when parameterized by the size of the lists of admissible colours. Lokshantov et al. [9] solved a long-standing open problem by giving a polynomial-time algorithm for Independent Set restricted to P_5-free graphs.

Our main result is that Independent Feedback Vertex Set is polynomial-time solvable for P_5-free graphs and this is proved in Sections 3 and 4. In Section 3 we give a polynomial-time algorithm for Near-Bipartiteness on P_5-free graphs. Then in Section 4 we show how to *extend* this algorithm to solve Independent Feedback Vertex Set in polynomial time for P_5-free graphs. Our results for Independent Feedback Vertex Set also hold for Independent Odd Cycle Transversal (see the arXiv version of our paper).

2 Hardness When H Contains a Cycle or Claw

The *line graph* $L(G)$ of a graph $G = (V, E)$ has the edge set E of G as its vertex set, and two vertices e_1 and e_2 of $L(G)$ are adjacent if and only if e_1 and e_2 share a common end-vertex in G. The *claw* is the graph with vertices a, b, c, d and edges ab, ac, ad. It is well known and easy to see that every line graph is claw-free. We omit the proof of Theorem 1.

▶ **Theorem 1.** Near-Bipartiteness *is* NP*-complete for line graphs of planar subcubic bipartite graphs.*

Feedback Vertex Set is also NP-complete for line graphs of planar cubic bipartite graphs [12]. Theorem 1 has the following immediate consequence (take $k = n$).

▶ **Corollary 2.** Independent Feedback Vertex Set *is* NP*-complete for line graphs of planar subcubic bipartite graphs.*

The *length* of a cycle C is the number of edges of C. The *girth* $g(G)$ of a graph G is the length of a shortest cycle of G. Proposition 3 follows from known results; we omit the proof.

▶ **Proposition 3.** *For every constant $g \geq 3$,* Independent Feedback Vertex Set *is* NP*-complete for graphs of maximum degree at most 4 and girth at least g.*

Recall that every line graph is claw-free. We also observe that for a graph H with a cycle C, the class of graphs of girth at least $|C| + 1$ is a subclass of the class of H-free graphs. Hence, we can combine Corollary 2 and Proposition 3 to obtain the following result.

▶ **Corollary 4.** *Let H be a graph that contains a claw or a cycle. Then* Independent Feedback Vertex Set *is* NP*-complete for H-free graphs of maximum degree at most 4.*

3 Near-Bipartiteness of P_5-free Graphs

In this section, we show that Near-Bipartiteness is in P for P_5-free graphs, i.e. we give a polynomial-time algorithm for testing if a P_5-free graph has an independent feedback vertex set. To obtain a *minimum* feedback vertex set we need to first run this algorithm (apart from the step where we encode an instance of Trouble-Free Colouring as an instance of 2-Satisfiability) and then do the additional work described in Section 4.

Our algorithm in this section solves a slightly more general problem, which is a special variant of List 3-Colouring. In the List 3-Colouring problem each vertex v is assigned a subset $L(v)$ of colours from $\{1, 2, 3\}$ and we must verify if a 3-colouring exists in which each vertex v is coloured with a colour from $L(v)$. We say that a 3-colouring of a graph G is *semi-acyclic* if the vertices coloured 2 or 3 induce a forest, and we note that G has such a colouring if and only if G is near-bipartite.

List Semi-Acyclic 3-Colouring
Instance: a graph G and a function $L : V(G) \to \{S \mid S \subseteq \{1, 2, 3\}\}$.
Question: does G have a semi-acyclic 3-colouring c such that $c(v) \in L(v)$ for all $v \in V(G)$?

A graph G is near-bipartite if and only if (G, L), with $L(v) = \{1, 2, 3\}$ for all $v \in V(G)$, is a yes-instance of List Semi-Acyclic 3-Colouring. To recognise near-bipartite P_5-free graphs in polynomial time, we will show the stronger statement that List Semi-Acyclic 3-Colouring is polynomial-time solvable for P_5-free graphs. A set of vertices in a graph G is *dominating* if every vertex of G is either in the set or has at least one neighbour in it. We will use a lemma of Bacsó and Tuza.

▶ **Lemma 5** ([2]). *Every connected P_5-free graph admits a dominating set that induces either a clique or a P_3.*

Lemma 5 implies that every connected 3-colourable P_5-free graph has a dominating set of size at most 3 (since it has no clique on more than three vertices). This was used by Randerath et al. [13] to show that 3-Colouring is polynomial-time solvable on P_5-free graphs. Their algorithm tries all possible 3-colourings of a dominating set of size at most 3. It then adjusts the lists of the other vertices (which were originally set to $\{1, 2, 3\}$) to lists of size at most 2. As shown by Edwards [5], 2-List Colouring can be translated to an instance of 2-Satisfiability, which is solvable in linear time. Hence this approach results in a polynomial (even constant) number of instances of the 2-Satisfiability problem. Our goal is also to apply Lemma 5 on a connected P_5-free graph G and to reduce an instance (G, L) of List Semi-Acyclic 3-Colouring to a polynomial number of instances of 2-Satisfiability. However, this is less straightforward than in the case of 3-Colouring restricted to P_5-free graphs: the restriction of List Semi-Acyclic 3-Colouring to lists of size 2 turns out to be NP-complete for general graphs even if every list consists of either colours 1 and 3 or only colour 2. We omit the proof of the next theorem.

▶ **Theorem 6.** List Semi-Acyclic 3-Colouring *is* NP*-complete even if $L(v) \in \{\{1, 3\}, \{2\}\}$ for every vertex v in the input graph.*

By Theorem 6, to prove that List Semi-Acyclic 3-Colouring is in P on P_5-free graphs, we need to refine our analysis and exploit P_5-freeness beyond the use of Lemma 5. We adapt the approach used by Hoàng et al. [8] to show that k-Colouring is in P on P_5-free graphs for all $k \geq 4$ (extending the analogous result of Randerath et al. [13] for 3-Colouring). Let us outline the proof of [8]. Lemma 5 implies that every k-colourable P_5-free graph G has a dominating set D of size at most k (as the clique number is at most k). Fix an ordering $D = \{v_1, \ldots, v_{|D|}\}$. Then decompose the set of vertices not in D into $|D|$ "layers" so that the vertices in a layer i are adjacent to v_i (and possibly to v_j for $j > i$) but not to any v_h with $h < i$. Using the P_5-freeness of G to analyse the adjacencies between different layers, it is possible to branch in such a way that a polynomial number of instances of $(k-1)$-Colouring are obtained. Hence, by repetition, a polynomial number of instances of 3-Colouring are reached, which can be solved in polynomial time due to the result of [13].

The algorithm of [8] works by considering the more general List k-Colouring problem, where each vertex v is assigned a list $L(v) \subseteq \{1, \ldots, k\}$ of permitted colours and the question is whether there is a colouring in which each vertex is assigned a colour from its list. The algorithm immediately removes any vertices whose lists have size 1 at any point (and then adjusts the lists of admissible colours of all neighbours of such vertices). We will follow the approach of [8], but cannot remove any vertices whose lists contain a singleton colour if this colour is 2 or 3. To overcome this extra complication we carefully analyse the 4-vertex cycles in the graph after observing that these cycles are the only obstacles that may prevent a 3-colouring of a P_5-free graph from being semi-acyclic.

For a subset $S \subseteq V(G)$ of a graph G, we let $G[S]$ denote the subgraph of G induced by S.

▶ **Theorem 7.** List Semi-Acyclic 3-Colouring *is solvable on P_5-free graphs in $O(n^{16})$ time.*

Proof. Consider an input (G, L) for the problem such that G is P_5-free. Since the problem can be solved component-wise, we may assume that G is connected. If G contains an induced K_4 then it is not 3-colourable and the input is a `no`-instance. As we can test whether G contains an induced K_4 in $O(n^4)$ time, we now assume that G is K_4-free. We may also assume that G contains at least three vertices, otherwise the problem can be trivially solved.

For $i \in \{1, 2, 3\}$ let $G_i = G[\{v \in V(G) \mid i \notin L(v)\}]$. We apply the following propagation rules exhaustively, and, later in the proof, every time we branch on possibilities, we assume that these rules are again applied exhaustively immediately afterwards.

Rule 1. If $u, v \in V(G)$ are adjacent and $|L(u)| = 1$, set $L(v) := L(v) \setminus L(u)$.
Rule 2. If $L(v) = \emptyset$ for some $v \in V(G)$, return `no`.
Rule 3. If G_i is not bipartite for some $i \in \{1, 2, 3\}$, return `no`.
Rule 4. If G_1 contains an induced C_4, return `no`.
Rule 5. If G contains an induced C_4, and exactly one vertex v of this cycle has a list containing the colour 1, set $L(v) = \{1\}$.

We must show that these rules are safe. That is, that when they modify the instance they do not affect whether or not it is a `yes`-instance or a `no`-instance, and when they return the answer `no`, this is correct and no semi-acyclic colouring that respects the lists can exist. This is trivial for Rules 1 and 2. We may apply Rule 3 since in any 3-colouring of G every pair of colour classes must induce a bipartite graph. We may apply Rules 4 and 5 since in every solution, every induced C_4 must contain at least one vertex coloured with colour 1. In fact, if there is a 3-colouring of G with a cycle made of vertices coloured only 2 and 3, then this cycle must be an even cycle. Since G is P_5-free, such a cycle must in fact be isomorphic to C_4. Hence the problem, when restricted to P_5-free graphs, is equivalent to testing whether G has a 3-colouring respecting the lists such that every induced C_4 contains at least one vertex coloured with colour 1.

By Lemma 5, G has a dominating set S that either is a clique or induces a P_3. If it is a clique, then it has at most three vertices, as G is K_4-free, so we can find such a set in $O(n^4)$ time. Thus, adding vertices arbitrarily if necessary, we may assume $S = \{a_1, a_2, a_3\}$. We consider all possible combinations of colours that can be assigned to the vertices in S, that is, we branch into at most 3^3 cases, in which a_1, a_2 and a_3 have each received a colour, or equivalently, have had their list of permissible colours reduced to size exactly 1. In each case we proceed as follows.

Assume that $L(a_1) = \{c_1\}$, $L(a_2) = \{c_2\}$ and $L(a_3) = \{c_3\}$ and again apply the propagation rules above. Partition the vertices of $V \setminus S$ into three parts V_1, V_2, V_3: let V_1 be the set of neighbours of a_1 in $V \setminus S$, let V_2 be the set of neighbours of a_2 in $V \setminus S$ that are not adjacent to a_1, and let $V_3 = V(G) \setminus (S \cup V_1 \cup V_2)$. Each vertex in V_3 is non-adjacent to a_1 and a_2, so it is adjacent to a_3, as S is dominating. For $i \in \{1, 2, 3\}$, if $v \in V_i$, then $L(v) \subseteq \{1, 2, 3\} \setminus \{c_i\}$ by Rule 1, so each vertex has at most two colours in its list. For $i \in \{1, 2, 3\}$ let V_i' be the subset of vertices v in V_i with $L(v) = \{1, 2, 3\} \setminus \{c_i\}$. Recall that for $i \in \{1, 2, 3\}$, we defined $G_i = G[\{v \in V(G) \mid i \notin L(v)\}]$. As for every $i \in \{1, 2, 3\}$, every vertex of V_i belongs to G_{c_i}, V_1, V_2 and V_3 each induce a bipartite graph in G by Rule 3. Therefore, we may partition each V_i' into two (possibly empty) independent sets V_i'' and V_i'''.

Our strategy is to reduce the instance (G, L) to a polynomial number of instances (G, L'), in which there are no edges between any two distinct sets V_i' and V_j' (defined with respect to L'). We will do this by branching on possible partial colourings in such a way that afterwards there are no edges between V_i'' and V_j''', no edges between V_i'' and V_j'' and no edges between V_i''' and V_j''' for every pair $i, j \in \{1, 2, 3\}$ with $i \neq j$. As the branching procedure is similar for each of these possible combinations, we pick an arbitrary pair, namely V_1'' and V_2''. As we shall see, we do not remove any edges between V_1'' and V_2''. Instead, we decrease the lists of some of their vertices to size 1, so that these vertices will leave $V_1' \cup V_2'$ by definition of V_1', V_2' (and thus leave V_1'' and V_2'' by definition of V_1'', V_2'').

Suppose that $G[V_1'' \cup V_2'']$ contains an induced $2P_2$ with edges uu' and vv' for $u, v \in V_1''$ and $u', v' \in V_2''$. Then $G[\{u', u, a_1, v, v'\}]$ is a P_5, a contradiction. It follows that $G[V_1'' \cup V_2'']$ is a $2P_2$-free bipartite graph, that is, the edges between V_1'' and V_2'' form a chain graph, which means that the vertices of V_1'' can be linearly ordered by inclusion of neighbourhood in V_2''. In other words, we fix an ordering $V_1'' = \{u_1, \ldots, u_k\}$ such that $N_{V_2''}(u_1) \supseteq \cdots \supseteq N_{V_2''}(u_k)$.

We choose an arbitrary colour $c' \in \{1, 2, 3\} \setminus \{c_1, c_2\}$. Note that if $c_1 \neq c_2$ then this choice is unique and otherwise there are two choices (as we will show, it suffices to branch on only one choice). Also note that every vertex in V_1'' and V_2'' has colour c' in its list.

We now branch over $k + 1$ possibilities, namely the possibilities that vertex u_i is the first vertex coloured with colour c' (so vertices $u_1, \ldots, u_{i-1}$, if they exist, do not get colour c') and the remaining possibility that no vertex of V_1'' is coloured with colour c'. To be more precise, for branch $i = 1$ we set $L(u_1) = \{c'\}$, for each branch $2 \leq i \leq k$ we remove colour c' from each of $L(u_1), \ldots, L(u_{i-1})$ and set $L(u_i) = \{c'\}$ and for branch $i = k + 1$ we remove colour c' from each of $L(u_1), \ldots, L(u_k)$. If $i = k + 1$, all vertices of V_1'' will have a unique colour in their list and thus leave V_1' and thus V_1'' by definition of V_1'. Hence, V_1'' becomes empty and thus we no longer have edges between V_1'' and V_2''. Otherwise, if $i \leq k$, then all of $u_1, \ldots, u_i$ will have a list containing exactly one colour, so they will leave V_1' and therefore V_1''. By Rule 1 all neighbours of u_i in V_2'' will have c' removed from their lists, so they will leave V_2' and therefore V_2''. By the ordering of neighbourhoods of vertices in V_1'', this means that no vertex remaining in V_1'' has a neighbour remaining in V_2'', so if $i \leq k$, then it is also the case that we no longer have edges between V_1'' and V_2''.

Note that removing all the edges between distinct sets V'_i and V'_j in the above way involves branching into $O(n^{12})$ cases. We consider each case separately, and for each case we proceed as below.

Thus we may assume that there are no edges between any two distinct sets V'_i and V'_j. We say that an induced C_4 is *tricky* if there exists a (proper) colouring of it (not necessarily extendable to all of G) using only the colours 2 and 3 such that every vertex receives a colour from its list. We say that a vertex in an induced C_4 is *good* for this C_4 if its list contains the colour 1. By definition of tricky, every good vertex for a tricky C_4 must belong to $V'_1 \cup V'_2 \cup V'_3$. By Rules 4 and 5, every tricky C_4 must contain at least two good vertices. If a C_4 contains two good vertices that are adjacent, then they must belong to the same set V'_i (since there are no edges between any two distinct sets V'_i and V'_j), so they must have the same list. This means that in every colouring of this C_4 that respects the lists, one of the good vertices in this C_4 will be coloured with colour 1, contradicting the definition of tricky. We conclude that every tricky C_4 must contain exactly two good vertices, which must be non-adjacent.

Suppose G contains a tricky induced C_4 on vertices v_1, v_2, v_3, v_4, in that order, such that v_1 and v_3 are good. Since the C_4 is tricky, we must either have:

- $2 \in L(v_1)$, $3 \in L(v_2)$, $2 \in L(v_3)$ and $3 \in L(v_4)$ or
- $3 \in L(v_1)$, $2 \in L(v_2)$, $3 \in L(v_3)$ and $2 \in L(v_4)$.

Since v_2 and v_4 are not good, and there are no edges between distinct sets of the form V'_i, the above implies that one of the following must hold:

- $L(v_1) = \{1, 2\}$, $L(v_2) = \{3\}$, $L(v_3) = \{1, 2\}$ and $L(v_4) = \{3\}$ or
- $L(v_1) = \{1, 3\}$, $L(v_2) = \{2\}$, $L(v_3) = \{1, 3\}$ and $L(v_4) = \{2\}$.

We say that an induced C_4 is *strongly tricky* if its vertices have lists of this form. Note that, by the above arguments, we may assume that all tricky induced C_4s in the instances we consider are in fact strongly tricky. For $S \subsetneq \{1, 2, 3\}$, let L_S denote the set of vertices v with $L(v) = S$ (to simplify notation, we will write L_i instead of $L_{\{i\}}$ and $L_{i,j}$ instead of $L_{\{i,j\}}$ wherever possible). Note that for distinct sets $S, T \subseteq \{1, 2, 3\}$ with $|S| = |T| = 2$, no vertex in L_S can have a neighbour in L_T, because such vertices would be in different sets V'_i, and therefore cannot be adjacent by our branching. By Rule 1, if $S \subsetneq T \subsetneq \{1, 2, 3\}$ with $|S| = 1$ and $|T| = 2$, then no vertex in L_S can have a neighbour in L_T. From the above two arguments it follows that if a vertex is in $L_{1,2}$, $L_{2,3}$ or $L_{1,3}$, then all its neighbours outside this set must be in L_3, L_1 or L_2, respectively.

Recall that every tricky induced C_4 is strongly tricky, and is therefore entirely contained in either $G[L_2 \cup L_{1,3}]$ or $G[L_3 \cup L_{1,2}]$. By Rule 3, G_1 and therefore $G[L_{2,3}]$ is bipartite. Hence we can colour the vertices of $L_{2,3}$ with colours from their lists such that no vertex in $L_{2,3}$ is adjacent to a vertex of the same colour in G and no induced C_4s are coloured with colours alternating between 2 and 3 (indeed, recall that induced C_4s cannot exist in $G(L_{2,3})$ by Rule 4). It therefore remains to check whether the vertices of $G[L_2 \cup L_{1,3}]$ (and $G[L_3 \cup L_{1,2}]$) can be coloured with colours from their lists so that no pair of adjacent vertices in $L_{1,3}$ (respectively $L_{1,2}$) receive the same colour and every strongly tricky C_4 has at least one vertex coloured 1. By symmetry, it is sufficient to show how to solve the $G[L_2 \cup L_{1,3}]$ case. Hence we have reduced the original instance (G, L) to a polynomial number of instances of a new problem, which we define below after first defining the instances.

▶ **Definition 8.** A graph $G = (V, E)$ is *troublesome* if every vertex v in G has list either $L(v) = \{2\}$ or $L(v) = \{1, 3\}$, such that L_2 is an independent set and $L_{1,3}$ induces a bipartite graph.

In particular, for each of our created instances the set L_2 is independent due to Rule 1 and $L_{1,3}$ induces a bipartite graph by Rule 3. Note that by definition of troublesome, all tricky induced C_4s in a troublesome graph are strongly tricky.

▶ **Definition 9.** Let G be a troublesome graph. A 3-colouring of the graph G is *trouble-free* if each vertex receives a colour from its list, no two adjacent vertices of G are coloured alike and at least one vertex of every strongly tricky induced C_4 of G receives colour 1.

This leads to the following problem.

Trouble-Free Colouring	
Instance:	a troublesome P_5-free graph G
Question:	does G have a trouble-free colouring?

It is easy to verify that Trouble-Free Colouring can be encoded as an instance of 2-Satisfiability. So, by branching, we have reduced the original instance (G, L) of List Semi-Acyclic 3-Colouring to a polynomial number of instances of 2-Satisfiability. If we find that one of the instances of the latter problem is a `yes`-instance, then we obtain a corresponding `yes`-instance of Trouble-Free Colouring. We therefore solve Trouble-Free Colouring on $G[L_2 \cup L_{1,3}]$ and (after swapping colours 2 and 3) on $G[L_3 \cup L_{1,2}]$. If one of these two instances of Trouble-Free Colouring is a `no`-instance, then we return `no` for this branch and try the next one. If both of these are `yes`-instances, then we return `yes` and obtain a semi-acyclic 3-colouring by combining the colourings on $G[L_1 \cup L_{2,3}]$, $G[L_2 \cup L_{1,3}]$ and (after swapping colours 2 and 3 back) $G[L_3 \cup L_{1,2}]$. If every branch returns `no` then the original graph has no semi-acyclic 3-colouring. This completes the proof of the correctness of the algorithm. We omit the runtime analysis. ◀

We obtain the following corollary.

▶ **Corollary 10.** Near-Bipartiteness *can be solved in $O(n^{16})$ time for P_5-free graphs.*

Proof. Let G be a graph. Set $L(v) = \{1, 2, 3\}$ for all $v \in V(G)$. Then G is near-bipartite if and only if (G, L) is a `yes`-instance of List Semi-acyclic 3-Colouring. In particular, the vertices coloured 1 by a semi-acyclic colouring of G form an independent feedback vertex set of G. The corollary follows by Theorem 7. ◀

4 Independent Feedback Vertex Sets of P_5-free Graphs

In this section we prove that Independent Feedback Vertex Set is polynomial-time solvable for P_5-free graphs by extending the algorithm from Section 3. We omit the proof of Lemma 11, which uses the proof of Theorem 7. As such, we heavily use Definitions 8 and 9. Let $G = (V, E)$ be a troublesome P_5-free graph. For a trouble-free colouring c of G, let $t_c(G) = |\{u \in V \mid c(u) = 1\}|$ denote the number of vertices of G coloured 1 by c. Let $t(G)$ be the minimum value $t_c(G)$ over all trouble-free colourings c of G.

▶ **Lemma 11.** *Let G be a near-bipartite P_5-free graph. In $O(n^{16})$ time it is possible to reduce the problem of finding the smallest independent feedback vertex set of G to finding the value $t(G')$ of $O(n^{12})$ instances of* Trouble-Free Colouring*, all on induced subgraphs of G.*

By Lemma 11, it suffices to prove the following lemma (in its proof we again use the terminology introduced in the proof of Theorem 7).

▶ **Lemma 12.** *Let G be a troublesome P_5-free graph on n vertices. Determining $t(G)$ can be done in $O(n^4)$ time.*

Proof. Let $G = (V, E)$ be a troublesome P_5-free graph. Note that in G, an induced C_4 on vertices v_1, v_2, v_3, v_4, in that order, is strongly tricky if $v_1, v_3 \in L_{1,3}$ and $v_2, v_4 \in L_2$.

We construct an auxiliary graph H as follows. We let $V(H) = L_{1,3}$. Every edge of $G[L_{1,3}]$ belongs to H. We say that such edges are *red*. For non-adjacent vertices $v_1, v_3 \in L_{1,3}$, if there is a strongly tricky induced C_4 on vertices v_1, v_2, v_3, v_4 with $v_2, v_4 \in L_2$, we add the edge v_1v_3 to H. We say that such edges are *blue*. Note that H is a supergraph of $G[L_{1,3}]$ and that there exists at most one edge, which is either blue or red, between any two vertices of H. We call a colouring of H *feasible* if the following two conditions are met:

1. no red edge is monochromatic, that is, the two end-vertices of every red edge must be coloured 1&3 respectively or 3&1 respectively;
2. the two end-vertices of every blue edge must be coloured, respectively, 1&3, 3&1 or 1&1 (the only forbidden combination is 3&3, as in this case we obtain a strongly tricky induced C_4 in G with colours 2 and 3).

We note that there is a one-to-one correspondence between the set of trouble-free colourings of G and the set of feasible colourings of H. Hence, we need to find a feasible colouring of H that minimises the number of vertices coloured 1. Let $R_1, \ldots, R_p$ be the components of $G[L_{1,3}]$, or equivalently, of the graph obtained from H after removing all blue edges. We call these components *red*. As $G[L_{1,3}]$ is bipartite and P_5-free, all red components of H are bipartite and P_5-free. We denote the bipartition classes of each R_i by X_i and Y_i, arbitrarily (note that these classes are unique, up to swapping their order). We apply the following five rules on H exhaustively (we omit proofs of correctness for these rules)

Rule 1. If there is a blue edge in H between two vertices $u, v \in X_i$ or two vertices $u, v \in Y_i$, then assign colour 1 to u and v.

Rule 2. If there is a blue edge e in H between a vertex $u \in X_i$ and a vertex $v \in Y_i$, then delete e from H.

Rule 3. If there are blue edges uv and uv' where $u \in X_i \cup Y_i$, $v \in X_j$ and $v' \in Y_j$ $(j \neq i)$, then assign colour 1 to u.

Rule 4. If an uncoloured vertex u is adjacent to a vertex with colour 3 via a blue edge, then assign colour 1 to u.

Rule 5. If an uncoloured vertex u is adjacent to a coloured vertex v via a red edge, then assign colour 1 to u if v has colour 3 and colour 3 to u otherwise.

Rule 6. If there is a red edge with end-vertices both coloured 1 or both coloured 3, or a blue edge with end-vertices both coloured 3, then return `no`.

Rule 7. Remove all vertices that have received colour 1 or colour 3, keeping track of the number of vertices coloured 1.

By Rules 1 and 2, if two vertices are in the same red component R_i, we may assume that they are not connected by a blue edge. Hence, we may assume from now on that red components contain no blue edges in H. By Rule 3, we may also assume that no vertex in $V(H) \setminus V(R_j)$ is joined via blue edges to both a vertex in X_j and a vertex in Y_j.

From H we construct another auxiliary graph H^* as follows. First, we replace each red component R_i on more than two vertices by an edge x_iy_i, which we say is a *red* edge. Hence, the set of red components of H is reduced to a set of *red* components in H^* in such a way that each red component of H^* is either an edge or a single vertex. Next, for $i \neq j$ we add an edge, which we say is a *blue* edge, between two vertices x_i and x_j if and only if there is a blue edge between a vertex in X_i and a vertex in X_j. We add blue edges between vertices y_i and y_j, and between vertices x_i and y_j in the same way.

Recall that, by Rules 1 and 2, no two vertices in any R_i are connected by a blue edge. So every feasible colouring of H corresponds to a feasible colouring of H^* and vice versa. To keep track of the number of vertices coloured 1, we introduce a weight function $w : V(H^*) \to \mathbb{Z}_+$ by setting $w(x_i) = |X_i|$ and $w(y_i) = |Y_i|$. Our new goal is to find a feasible colouring c of H^* that minimises the sum of the weights of the vertices coloured 1, which we denote by $w(c)$.

Since for each i no vertex in $V(H) \setminus V(R_i)$ is joined via blue edges to both a vertex in X_i and a vertex in Y_i, we find that H^* contains no triangle consisting of one red edge and two blue edges. As red edges induce a disjoint union of isolated edges, this means that the only triangles in H^* consist of only blue edges. Let $B_1, \ldots, B_q$ be the components of the graph obtained from H^* after removing all red edges. We call these components *blue* (even in the case where they are singletons). We need the following claim (we omit its proof).

▶ **Claim 13.** *Each B_i is a complete graph.*

By Claim 13, H^* is the disjoint union of several blue complete graphs with red edges between them. Recall that we allow the case where these blue complete graphs contain only one vertex. On H^* we apply the following rule exhaustively in combination with Rules 4–7. While doing this we keep track of the weights of the vertices coloured 1.

Rule 8. If there exist (red) edges u_1v_1 and u_2v_2 for $u_1, u_2 \in B_i$ and $v_1, v_2 \in B_j$ $(i \neq j)$, then assign colour 1 to every vertex in $(B_i \cup B_j) \setminus \{u_1, u_2, v_1, v_2\}$.

Since Rules 4 and 5 can be safely applied on H, they can be safely applied on H^*. It follows that Rules 6 and 7 can also be safely applied on H^*. We may also safely apply Rule 8: the red edges u_1v_1 and u_2v_2 force u_i and v_i to have different colours for $i \in \{1, 2\}$, whereas the blue components forbid u_1, u_2 both being coloured 3 and v_1, v_2 both being coloured 3. Hence, exactly one of u_1, u_2 and exactly one of v_1, v_2 must be coloured 3. Because at most one vertex in any blue component may be coloured 3, this implies that all vertices in $(B_i \cup B_j) \setminus \{u_1, u_2, v_1, v_2\}$ must be coloured 1.

As every vertex is incident with at most one red edge in H^*, we obtain a resulting graph that is an induced subgraph of H^* with the following property: if there exist (red) edges u_1v_1 and u_2v_2 for $u_1, u_2 \in B_i$ and $v_1, v_2 \in B_j$, then $\{u_1, u_2, v_1, v_2\}$ induces a connected component of H^*. We can colour such a 4-vertex component in exactly two ways and we remember the colouring with minimum weight (either $w(u_1) + w(v_2)$ or $w(u_2) + w(v_1)$ depending on whether u_1 gets colour 1 or 3, respectively). Hence, from now on we may assume that the resulting graph, which we again denote by H^*, does not have such components. That is, there is at most one red edge between any two blue components of H^*. As we can colour H^* component-wise, we may assume without loss of generality that H^* is connected.

For each B_i we define the subset B_i' to consist of those vertices of B_i not incident with a red edge, and we let $B_i'' = B_i \setminus B_i'$. We note the following. If we colour every vertex of some B_i'' with colour 1, then every neighbour of every vertex of B_i'' in any other blue component B_j must be coloured 3 by Rule 5 (recall that vertices in different blue components are connected to each other only via red edges). As soon as one vertex u in some blue component B_j has colour 3, all other vertices in $B_j - u$ must get colour 1 by Rule 4. In this way we can use Rule 4 and 5 exhaustively to *propagate* the colouring to other vertices of H^* where we have no choice over what colour to use.

Recall that no vertex of H^* is incident with more than one red edge. This is a crucial fact: it implies that propagation to other blue components of H^* happens only via red edges vw between two blue components, one end-vertex of which, say v, is first coloured 1, which implies that the other end-vertex w of such an edge must get colour 3; this in turn implies that all other vertices in the blue component containing w must get colour 1 and so on. Hence, as H^* was assumed to be connected, colouring every vertex of a set B_i'' with colour 1 propagates to all vertices of H^* except for the vertices of B_i'. Note that we may still colour (at most) one vertex of B_i' with colour 3.

Due to the above, we now do as follows for each $i \in \{1, \ldots, q\}$ in turn: We colour every vertex of B_i'' with colour 1 and propagate to all vertices of H^* except for the vertices of B_i'. If we obtain a monochromatic red edge or a blue edge whose end-vertices are coloured 3, we discard this option (by Rule 6). Otherwise, we assign colour 3 to a vertex $u \in B_i'$ with maximum weight $w(u)$ over all vertices in B_i' (if $B_i' \neq \emptyset$). We store the resulting colouring c_i that corresponds to this option.

After doing the above for all q options, it remains to consider the cases where every B_i'' contains (exactly) one vertex coloured 3. Before we can use another propagation argument that tells us which vertices get colour 3, we first perform the following steps, only applying a step when the previous ones have been applied exhaustively. These steps follow immediately from the assumption that every B_i'' contains a vertex coloured 3.

(i) Colour all vertices of every B_i' with colour 1 (doing this does not cause any propagation).
(ii) If some B_i'' consists of a single vertex, then colour this vertex with colour 3, and afterwards propagate by using Rule 5 exhaustively.
(iii) Remove coloured vertices using Rule 7.

If due to (ii) we obtain a monochromatic red edge or a blue edge whose end-vertices are coloured 3, we discard this option (using Rule 6). Otherwise, we may assume from now on that $B_i' = \emptyset$, so $B_i'' = B_i$ due to (i) and that $|B_i| \geq 2$ due to (ii). Note that doing (iii) does not disconnect the graph: the vertices in the vertices in B_i' that are coloured in (i) only have neighbours in the clique B_i (and these are via blue edges) and if a vertex of $v \in B_i''$ is coloured with colour 3 in (ii), then its only neighbour w (via a red edge) is in a set B_j'' and since (i) has been applied exhaustively, the only other neighbours of w are in B_j'' (via blue edges), so the propagation stops there and the graph does not become disconnected.

By our procedure, every vertex of every blue component B_i is incident with a red edge, so the total number of outgoing red edges for each B_i is equal to $|B_i| \geq 2$, and all outgoing red edges go to $|B_i|$ different blue components. Hence the graph H' obtained from H^* by contracting each blue component to a single vertex has minimum degree at least 2. As H' has minimum degree at least 2, we find that H' contains an edge that is not a bridge (a bridge in a connected graph is an edge whose removal disconnects the graph). Let uv be the corresponding red edge in H^*, say u belongs to B_i and v belongs to B_j.

We have two options to colour u and v, namely by $1, 3$ or $3, 1$. We try them both. Suppose we first give colour 1 to u. Then we propagate in the same way as before. Because uv is not a bridge in H', eventually we propagate back to B_i by giving colour 3 to an uncoloured vertex of B_i. When that happens we have "identified" the colour-3 vertex of B_i and then need to colour all other vertices of B_i with colour 1. This means that we can in fact propagate to all blue components of H^*, just as before. If at some point we obtain a monochromatic red edge or a blue edge with end-vertices coloured 3, then we discard this option (by Rule 6). Next, we give colour 1 to v and proceed similarly.

At the end we have at most $q + 2$ different feasible colourings of H^*. We pick the one with minimum weight and translate the colouring to a feasible colouring of H. Finally, we translate the feasible colouring of H to a trouble-free colouring of the original graph G. We omit the analysis of the runtime. ◀

▶ **Theorem 14.** *The size of a minimum independent feedback vertex set of a P_5-free graph on n vertices can be computed in $O(n^{16})$ time.*

Proof. Let G be a P_5-free graph on n vertices. As we can check in $O(n^{16})$ time if G is near-bipartite, we may assume without loss of generality that G is near-bipartite. Then, by Lemma 11, in $O(n^{16})$ time we can reduce the problem finding the value $t(G')$ of $O(n^{12})$ instances of Trouble-Free Colouring, all on induced subgraphs of G (which have at most n vertices). By Lemma 12 we can compute $t(G')$ in $O(n^4)$ time for each such instance. This gives a total runtime of $O(n^{16})$. ◀

▶ **Remark 15.** *From our proof, we can find in polynomial time not just the size of a minimum independent feedback vertex set, but also the set itself. The corresponding algorithm can also be adapted to find in polynomial time a maximum independent feedback vertex of a P_5-free graph, or an independent feedback vertex set of arbitrary fixed size (if one exists). Our algorithm can also be adapted to solve* Independent Odd Cycle Transversal *in $O(n^{16})$ time.*

References

1 Akanksha Agrawal, Sushmita Gupta, Saket Saurabh, and Roohani Sharma. Improved algorithms and combinatorial bounds for independent feedback vertex set. *Proc. IPEC 2016, LIPIcs*, 63:2:1–2:14, 2017.

2 Gábor Bacsó and Zsolt Tuza. Dominating cliques in P_5-free graphs. *Periodica Mathematica Hungarica*, 21(4):303–308, 1990.

3 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Recognizing graphs close to bipartite graphs. *Proc. MFCS 2017, LIPIcs*, 83:70:1–70:14, 2017.

4 Andreas Brandstädt, Synara Brito, Sulamita Klein, Loana Tito Nogueira, and Fábio Protti. Cycle transversals in perfect graphs and cographs. *Theoretical Computer Science*, 469:15–23, 2013.

5 Keith Edwards. The complexity of colouring problems on dense graphs. *Theoretical Computer Science*, 43:337–343, 1986.

6 Wayne Goddard and Michael A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313(7):839–854, 2013.

7 Petr A. Golovach and Pinar Heggernes. Choosability of P_5-free graphs. *Proc. MFCS 2009, LNCS*, 5734:382–391, 2009.

8 Chính T. Hoàng, Marcin Kamiński, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57(1):74–81, 2010.

9 Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. *Proc. SODA 2014*, pages 570–581, 2014.

10 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30:1–30:35, 2013.

11 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.

12 Andrea Munaro. On line graphs of subcubic triangle-free graphs. *Discrete Mathematics*, 340(6):1210–1226, 2017.

13 Bert Randerath, Ingo Schiermeyer, and Meike Tewes. Three-colourability and forbidden subgraphs. II: polynomial algorithms. *Discrete Mathematics*, 251(1–3):137–153, 2002.

14 Yuma Tamura, Takehiro Ito, and Xiao Zhou. Algorithms for the independent feedback vertex set problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E98-A(6):1179–1188, 2015.

15 Aifeng Yang and Jinjiang Yuan. Partition the vertices of a graph into one independent set and one acyclic set. *Discrete Mathematics*, 306(12):1207–1216, 2006.

On the Convergence Time of a Natural Dynamics for Linear Programming

Vincenzo Bonifaci

Institute for the Analysis of Systems and Informatics, National Research Council of Italy (IASI-CNR), Rome, Italy
vincenzo.bonifaci@iasi.cnr.it

Abstract

We consider a system of nonlinear ordinary differential equations for the solution of linear programming (LP) problems that was first proposed in the mathematical biology literature as a model for the foraging behavior of acellular slime mold *Physarum polycephalum*, and more recently considered as a method to solve LP instances. We study the convergence time of the continuous Physarum dynamics in the context of the linear programming problem, and derive a new time bound to approximate optimality that depends on the relative entropy between projected versions of the optimal point and of the initial point. The bound scales logarithmically with the LP cost coefficients and linearly with the inverse of the relative accuracy, establishing the efficiency of the dynamics for arbitrary LP instances with positive costs.

1998 ACM Subject Classification F.1.1 Models of Computation, G.1.6 Optimization

Keywords and phrases linear programming, natural algorithm, Physarum polycephalum, relative entropy, Mirror Descent

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.17

1 Introduction

The theoretical analysis of natural systems has historically been the domain of mathematical biology, dynamical systems theory, and physics, but certain natural processes are capable of exhibiting remarkable information processing abilities which are often best understood from an optimization perspective. Indeed, the application of a "computational lens" to such processes has been advocated in different disciplines, and efforts are underway to identify, classify and analyze these so-called *natural algorithms* [9, 15].

One such example can be found in the slime mold *Physarum polycephalum*. *P. polycephalum* is an acellular, amoeboid slime mold in the *Mycetozoa* group. In controlled experiments, the slime mold's capabilities have been leveraged to determine the shortest path between two locations in a network [14, 20] and, more generally, to adaptively form efficient transport networks [22]. In fact, a dynamical model proposed by the mathematical biologists to describe the time evolution of *P. polycephalum*'s network physiology [21] has been rigorously proved to be algorithmically efficient for problems such as the single-source single-sink shortest path problem [4, 8] and the minimum-cost transshipment problem [11, 18].

More recently, a variant of what we will call for short the *Physarum dynamics* has been proposed for solving linear programming (LP) problems [12]. Such dynamics is a direct mathematical extension of the one that has been studied for the shortest path and transshipment problems. It was shown that, under very mild assumptions on the linear program, the dynamics converges to an optimal LP solution [19]. However, the bound for the time of convergence of a discretization of the dynamics to an approximate solution has only been proved to be polynomial when the LP cost coefficients are polynomially bounded and the constraint matrix has bounded maximum subdeterminant.

The contribution of this paper is to study the convergence time of the continuous Physarum dynamics in the context of the linear programming problem, and to derive a new time bound to approximate optimality that does not depend on the maximum subdeterminant of the constraint matrix, and depends only logarithmically on the LP costs, establishing efficiency for any LP instance with positive costs. The proof is based on convex duality and on a potential function that involves the relative entropy between the optimal and the current LP solution. The main technical novelty is represented by two ingredients: the use of a dimensionless potential function, and the explicit recognition of the crucial role played by the relative entropy function. We leave the study of a discretized version of the dynamics for future work, but it is natural to conjecture that some appropriate discretization should behave similarly to the continuous time dynamics. Indeed, several convex optimization methods can be interpreted as the discretization of an ordinary differential equation system, the solutions of which are guaranteed to converge to the set of minimizers; a well-known example is the interior point method [3, 13].

There is another compelling reason to study the convergence properties of the Physarum dynamics. It has been showed that, at least when started from a feasible point, this dynamics can be interpreted as a *natural gradient descent* algorithm in a space endowed with a non-Euclidean metric obtained from an entropy-like function [19]. This is also the case for certain incarnations of well-known meta-algorithms, such as Mirror Descent [5, 16], which are at the basis of very effective approximation algorithms for machine learning and convex optimization problems [2]. One can show that when the feasible LP region is the unit simplex, but independently of whether the initial point is feasible or not, the Physarum dynamics is identical to the continuous Mirror Descent dynamics of Nemirovski and Yudin [16] in the metric generated by the negative entropy function. Thus, when the feasible LP region is the simplex, the dynamics can be interpreted as a Mirror Descent method in a non-Euclidean metric [1, 10]. However, a similar connection may not hold more generally, suggesting that the Physarum dynamics is different from any known convex optimization method. A full characterization of the meta-algorithm behind the dynamics remains open; we believe it deserves to be investigated, due to its potential to suggest a novel iterative approach to linear optimization problems.

1.1 Linear programming and the Physarum dynamics

Let N and E be two finite index sets. Given a real matrix $\mathbf{A} \in \mathbb{R}^{N \times E}$, a positive vector $\mathbf{c} \in \mathbb{R}^E_{>0}$, and a vector $\mathbf{b} \in \mathbb{R}^N$, we consider the linear programming problem

$$\begin{aligned} &\min \operatorname{cost}(\mathbf{x}) \\ &\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b} \\ &\qquad \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbb{R}^E \end{aligned} \tag{1}$$

where $\operatorname{cost}(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{c}^\top \mathbf{x}$. We assume that $\mathbf{A}$ has full rank and that a nonzero optimal solution to (1) exists; uniqueness is not required. We denote by $\mathbf{x}^*$ an arbitrary optimal solution to (1), and denote by $\operatorname{opt} \stackrel{\text{def}}{=} \operatorname{cost}(\mathbf{x}^*)$ its value.

We describe the *(directed) Physarum dynamics* [4, 7, 8, 11, 18, 19, 21] that solves (1). Let $\mathbf{x} \in \mathbb{R}^E_{>0}$ be a positive vector, and let $\mathbf{C}$ be the diagonal matrix with entries x_j/c_j, for $j \in E$. Let $\mathbf{L} \stackrel{\text{def}}{=} \mathbf{A}\mathbf{C}\mathbf{A}^\top$; the matrix $\mathbf{L}$ is nonsingular and positive definite. Let $\mathbf{p} \in \mathbb{R}^N$ be the unique solution to $\mathbf{Lp} = \mathbf{b}$, and let $\mathbf{q} \overset{\text{def}}{=} \mathbf{CA}^\top \mathbf{p}$. The *Physarum dynamics* for the linear program (1) is

$$\dot{x}_j(t) = q_j(t) - x_j(t) \qquad \text{for all } j \in E \tag{2}$$

over the domain $\Omega \overset{\text{def}}{=} \mathbb{R}^E_{>0}$, where we used the notation $\dot{x}_j(t) \overset{\text{def}}{=} (d/dt)x_j(t)$. In vector notation, and omitting the implicit dependency on time, the Physarum dynamics can be written as

$$\dot{\mathbf{x}} = \mathbf{CA}^\top \mathbf{L}^{-1}\mathbf{b} - \mathbf{x}. \tag{3}$$

The dynamical system has an initial condition of the form $\mathbf{x}(0) = \mathbf{s}$ for some $\mathbf{s} \in \mathbb{R}^E_{>0}$. Existence of a solution $\mathbf{x}(t)$ to (3) for $t \in [0, \infty)$ has been proved by Straszak and Vishnoi [19, Theorem 1.1]. The system (3) is well-defined irrespective of whether the starting vector $\mathbf{x}(0)$ satisfies $\mathbf{Ax}(0) = \mathbf{b}$ or not; the case where it does is referred to as the *feasible start* case. In the special case where $\mathbf{A}$ is derived from the signed incidence matrix of a graph, problem (1) is a *minimum-cost transshipment problem* and several of the quantities defined above have an intuitive interpretation; we refer to Section 2.2 for details.

1.2 Our contribution

From previous results, it is known that the solution to (3) exists, and that it converges to a feasible and optimal solution of the linear program (1). The known bound on the convergence time, however, depends on the largest absolute value of a subdeterminant of the constraint matrix $\mathbf{A}$. Our main contribution is to show that, in the case of feasible start, this dependence is unnecessary, and that one can obtain a bound that only depends logarithmically on the ratio between the starting cost and the optimal cost, and on the relative entropy of the optimal solution with respect to the starting solution. More precisely, we prove the following theorem.

▶ **Theorem 1.** *For a feasible initial condition* $\mathbf{s} \in \mathbb{R}^E_{>0}$, *consider the solution* $\mathbf{x} : [0, \infty) \to \Omega$ *to the Physarum dynamics* (3) *with* $\mathbf{x}(0) = \mathbf{s}$. *Then* $\mathbf{x}(t)$ *is a feasible solution to* (1) *for any* $t \geq 0$, *and for any* $\epsilon > 0$, *it holds that* $\text{cost}(\mathbf{x}(t)) \leq (1 + \epsilon)\text{opt}$ *for all*

$$t \geq \frac{6}{\epsilon}\left(\ln \frac{\text{cost}(\mathbf{x}(0))}{\text{opt}} + \text{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(0))\right),$$

where $\text{KL}(\cdot, \cdot)$ *denotes the relative entropy (Kullback-Leibler divergence) between distributions, and* $\xi_j(0) \overset{\text{def}}{=} c_j x_j(0)/\text{cost}(\mathbf{x}(0))$, $\xi^*_j \overset{\text{def}}{=} c_j x^*_j/\text{opt}$ *for* $j \in E$. *In particular,* $\text{cost}(\mathbf{x}(t)) \leq (1 + \epsilon)\text{opt}$ *for all*

$$t \geq \frac{6}{\epsilon}\left(2 \ln \frac{\text{cost}(\mathbf{x}(0))}{\text{opt}} + \ln \mu\right),$$

where $\mu \overset{\text{def}}{=} \max_{j \in E} x^*_j / x_j(0)$.

We remark that our result applies to the continuous formulation of the dynamics, and not necessarily to its Euler discretization that has been considered, together with the continuous one, in previous papers. While we conjecture that *some* discretization may be similarly efficient as the bound in Theorem 1 suggests, it may also be the case that a simple Euler discretization is insufficient to obtain such a result and that a more accurate discretization technique, such as a Runge-Kutta method, would help in this sense.

The appearance of the relative entropy term in our potential function is not an accident: it can be shown that when the feasible LP region is the unit simplex, independently of whether the dynamics is initialized with a feasible point or not, its trajectories coincide with those of the continuous Mirror Descent method of Nemirovski and Yudin [16] in a metric with geometry dictated by the negative entropy function – also known as the *information geometry* metric [1].

1.3 Related work

An undirected variant of the Physarum dynamics has been first proposed in the mathematical biology literature by Tero, Kobayashi and Nakagaki [21] as a model for the foraging physiology of the true slime mold *Physarum polycephalum*, an acellular organism that has been proved capable of solving shortest path problems effectively in laboratory experiments [14]. The convergence to optimality of the continuous dynamics for the shortest path problem and for its close generalization – the minimum-cost transshipment problem – has been studied analytically by Bonifaci, Mehlhorn and Varma [8] and by Ito et al. [11]. An analysis of the convergence time of the Euler discretization of the dynamics was carried out by Becchetti et al. [4] for the shortest path problem, and by Straszak and Vishnoi [18] for the minimum-cost transshipment problem. In summary, these works proved that the Physarum dynamics yields a polynomial-time approximation scheme to the shortest path problem and to the transshipment problem, assuming that the costs associated to the edges of the network are polynomially bounded. Observe that, in the statement of Theorem 1, the costs are confined within logarithms, and thus a discrete version of the dynamics that achieved a similar convergence time as in Theorem 1 would *not* require the costs to be polynomially bounded to be efficient.

The generalization of the Physarum dynamics to linear programming problems that we consider here has been first suggested by Johannson and Zou [12]. Most relevant to the current paper is the work of Straszak and Vishnoi [19], who initiated the rigorous study of the Physarum dynamics for LP problems of the form (1). Straszak and Vishnoi proved that a solution to the dynamics exists over the entire time horizon $[0, \infty)$, and that a bound on the convergence time of the continuous dynamics can be expressed in terms of the parameter $\mathcal{D}$, the *largest absolute value of a subdeterminant* of the constraint matrix $\mathbf{A}$, as summarized by their theorem that we quote here for comparison.

▶ **Theorem 2.** *[19, Theorem 6.3] Suppose that $\mathbf{x} : [0, \infty) \to \Omega$ is any solution to the Physarum dynamics. Then, for some $R, \nu > 0$ depending only on $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{x}(0)$, we have*

$$|\mathrm{cost}(\mathbf{x}(t)) - \mathrm{opt}| \leq R \cdot \exp(-\nu t),$$

where one can take $\nu = \mathcal{D}^{-3}$ and $R = \exp(8\mathcal{D}^2 \cdot \|\mathbf{c}\|_1 \cdot \|\mathbf{b}\|_1) \cdot (|E| + M_x)^2$. Here,

$$\mathcal{D} \stackrel{\text{def}}{=} \max\{|\det(\mathbf{A}')| \,:\, \mathbf{A}' \textit{ a square submatrix of } \mathbf{A}\},$$

and

$$M_x \stackrel{\text{def}}{=} \max\left(\max_{j\in E} x_j(0), \max_{j\in E} x_j^{-1}(0)\right).$$

Compared to this result of Straszak and Vishnoi, our contribution is to derive a bound that avoids the dependency on $\mathcal{D}$, thus showing that the dynamics are efficient –to the extent made precise in the statement of Theorem 1– for all linear programs of the form (1), not just for those with special constraint matrices. Note that, in general, the bounds of Theorem 1 and 2 are incomparable: for a fixed relative error ϵ, the time for convergence guaranteed by Theorem 1 scales polynomially in the input encoding length, while Theorem 2 only yields an exponential dependence; on the other hand, for a fixed input, Theorem 2 achieves a polynomial dependence on $\log(1/\epsilon)$ (by taking $t \gg \mathcal{D}^3$), while this is $O(1/\epsilon)$ in Theorem 1. It is known that a simultaneous polynomial dependence on $\log(1/\epsilon)$ and on the input length cannot be achieved [18, Appendix B].

As mentioned in the introduction, several convex optimization methods can be interpreted as discretizations of ordinary differential equation systems: for example, the Interior Point method [3, 13] and the Mirror Descent method [16, Chapter 3]. Straszak and Vishnoi [19] proved that the Physarum dynamics with feasible start can be interpreted as natural gradient descent in an appropriate information metric. Amari [1] gives an overview of natural gradient methods in the context of information geometry; see also Raskutti and Mukherjee [17].

1.4 Organization of the paper

The remainder of the paper is organized as follows. In Section 2 we prove some basic facts about the Physarum dynamics, including an alternative characterization of the vector $\mathbf{q} \in \mathbb{R}^E$ defined in Section 1.1. In Section 3 we discuss the time of convergence to the feasible region of the LP and prove that the set of feasible LP solutions is an invariant set for the dynamics. In Section 4 we consider the time evolution of the cost of a feasible solution and prove our main result, Theorem 1. We summarize and discuss our findings in Section 5.

2 Basic properties of the dynamics

2.1 Notation

In the paper we reserve boldface symbols for vectors or matrices and non-boldface symbols for scalars or sets. We use the standard norms: for example, for $\mathbf{v} \in \mathbb{R}^n$: $\|\mathbf{v}\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^n |v_i|$, $\|\mathbf{v}\|_2 \stackrel{\text{def}}{=} (\sum_{i=1}^n v_i^2)^{1/2}$. With the notation $\mathbf{Diag}((d_i)_{i=1}^n)$ we mean the $n \times n$ diagonal matrix with d_i as the ith term on the main diagonal.

For the whole paper, the linear program (1) is fixed, in other words the triple $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ is fixed. Whenever the matrices or vectors $\mathbf{C} = \mathbf{Diag}((x_j/c_j)_{j\in E})$, $\mathbf{L} = \mathbf{ACA}^\top$, $\mathbf{p} = \mathbf{L}^{-1}\mathbf{b}$, or $\mathbf{q} = \mathbf{CA}^\top\mathbf{p}$ appear, they should be understood as computed with respect to a point $\mathbf{x} \in \mathbb{R}^E_{>0}$. As $\mathbf{x} = \mathbf{x}(t)$ evolves in time with the dynamics (3), the former quantities are time-varying as well. The quantity $\mathcal{E} \stackrel{\text{def}}{=} \mathbf{q}^\top\mathbf{R}\mathbf{q}$ is called the *energy* of the vector $\mathbf{q}$.

For a strictly convex and differentiable function $\psi : \Omega \to \mathbb{R}$, the *Bregman divergence* under ψ is the function

$$D_\psi(\mathbf{x}', \mathbf{x}) \stackrel{\text{def}}{=} \psi(\mathbf{x}') - \psi(\mathbf{x}) - \nabla\psi(\mathbf{x})^\top \cdot (\mathbf{x}' - \mathbf{x}),$$

where $\mathbf{x}', \mathbf{x} \in \Omega$. The Bregman divergence is in general not symmetric, but it is nonnegative and satisfies $D_\psi(\mathbf{x}', \mathbf{x}) = 0$ if $\mathbf{x}' = \mathbf{x}$. The *Legendre dual* of ψ is the function $\psi^\star : \mathbb{R}^E \to \mathbb{R}$ defined by

$$\psi^\star(\mathbf{y}) \stackrel{\text{def}}{=} \sup_{\mathbf{x}\in\mathbb{R}^E} (\mathbf{x}^\top\mathbf{y} - \psi(\mathbf{x})),$$

Note that a vector $\mathbf{x} \in \Omega$ is a maximizer of $\mathbf{x}^\top\mathbf{y} - \psi(\mathbf{x})$ iff $\mathbf{y} = \nabla\psi(\mathbf{x})$.

2.2 Intuition: The network case

The interpretation of the dynamics defined in Section 1.1 in the case where the constraint matrix is a network matrix is particularly appealing, as most of the statements below have physical interpretations in that case. Indeed, when $\mathbf{A}$ is derived from the (signed) node-edge incidence matrix of a graph with node set N and edge set E [1], the dynamics (3) have a natural interpretation in terms of electrical networks: the vector $\mathbf{b}$ prescribes the external in-flow of current at each node, the matrix $\mathbf{L}$ is the (reduced) graph Laplacian, the vector $\mathbf{p}$ defines the Kirchhoff node potentials, the vector $\mathbf{q}$ is the electrical flow, and $\mathcal{E}$ is the energy dissipation (per unit time) of the network. In this context, Lemma 3 below is nothing but the *principle of least action* for electrical networks, also known as Thomson's principle, stating that the electrical flow is the feasible flow that minimizes energy dissipation [6, Theorem IX.2]. The duality relation (5) becomes Ohm's law. Proposition 4 is the conservation of energy principle, stating that if one replaces a network with a current source s and a sink $\bar{s}$ with a single wire whose resistance is the effective resistance of the network, then the total energy in the system does not change [6, Theorem IX.3]. Proposition 5 is known as Tellegen's theorem. Of course, the difference with classical circuit theory is that the resistor values are dynamically adjusted in response to the flow: the Physarum dynamics adjusts the edges' resistances c_j/x_j, by updating the x_j via (2). In the network case, the dynamics converges to the solution of a minimum-cost transshipment problem with cost function prescribed by $\mathbf{c}$ and node demands/supplies prescribed by $\mathbf{b}$ (see for example [18, Theorem 1.2]). However, we remark that in the following statements we never require $\mathbf{A}$ to be derived from a network matrix: our results hold for any full-rank matrix.

2.3 Extremal properties

We start by giving an alternative characterization of the vector $\mathbf{q} \overset{\text{def}}{=} \mathbf{C}\mathbf{A}^\top\mathbf{L}^{-1}\mathbf{b}$.

▶ **Lemma 3.** *The vector $\mathbf{q} \in \mathbb{R}^E$ defined in Section 1.1 equals the unique optimal solution to the continuous quadratic optimization problem:*

$$\begin{aligned} &\min\ \mathbf{f}^\top\mathbf{R}\mathbf{f} \\ &\ \textit{s.t.}\ \ \mathbf{A}\mathbf{f} = \mathbf{b}. \end{aligned} \tag{4}$$

where $\mathbf{R} \overset{\text{def}}{=} \mathbf{C}^{-1} \in \mathbb{R}^{E\times E}$ is the diagonal matrix with value $r_j \overset{\text{def}}{=} c_j/x_j$ for the j-th element of the main diagonal. Moreover,

$$\mathbf{R}\mathbf{q} = \mathbf{A}^\top\mathbf{p}. \tag{5}$$

Proof. To establish (5), simply left-multiply with $\mathbf{R}$ the identity $\mathbf{q} = \mathbf{C}\mathbf{A}^\top\mathbf{p}$. It remains to establish the first part of the claim. Since the objective function in (4) is strictly convex, the problem has a unique optimal point. Consider any feasible point $\mathbf{f}$, and define $\mathbf{g} = \mathbf{f} - \mathbf{q}$. Then $\mathbf{A}\mathbf{g} = \mathbf{b} - \mathbf{b} = \mathbf{0}$ and hence

$$\mathbf{f}^\top\mathbf{R}\mathbf{f} = (\mathbf{q}+\mathbf{g})^\top\mathbf{R}(\mathbf{q}+\mathbf{g}) = \mathbf{q}^\top\mathbf{R}\mathbf{q} + 2\mathbf{g}^\top\mathbf{R}\mathbf{q} + \mathbf{g}^\top\mathbf{R}\mathbf{g} \geq \mathbf{q}^\top\mathbf{R}\mathbf{q},$$

since $\mathbf{g}^\top\mathbf{R}\mathbf{g} \geq 0$ and $\mathbf{g}^\top\mathbf{R}\mathbf{q} = \mathbf{g}^\top\mathbf{A}^\top\mathbf{p} = (\mathbf{A}\mathbf{g})^\top\mathbf{p} = \mathbf{0}^\top\mathbf{p} = 0$. Therefore, the objective function value of any feasible point $\mathbf{f}$ is at least as large as the objective function value of $\mathbf{q}$. ◀

[1] More precisely, since we stipulated that $\mathbf{A}$ should be full rank, we omit from N one of the nodes and omit the corresponding row from $\mathbf{A}$; this corresponds to "grounding" the potential value of this node to zero.

▶ **Proposition 4.** $\mathcal{E} = \mathbf{b}^\top \mathbf{L}^{-1} \mathbf{b} = \mathbf{p}^\top \mathbf{L} \mathbf{p}$.

Proof. $\mathbf{q}^\top \mathbf{R} \mathbf{q} = (\mathbf{b}^\top \mathbf{L}^{-1} \mathbf{A} \mathbf{C}) \mathbf{R} (\mathbf{C} \mathbf{A}^\top \mathbf{L}^{-1} \mathbf{b}) = (\mathbf{b}^\top \mathbf{L}^{-1})(\mathbf{A} \mathbf{C} \mathbf{A}^\top)(\mathbf{L}^{-1} \mathbf{b}) =$
$= \mathbf{p}^\top \mathbf{L} \mathbf{p}$. ◀

▶ **Proposition 5.** *Let* $\mathbf{f}$ *satisfy* $\mathbf{A}\mathbf{f} = \mathbf{b}$. *Then*

$$\mathbf{f}^\top \mathbf{A}^\top \mathbf{p}(t) = \mathcal{E}(t). \tag{6}$$

Proof. Since $\mathbf{A}\mathbf{f} = \mathbf{b}$, we have $\mathbf{p}^\top \mathbf{A} \mathbf{f} = \mathbf{p}^\top \mathbf{b} = \mathbf{p}^\top \mathbf{L} \mathbf{p} = \mathcal{E}$. The last equality is due to Proposition 4. ◀

3 Convergence to the feasible region

In this section we discuss the time of convergence to the feasible region $\mathbf{A}\mathbf{x} = \mathbf{b}$. In particular, we aim to show that feasibility is invariant under the dynamics: a feasible starting point remains feasible at all times. It turns out that a stronger property holds: the Euclidean norm of the "infeasibility error" $\mathbf{e} \overset{\text{def}}{=} \mathbf{A}\mathbf{x} - \mathbf{b}$ approaches zero exponentially fast (Lemma 7).

▶ **Proposition 6.** $\mathbf{A}\dot{\mathbf{x}} = \mathbf{b} - \mathbf{A}\mathbf{x}$.

Proof. Using the definition of the dynamics (3), $\mathbf{A}\dot{\mathbf{x}} = \mathbf{A}\mathbf{C}\mathbf{A}^\top \mathbf{L}^{-1}\mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{L}\mathbf{L}^{-1}\mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{b} - \mathbf{A}\mathbf{x}$. ◀

▶ **Lemma 7.** *Let* $\mathbf{e}(t) \overset{\text{def}}{=} \mathbf{A}\mathbf{x}(t) - \mathbf{b}$. *Then* $\|\mathbf{e}(t)\|_2 = \|\mathbf{e}(0)\|_2 \exp(-t)$ *for any* $t > 0$. *In particular, if* $\mathbf{A}\mathbf{x}(0) = \mathbf{b}$ *then* $\mathbf{A}\mathbf{x}(t) = \mathbf{b}$ *for all* $t > 0$.

Proof. We have

$$\begin{aligned}\frac{d}{dt}\|\mathbf{e}\|_2^2 &= \frac{d}{dt}(\mathbf{A}\mathbf{x} - \mathbf{b})^\top(\mathbf{A}\mathbf{x} - \mathbf{b}) = \frac{d}{dt}\left(\mathbf{x}^\top \mathbf{A}^\top \mathbf{A}\mathbf{x} - 2\mathbf{b}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{b}\right) = \\ &= 2\mathbf{x}^\top \mathbf{A}^\top \mathbf{A}\dot{\mathbf{x}} - 2\mathbf{b}^\top \mathbf{A}\dot{\mathbf{x}} = 2(\mathbf{x}^\top \mathbf{A}^\top - \mathbf{b}^\top)\mathbf{A}\dot{\mathbf{x}} = -2\|\mathbf{e}\|_2^2,\end{aligned}$$

where we used Proposition 6. Solution of the differential equation above yields $\|\mathbf{e}(t)\|_2^2 = \|\mathbf{e}(0)\|_2^2 \exp(-2t)$. Taking square roots yields the claim. ◀

4 Convergence in cost value

To analyze the convergence in cost values, and eventually prove Theorem 1, it will be useful to consider normalized versions of the candidate solution $\mathbf{x}(t)$ and of the optimal vector $\mathbf{x}^*$. For any $j \in E$, let $\xi_j(t) \overset{\text{def}}{=} c_j x_j(t)/\text{cost}(\mathbf{x}(t))$, $\xi_j^* \overset{\text{def}}{=} c_j x_j^*/\text{opt}$. Then, by construction, $\mathbf{1}^\top \boldsymbol{\xi}^* = \mathbf{1}^\top \boldsymbol{\xi}(t) = 1$, $\boldsymbol{\xi}(t) > \mathbf{0}$ and $\boldsymbol{\xi}^* \geq \mathbf{0}$, so $\boldsymbol{\xi}(t)$ and $\boldsymbol{\xi}^*$ can be interpreted as probability distributions over E. The *relative entropy* of $\boldsymbol{\xi}^*$ with respect to $\boldsymbol{\xi}$, or *Kullback-Leibler divergence* $\text{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(t))$, is defined as:

$$\text{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(t)) \overset{\text{def}}{=} \sum_{j \in E} \xi_j^* \ln \frac{\xi_j^*}{\xi_j(t)}.$$

The KL divergence is the Bregman divergence of the negative entropy function $\mathbf{x} \mapsto \sum_j x_j \ln x_j$; it is always nonnegative, and it is zero iff $\boldsymbol{\xi}^* = \boldsymbol{\xi}(t)$ (see for example [1, Chapter 1]).

We can now define the potential function that is central to our analysis. Let

$$\Phi(t) \stackrel{\text{def}}{=} \ln \frac{\text{cost}(\mathbf{x}(t))}{\text{opt}} + \text{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(t)). \tag{7}$$

Note that the first term is nonnegative whenever $\mathbf{x}(t)$ is feasible for the LP, and the second term is always nonnegative. Similar to previous analysis of Physarum dynamics based on potential functions [4,18,19], the potential function Φ contains a term that depends on the cost of the candidate solution $\mathbf{x}$, and an "entropic barrier" term that captures the geometry of the feasible region: in particular, the second term penalizes distributions that get too close to the boundary of the positive orthant whenever the corresponding coordinate of the optimal solution is not on the boundary (that is, $\xi_j(t) \approx 0$ but $\xi_j^* > 0$). A difference with respect to previous papers is that the potential function (7) is dimensionless, which is natural since our aim is to bound the relative, rather than absolute, approximation error.

To proceed further, we study the evolution of the potential function over time. We start by bounding the derivative of various terms that compose it.

▶ **Lemma 8.** *For any* $\mathbf{x}(t) \in \mathbb{R}^E_{>0}$,

$$\frac{d}{dt} \ln \frac{\text{cost}(\mathbf{x}(t))}{\text{opt}} \leq \left(\frac{\mathcal{E}(t)}{\text{cost}(\mathbf{x}(t))} \right)^{1/2} - 1. \tag{8}$$

Proof.

$$\begin{aligned}
\frac{d}{dt} \ln \frac{\text{cost}(\mathbf{x})}{\text{opt}} &= \frac{\frac{d}{dt}\text{cost}(\mathbf{x})}{\text{cost}(\mathbf{x})} = \frac{\mathbf{c}^\top \dot{\mathbf{x}}}{\text{cost}(\mathbf{x})} \\
&\stackrel{(2)}{=} \frac{\mathbf{c}^\top (\mathbf{q} - \mathbf{x})}{\text{cost}(\mathbf{x})} = \frac{\mathbf{c}^\top \mathbf{q}}{\text{cost}(\mathbf{x})} - 1 \\
&\stackrel{(*)}{=} \frac{\sum_{j \in E} r_j q_j x_j}{\text{cost}(\mathbf{x})} - 1 \\
&\stackrel{(**)}{\leq} \frac{\left(\sum_{j \in E} r_j q_j^2\right)^{1/2} \left(\sum_{j \in E} r_j x_j^2\right)^{1/2}}{\text{cost}(\mathbf{x})} - 1 \\
&= \frac{(\mathcal{E}\text{cost}(\mathbf{x}))^{1/2}}{\text{cost}(\mathbf{x})} - 1,
\end{aligned}$$

where in the third equality we used the definition of the dynamics, in (*) we used $r_j = c_j / x_j$, and in (**) we used the Cauchy-Schwarz inequality. For the last equality, we used the definition of the energy $\mathcal{E} = \mathbf{q}^\top \mathbf{R} \mathbf{q}$ and (once more) the fact $r_j = c_j / x_j$. ◀

The following lemma is instrumental in bounding the time derivative of the KL divergence term in (7).

▶ **Lemma 9.** *For any* $\mathbf{x}(t) \in \mathbb{R}^E_{>0}$,

$$\frac{d}{dt} \sum_{j \in E} \frac{c_j x_j^*}{\text{opt}} \ln \frac{x_j^*}{x_j(t)} = 1 - \frac{\mathcal{E}(t)}{\text{opt}}. \tag{9}$$

Proof. We start by computing

$$\sum_j \frac{c_j}{\text{opt}} x_j^* \ln \frac{x_j^*}{x_j} = -\frac{1}{\text{opt}} \sum_j c_j x_j^* \ln x_j + \frac{1}{\text{opt}} \sum_j c_j x_j^* \ln x_j^*.$$

The second term above is constant, so

$$\begin{aligned}\frac{d}{dt}\sum_j \frac{c_j}{\mathrm{opt}} x_j^* \ln\frac{x_j^*}{x_j} &= -\frac{1}{\mathrm{opt}}\sum_j c_j x_j^* \frac{\dot{x}_j}{x_j} = \\ &= -\frac{1}{\mathrm{opt}}\sum_j c_j x_j^* \frac{q_j - x_j}{x_j} = \\ &= 1 - \frac{1}{\mathrm{opt}}\sum_j r_j q_j x_j^* = \\ &= 1 - \frac{1}{\mathrm{opt}}\mathbf{x}^{*\top}\mathbf{R}\mathbf{q} = \\ &\overset{(5)}{=} 1 - \frac{1}{\mathrm{opt}}\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{p} = \\ &\overset{(6)}{=} 1 - \frac{\mathcal{E}}{\mathrm{opt}}.\end{aligned}$$

We used (5) (Lemma 3) and the alternative characterization of the energy (6) given by Proposition 5. ◀

▶ **Lemma 10.** *For any* $\mathbf{x}(t) \in \mathbb{R}^E_{>0}$,

$$\frac{d}{dt}\mathrm{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(t)) \le \left(\frac{\mathcal{E}(t)}{\mathrm{cost}(\mathbf{x}(t))}\right)^{1/2} - \frac{\mathcal{E}(t)}{\mathrm{opt}}.$$

Proof.

$$\begin{aligned}\frac{d}{dt}\sum_j \frac{c_j x_j^*}{\mathrm{opt}} \ln\frac{c_j x_j^*/\mathrm{opt}}{c_j x_j/\mathrm{cost}(\mathbf{x})} &= \frac{d}{dt}\sum_j \frac{c_j x_j^*}{\mathrm{opt}} \ln\frac{x_j^*}{x_j} + \frac{d}{dt}\sum_j \frac{c_j x_j^*}{\mathrm{opt}} \ln\frac{\mathrm{cost}(\mathbf{x})}{\mathrm{opt}} \\ &\overset{(9)}{=} 1 - \frac{\mathcal{E}}{\mathrm{opt}} + \frac{d}{dt}\ln\frac{\mathrm{cost}(\mathbf{x})}{\mathrm{opt}} \cdot 1 \\ &\overset{(8)}{\le} 1 - \frac{\mathcal{E}}{\mathrm{opt}} + \left(\frac{\mathcal{E}}{\mathrm{cost}(\mathbf{x})}\right)^{1/2} - 1 \\ &= \left(\frac{\mathcal{E}}{\mathrm{cost}(\mathbf{x})}\right)^{1/2} - \frac{\mathcal{E}}{\mathrm{opt}}.\end{aligned}$$

In the second equality we used Lemma 9; for the inequality we applied Lemma 8. ◀

We are ready to prove that the more expensive a solution is, the more the potential function has to decrease.

▶ **Lemma 11.** *If* $\mathrm{cost}(\mathbf{x}(t)) \ge (1+\epsilon)^2\mathrm{opt}$ *for some* $\epsilon \in (0, 1/2)$, *then* $(d/dt)\Phi(t) \le -\epsilon/2$.

Proof. Let $\gamma \overset{\text{def}}{=} \mathcal{E}/\mathrm{opt}$ and $\delta \overset{\text{def}}{=} 1/(1+\epsilon)$. Combining Lemma 8 and Lemma 10 yields

$$\begin{aligned}\frac{d}{dt}\Phi(t) &= 2\left(\frac{\mathcal{E}}{\mathrm{cost}(\mathbf{x})}\right)^{1/2} - 1 - \frac{\mathcal{E}}{\mathrm{opt}} \\ &\overset{(*)}{\le} 2\delta\gamma^{1/2} - 1 - \gamma = \\ &= -2(1-\delta)\gamma^{1/2} - (1-\gamma^{1/2})^2,\end{aligned}$$

where in $(*)$ we used the assumption $\mathrm{cost}(\mathbf{x}) \geq (1+\epsilon)^2\mathrm{opt}$. Note that both summands in the last expression are negative. We distinguish two cases. If $(1-\gamma^{1/2})^2 \geq \epsilon/2$, then by ignoring the first summand above we obtain

$$\frac{d}{dt}\Phi \leq -(1-\gamma^{1/2})^2 \leq -\epsilon/2,$$

which proves the claim. Otherwise, if $(1-\gamma^{1/2})^2 < \epsilon/2$, then $\gamma^{1/2} > 1-(\epsilon/2)^{1/2}$ and by ignoring the second summand we obtain

$$\frac{d}{dt}\Phi \leq -2(1-\delta)(1-(\epsilon/2)^{1/2}) \leq -2\cdot\frac{1-(\epsilon/2)^{1/2}}{1+\epsilon}\epsilon \leq -2\frac{1/2}{3/2}\epsilon < -\epsilon/2. \qquad \blacktriangleleft$$

The next lemma ensures that, for feasible solutions, the energy is always a valid lower bound on the cost.

▶ **Lemma 12.** *Suppose* $\mathbf{x}(t) \geq 0$*,* $\mathbf{A}\mathbf{x}(t) = \mathbf{b}$*. Then* $\mathcal{E}(t) \leq \mathrm{cost}(\mathbf{x}(t))$*.*

Proof. By the assumption, $\mathbf{x}(t)$ is a feasible LP solution. By Lemma 3, $\mathbf{q}(t)$ is a minimizer of the quadratic form $\mathbf{f}^\top\mathbf{R}\mathbf{f}$ among all vectors $\mathbf{f}$ satisfying $\mathbf{A}\mathbf{f} = \mathbf{b}$. One possible such vector is $\mathbf{x}$. Thus,

$$\mathcal{E} = \mathbf{q}^\top\mathbf{R}\mathbf{q} \leq \mathbf{x}^\top\mathbf{R}\mathbf{x} = \sum_{j\in E}\frac{c_j}{x_j}x_j^2 = \mathrm{cost}(\mathbf{x}). \tag{10}$$

◀

As a corollary, by Lemma 8 the cost of a feasible solution does not increase over time.

▶ **Corollary 13.** *Suppose* $\mathbf{x}(t) \geq 0$*,* $\mathbf{A}\mathbf{x}(t) = \mathbf{b}$*. Then* $(d/dt)\mathrm{cost}(\mathbf{x}(t)) \leq 0$*.*

Proof. Combine Lemma 12 and Lemma 8. ◀

All ingredients are now into place to derive our main claim, from which Theorem 1 will directly follow.

▶ **Theorem 14.** *Suppose* $\mathbf{x}(0) > 0$*,* $\mathbf{A}\mathbf{x}(0) = \mathbf{b}$*. Then*
(a) $\mathbf{x}(t)$ *is feasible for LP* (1) *for any* $t \geq 0$*;*
(b) $\mathrm{cost}(\mathbf{x}(t)) \leq (1+\epsilon)\mathrm{opt}$ *for all*

$$t \geq \frac{\Phi(0)}{\epsilon/6} = \frac{6}{\epsilon}\left(\ln\frac{\mathrm{cost}(\mathbf{x}(0))}{\mathrm{opt}} + \mathrm{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(0))\right).$$

Proof. By assumption, we start with a feasible initial solution $\mathbf{x}(0)$, thus by Lemma 7 the solution $\mathbf{x}(t)$ stays feasible for all $t \geq 0$; this proves point (a). By Corollary 13, the cost of $\mathbf{x}(t)$ can only decrease as t increases. To prove point (b), assume, by contradiction, that $\mathrm{cost}(\mathbf{x}(t_0))$ is larger than $(1+\epsilon)\mathrm{opt}$ for some t_0 that is larger than $\Phi(0)/(\epsilon/6)$. By Lemma 11, $(d/dt)\Phi(t) \leq -\epsilon/6$ for all t such that

$$\mathrm{cost}(\mathbf{x}(t)) \geq (1+\epsilon)\mathrm{opt} = (1+3\epsilon')\mathrm{opt} \geq (1+\epsilon')^2\mathrm{opt},$$

where $\epsilon' \overset{\mathrm{def}}{=} \epsilon/3$. In particular, $(d/dt)\Phi(t) \leq -\epsilon/6$ would hold for all $t \in [0, t_0]$. This implies the desired contradiction, since $\Phi(t_0) = \Phi(0) + \int_0^{t_0}\frac{d}{dt}\Phi(t) \leq \Phi(0) - (\epsilon/6)t_0$ would have to be negative. This is impossible since $\mathbf{x}(t)$ is feasible at all times and thus $\Phi(t)$ is nonnegative for all t. ◀

Proof of Theorem 1. Theorem 14 already proves the first part of Theorem 1. For the second part, observe that if

$$\mu \overset{\text{def}}{=} \max_{j \in E} \frac{x_j^*}{x_j(0)},$$

then

$$\begin{aligned}
\mathrm{KL}(\boldsymbol{\xi}^*, \boldsymbol{\xi}(t)) &= \sum_{j \in E} \frac{c_j x_j^*}{\mathrm{opt}} \ln \frac{c_j x_j^*/\mathrm{opt}}{c_j x_j/\mathrm{cost}(\mathbf{x})} \\
&= \sum_j \frac{c_j x_j^*}{\mathrm{opt}} \ln \frac{x_j^*}{x_j} + \sum_j \frac{c_j x_j^*}{\mathrm{opt}} \ln \frac{\mathrm{cost}(\mathbf{x})}{\mathrm{opt}} \\
&\le (\ln \mu) \cdot \sum_j \frac{c_j x_j^*}{\mathrm{opt}} + \left(\ln \frac{\mathrm{cost}(\mathbf{x})}{\mathrm{opt}} \right) \cdot \sum_j \frac{c_j x_j^*}{\mathrm{opt}} \\
&= \ln \mu + \ln \frac{\mathrm{cost}(\mathbf{x})}{\mathrm{opt}}.
\end{aligned}$$

Substitution in the bound of Theorem 14 yields the claim. ◀

5 Discussion

We have shown that the Physarum dynamics converges fast for LP instances with positive costs when starting from a feasible point. More precisely, the convergence is inversely proportional in time and logarithmic on the ratio between the initial cost and the optimal one, and the ratio between coordinates of the initial vector and the optimal solution. This result avoids all dependence on the coefficients of the constraint matrix, as opposed to a previous bound which was polynomial in the maximum subdeterminant of this matrix.

We were able to study only the continuous variant of the dynamics and did not derive bounds for the discretized dynamics that could be deduced from it. However, the fact that the continuous dynamics has desirable properties and converges fast is often a solid indication that the resulting discrete algorithm might work well. Clearly, establishing this formally is a nontrivial task and improving the bounds of Straszak and Vishnoi [19] for the discrete variant remains an important open question in this setting. Moreover, our argument relied on the assumption of a feasible starting point, which is most likely not required by the result. The main obstacle, from this point of view, is to appropriately replace Lemma 12.

We also observe that the dependence of accuracy in time has been proved to be at most of order t^{-1} in our analysis, whereas gradient methods for linear programming typically have worse bounds, of the order of $t^{-1/2}$. It would be interesting to know if such an improved rate can be maintained when performing a time-discretization of the dynamics. From a broader perspective, as pointed out in the introduction, a full characterization of the meta-algorithm behind the Physarum dynamics remains open.

Acknowledgments. The author would like to thank Kurt Mehlhorn for suggesting a shorter proof of Lemma 3.

References

1 S. Amari. *Information Geometry and Its Applications.* Springer, 2016.

2 S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

3 D. A. Bayer and J. C. Lagarias. The nonlinear geometry of linear programming, I. Affine and projective scaling trajectories. *Trans. of the American Mathematical Society*, 314:499–526, 1989.

4 L. Becchetti, V. Bonifaci, M. Dirnberger, A. Karrenbauer, and K. Mehlhorn. Physarum can compute shortest paths: Convergence proofs and complexity bounds. In *Proc. of the 40th Int. Colloquium on Automata, Languages and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 472–483. Springer, 2013.

5 A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Oper. Res. Lett.*, 31(3):167–175, 2003.

6 B. Bollobás. *Modern Graph Theory.* Springer, New York, 1998.

7 V. Bonifaci. Physarum can compute shortest paths: A short proof. *Inf. Process. Lett.*, 113(1-2):4–7, 2013.

8 V. Bonifaci, K. Mehlhorn, and G. Varma. Physarum can compute shortest paths. In *Proc. of the 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 233–240. SIAM, 2012.

9 B. Chazelle. Natural algorithms and influence systems. *Communications of the ACM*, 55(12):101–110, 2012.

10 J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics.* Cambridge University Press, 1998.

11 K. Ito, A. Johansson, T. Nakagaki, and A. Tero. Convergence properties for the Physarum solver. arXiv:1101.5249v1, Jan 2011.

12 A. Johannson and J. Y. Zou. A slime mold solver for linear programming problems. In *How the World Computes - Turing Centenary Conference and 8th Conference on Computability in Europe*, pages 344–354. Springer, 2012.

13 N. K. Karmarkar. Riemannian geometry underlying interior–point methods for linear programming. In J. C. Lagarias and M. J. Todd, editors, *Mathematical Developments Arising from Linear Programming*, volume 114 of *Contemporary Mathematics*, pages 51–75. American Mathematical Society, 1990.

14 T. Nakagaki, H. Yamada, and Á. Tóth. Maze-solving by an amoeboid organism. *Nature*, 407:470, 2000.

15 S. Navlakha and Z. Bar-Joseph. Algorithms in nature: the convergence of systems biology and computational thinking. *Molecular Systems Biology*, 7:546, 2011.

16 A. S. Nemirovski and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization.* John Wiley, 1983.

17 G. Raskutti and S. Mukherjee. The information geometry of mirror descent. *IEEE Trans. Information Theory*, 61(3):1451–1457, 2015.

18 D. Straszak and N. K. Vishnoi. Natural algorithms for flow problems. In *Proc. of the 27th ACM-SIAM Symposium on Discrete Algorithms*, pages 1868–1883. SIAM, 2016.

19 D. Straszak and N. K. Vishnoi. On a natural dynamics for linear programming. In *Proc. of the 2016 ACM Conf. on Innovations in Theoretical Computer Science*, page 291. ACM, 2016.

20 A. Tero, R. Kobayashi, and T. Nakagaki. Physarum solver: A biologically inspired method of road-network navigation. *Physica A*, 363:115–119, 2006.

21 A. Tero, R. Kobayashi, and T. Nakagaki. A mathematical model for adaptive transport network in path finding by true slime mold. *Journal of Theoretical Biology*, 244:553–564, 2007.

22 A. Tero, S. Takagi, T. Saigusa, K. Ito, D. P. Bebber, M. D. Fricker, K. Yumiki, R. Kobayashi, and T. Nakagaki. Rules for biologically inspired adaptive network design. *Science*, 327:439–442, 2010.

Routing on the Visibility Graph

Prosenjit Bose[*1], **Matias Korman**[†2], **André van Renssen**[‡3], **and Sander Verdonschot**[§4]

1 **School of Computer Science, Carleton University, Ottawa, Canada**
`jit@scs.carleton.ca`

2 **Tohoku University, Sendai, Japan**
`mati@dais.is.tohoku.ac.jp`

3 **National Institute of Informatics, Tokyo and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
`andre@nii.ac.jp`

4 **School of Computer Science, Carleton University, Ottawa, Canada**
`sander@cg.scs.carleton.ca`

Abstract

We consider the problem of routing on a network in the presence of line segment constraints (i.e., obstacles that edges in our network are not allowed to cross). Let P be a set of n points in the plane and let S be a set of non-crossing line segments whose endpoints are in P. We present two deterministic 1-local $O(1)$-memory routing algorithms that are guaranteed to find a path of at most linear size between any pair of vertices of the *visibility graph* of P with respect to a set of constraints S (i.e., the algorithms never look beyond the direct neighbours of the current location and store only a constant amount of information). Contrary to *all* existing deterministic local routing algorithms, our routing algorithms do not route on a plane subgraph of the visibility graph.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Routing, constraints, visibility graph, Θ-graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.18

1 Introduction

Routing is a fundamental problem in networking. The goal is to find a path from a source vertex to a destination vertex in the network. When the whole network is known to the routing algorithm, there exist many algorithms to find paths. The problem is more challenging when the only information available is the location of the current vertex, its neighbours and a constant amount of additional information (such as the source and destination vertex). This is often referred to as *local* routing (or k-local for some constant k, when the k-neighbourhood is considered). In our setting, we assume that the network is a graph embedded in the plane, with edges being straight line segments connecting pairs of vertices, weighted by the Euclidean distance between their endpoints. Algorithms routing on such networks are referred to as *geometric* routing algorithms (see [7] and [8] for surveys of the area).

[*] P. B. is supported in part by NSERC.
[†] M. K. was partially supported by MEXT KAKENHI Nos. 15H02665, and 17K12635.
[‡] A. v. R. was supported by JST ERATO Grant Number JPMJER1201, Japan.
[§] S. V. is supported in part by NSERC and the Carleton-Fields Postdoctoral Award.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 18; pp. 18:1–18:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Deterministic routing algorithms that guarantee delivery in these networks typically route on plane subgraphs of the complete Euclidean graph. This means that of the potentially quadratic number of edges available to the routing algorithm, only a linear number are ever considered. This forces these algorithms to use paths that are much longer than necessary. In this paper, we present the first deterministic local routing algorithm that considers more edges by not restricting its choices to a plane subgraph.

Moreover, we study routing algorithms in a more general setting. In certain cases, some edges of a network may not be usable if for example there is a large obstacle blocking direct communication between two nodes. We model this impossibility via a set S of non-intersecting *line segment constraints* whose endpoints are vertices of the network. Given a set P of n points in the plane and a set S of non-intersecting line segment constraints, we say that two vertices u and v can *see each other* provided that either the line segment uv does not properly intersect any constraint in S or uv is itself a constraint in S. If two vertices u and v can see each other, the line segment uv is referred to as a *visibility edge*. The *visibility graph* of P with respect to a set of constraints S, denoted $\mathit{Vis}(P, S)$, has P as vertex set and all visibility edges as edge set. In other words, $\mathit{Vis}(P, S)$ is the complete graph on P minus all edges that properly intersect one or more constraints in S.

Although this setting has been studied extensively in the context of motion planning amid obstacles ([5, 6, 1, 4]), there has not been much work on routing in this setting. Bose *et al.* [2] showed that it is possible to route locally and 2-competitively between any two visible vertices in the constrained Θ_6-graph. Additionally, an 18-competitive routing algorithm between any two visible vertices in the constrained half-Θ_6-graph was provided. In the same paper it was shown that no deterministic local routing algorithm is $o(\sqrt{n})$-competitive between all pairs of vertices of the constrained Θ_6-graph, regardless of the amount of memory it is allowed to use. Recently, the authors presented a non-competitive 1-local $O(1)$-memory routing algorithm to route on the visibility graph by determining locally the edges of the constrained half-Θ_6-graph [3], a plane subgraph of $\mathit{Vis}(P, S)$.

We present two deterministic 1-local $O(1)$-memory routing algorithms on $\mathit{Vis}(P, S)$. The first algorithm locally computes a non-plane subgraph of the visibility graph (the constrained Θ_6-graph) and routes on it. We then modify this algorithm to obtain a routing algorithm that routes directly on the visibility graph. To the best of our knowledge, this is the first local routing algorithm does not compute a plane subgraph of the visibility graph.

2 Preliminaries

The Θ_m-graph plays an important role in our routing strategy. We begin by defining it. Define a *cone* C to be the region in the plane between two rays originating from a vertex referred to as the *apex* of the cone. When constructing a (constrained) Θ_m-graph, for each vertex u consider the rays originating from u with the angle between consecutive rays being $2\pi/m$. Each pair of consecutive rays defines a cone. The cones are oriented such that the bisector of some cone coincides with the vertical ray emanating from u that lies above u. Let this cone be C_0 of u and number the cones in clockwise order around u (see Fig. 1). The cones around the other vertices have the same orientation as the ones around u. We write C_i^u to indicate the i-th cone of a vertex u, or C_i if u is clear from the context. For ease of exposition, we only consider point sets in general position: no two points lie on a line parallel to one of the rays that define the cones, no two points lie on a line perpendicular to the bisector of a cone, and no three points are collinear. The main implication of this assumption is that no point lies on a cone boundary.

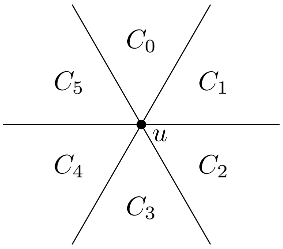


Figure 1 The cones with apex u in the Θ_6-graph. All points of S have exactly six cones.

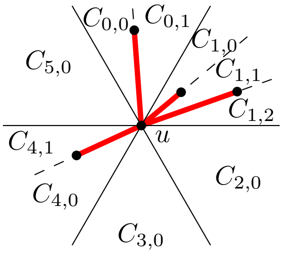


Figure 2 The subcones with apex u in the constrained Θ_6-graph (constraints denoted as red thick segments).

Let vertex u be an endpoint of a constraint c (if any) and let v be the other endpoint and let cone C_i^u be the cone that contains it. The lines through all such constraints c split C_i^u into several *subcones* (see Fig. 2). We use $C_{i,j}^u$ to denote the j-th subcone of C_i^u (again, numbered in clockwise order). When a constraint $c = (u, v)$ splits a cone of u into two subcones, we define v to lie in both of these subcones. We consider a cone that is not split to be a single subcone.

We now introduce the *constrained* Θ_m-graph: for each subcone $C_{i,j}$ of each vertex u, add an edge from u to the closest vertex in that subcone that can see u, where distance is measured along the bisector of the original cone (*not the subcone*). More formally, we add an edge between two vertices u and v if v can see u, $v \in C_{i,j}^u$, and for all points $w \in C_{i,j}^u$ that can see u, $|uv'| \leq |uw'|$, where v' and w' denote the projection of v and w on the bisector of C_i^u and $|xy|$ denotes the length of the line segment between two points x and y. Note that our general position assumption implies that each vertex adds at most one edge per subcone.

We now define our routing model. Formally, a routing algorithm A is a deterministic 1-local, $O(1)$-memory routing algorithm, if the vertex to which a message is forwarded from the current vertex s is a function of s, t, $N(s)$, and M, where t is the destination vertex, $N(s)$ is the set of vertices adjacent to s and set of constraints incident to s and M is a memory of constant size, stored with the message. We consider a unit of memory to consist of a $\log_2 n$ bit integer or a point in P. Our model assumes that the only information stored at each vertex of the graph is $N(s)$.

▶ **Lemma 1.** [1] *Let u, v, and w be three arbitrary points in the plane such that uw and vw are visibility edges and w is not the endpoint of a constraint intersecting the interior of triangle uvw. Then there exists a convex chain of visibility edges from u to v in triangle uvw, such that the polygon defined by uw, wv and the convex chain is empty and does not contain any constraints.*

If u and v do not see each other, the above lemma proves the existence of a convex path between them. We use this property repeatedly in our routing algorithm.

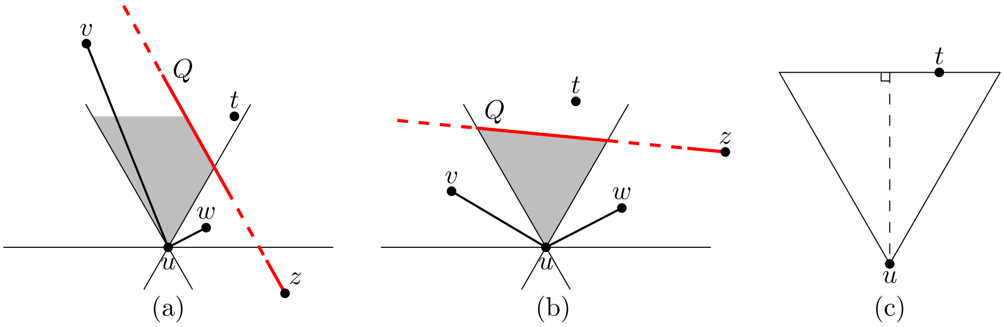


Figure 3 (a) The situation in which Θ-routing follows an edge to v and ends up further away from the destination. (b) The situation where the Θ-routing algorithm cannot follow any edges at u, since the destination t lies behind a constraint. (c) The canonical triangle of u.

3 Routing in the Constrained Θ_6-Graph

Prior to describing our routing strategy for the entire visibility graph, we first provide one for the constrained Θ_6-graph. Note that the Θ_6-graph is not necessarily plane. In this section, we assume that we are given the constrained Θ_6-graph explicitly. In the next section, we show how to use this algorithm to route on the visibility graph.

If there are no constraints, there exists a simple local routing algorithm that works on all Θ-graphs with at least 4 cones. This routing algorithm, which we call Θ-routing, always follows the edge to the closest vertex in the cone that contains the destination. In the constrained setting, this algorithm follows the edge to the closest vertex in the *subcone* that contains the destination. Unfortunately, this approach does not necessarily succeed in the constrained setting due to two issues. First, a key factor of convergence in the unconstrained Θ-routing algorithm is that each step gets us closer to the destination (as long as we have at least 6 cones). Unfortunately, this property need not hold in the constrained setting (see Figure 3a).

A second, more important problem is that the cone containing the destination need not contain any visible vertices. This happens when a constraint is directly blocking visibility (see Figure 3b). In this case, the Θ-routing algorithm will get stuck, since it cannot follow any edge in that cone.

The first problem can be easily fixed: given a vertex u and the destination t, we define the *canonical triangle* of u with respect to t, denoted $\triangle_{ut}$, as the triangle with apex u, bounded by the cone boundaries of the cone of u that contains t and the line through t perpendicular to the bisector of the cone (see Figure 3c). If the edge of u that lies in that cone ends outside the canonical triangle, we say it is *invalid* and we ignore it. By ignoring invalid edges we make sure that any edge we follow leads to a vertex that is closer to t.

To solve the second problem, the routing algorithm needs to find a path even when an obstacle is blocking visibility to the destination (either blocking all visibility from u in the cone of t or because the edge in that cone is invalid). In this case the algorithm enters the *obstacle avoidance phase*, routing differently until an endpoint of the blocking constraint is reached.

Intuitively, our algorithm uses the Θ-routing algorithm until it gets stuck, at which point it switches to the obstacle avoidance phase in order to get around the constraint blocking its visibility to t. After this phase ends, the algorithm switches back to the Θ-routing algorithm. This process is repeated until t is reached. A more precise description follows in Section 3.2.

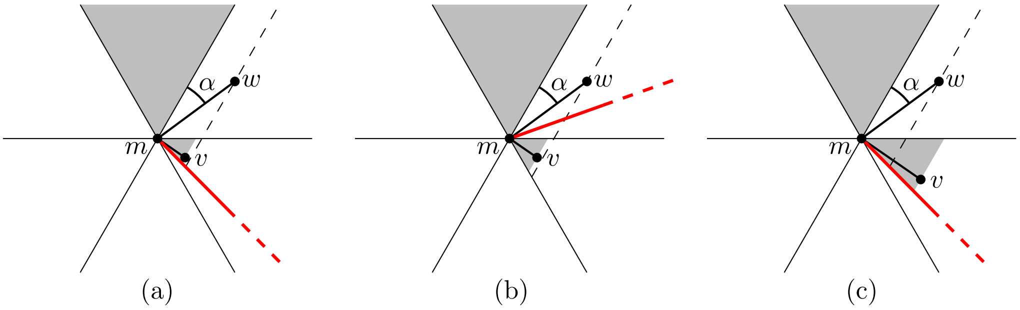


Figure 4 Routing from a vertex m. (a) Follow the edge to v, since v lies in C_4^w. (b) Follow the edge to w, since m is the endpoint of a constraint that intersects mvw. (c) Follow the edge to w, since v lies outside of C_4^w.

3.1 Obstacle Avoidance Phase

We now describe the obstacle avoidance phase. The algorithm enters this phase when routing from source s to destination t, and reaches a vertex u that does not have any valid edges in the cone that contains t. This can only happen if a constraint Q is blocking visibility (if many of them exist, let Q be the one whose intersection with segment $\overline{ut}$ is closest to u). The goal of this phase is to reach the right endpoint of Q, which we denote as z. The main difficulty with this phase is that the algorithm does not know where z is, since Q is not incident on u. In order to overcome this difficulty, the algorithm exploits several geometric properties arising from the unique symmetries present in the constrained Θ_6-graph, some of which are outlined in the proof of Lemma 2.

Without loss of generality, t lies in C_0^u. We first describe the case where u has no edges in C_0. The general case, where u may have invalid edges in C_0, will be considered afterwards. In this first case, the algorithm proceeds as follows. At a current vertex m, the algorithm considers one of two candidate edges to follow (see Figure 4). The first is the edge to the closest visible vertex v in the subcone of C_2^m that shares a boundary with C_1^m. The second edge is the edge from m to the vertex w in C_1^m that minimizes the angle α between $\overline{mw}$ and the right boundary of C_0^m. If v lies in C_4^w and m is not the endpoint of a constraint that intersects the interior of triangle mvw, the algorithm follows the edge to v. Otherwise, it follows the edge to w. In the proof of Lemma 2, we show that at least one of v or w exists. If one of the two vertices v or w does not exist, the algorithm follows the edge that does exist. The obstacle avoidance phase ends when the algorithm reaches the endpoint of a constraint that intersects $\overline{ut}$. In order to recognize this, the algorithm stores u when the phase begins.

▶ **Lemma 2.** *When u has no edges in the cone containing the destination t, the obstacle avoidance phase initiated by u reaches the right endpoint z of the closest constraint Q blocking visibility to t.*

Proof. Without loss of generality, let t lie in C_0^u. Since u has no edges in C_0, the closest constraint Q must intersect both boundaries of C_0^u. This implies that z is either in C_1^u or C_2^u. We maintain the invariant that each intermediate vertex m has no edges in C_0^m and that the intersection of the right boundary of C_0^m and Q is closer to z than in the previous step. We first show that there always exists either a w in C_1^m or a v in C_2^m. This implies that our algorithm eventually reaches z since there are a finite number of points in P.

As a consequence of our invariant, z must either lie in C_1^m or C_2^m. Since m has no edges in C_0, we have that Q is the closest constraint to m in C_0^m. Thus, any point x on $Q \cap C_0^m$ is

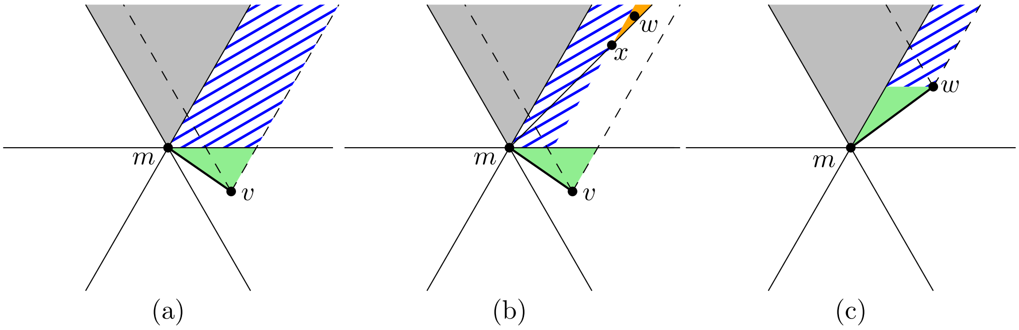


Figure 5 (a) If m routes to v, the union of green and blue regions must be empty of points. (b) An illustration of the proof: if the region is not empty, we find a point x that must have an edge with m that we would have followed instead of v. (c) Routing from m to w.

visible from both m and z. Hence, we can apply Lemma 1 to the triangle mxz and obtain a convex chain of visibility edges from m to z. In particular, this implies that m can see a vertex in $C_1 \cup C_2$, and therefore has an edge in $C_1 \cup C_2$. What remains to be shown is that the invariant is maintained after every step of the algorithm. We note that for any vertex in $C_1^m \cup C_2^m$ the intersection of the right boundary of its cone C_0 is closer to z than that of m. Thus, it remains to show that C_0 of this next vertex contains no edges. We consider the following two cases.

The algorithm follows the edge to v. If the algorithm follows the edge to v, recall that v lies in C_4^w and m is not the endpoint of a constraint that intersects the interior of triangle mvw. In particular, this means that w lies outside of C_0^v. Since v is the closest visible vertex in the subcone of C_2^m that shares a boundary with C_1^m, the part of C_0^v below the horizontal line through m must be empty of points visible to v (see Figure 5a).
By the invariant, $C_0^m \cap C_0^v$ is empty of visible points. What remains to be shown is that there are no points visible to v in $C_0^v \setminus C_0^m$ above the horizontal line through m. If this region is not empty, we sweep the region using the right boundary of C_0^m. Let x be the first vertex hit by this sweep that is visible to m. This implies that the $\triangle_{xm}$ is empty of points visible to x since it is contained in the union of C_0^m (which is empty), the swept part of C_1^m, and a portion of C_2^m that must also be empty by our choice of v. This implies that there is an edge from x to m. This means that w must exist. By construction, $\overline{mw}$ forms the smallest angle with the right boundary of C_0^m. This means that $x \in \triangle_{mw}$. Furthermore, since mw and mv are visibility edges, Lemma 1 implies the existence of a vertex visible to w in $\triangle_{wm}$. This contradicts the existence of the edge mw. Thus, C_0^v is empty of vertices visible to m. Suppose that there was a vertex y visible to v in C_0^v, then since vy and vm are visibility edges, Lemma 1 implies the existence of a vertex visible to m in C_0^v, which is a contradiction.

The algorithm follows the edge to w. As in the previous case, we consider the part below the horizontal line through w and the part above (solid green and dashed blue regions in Figure 5c, respectively). The former region must be empty or the edge mw would not be present: any point visible to m in this region prevents m from creating an edge to w and vice versa. An argument similar to the one for v, showing that the region above the horizontal boundary of C_1 is empty, also proves that the region above the horizontal line through w is empty. Thus, C_0^w must be empty of points visible to w. ◀

We now consider the general case, where u may have invalid edges in C_0 (see Figure 6a).

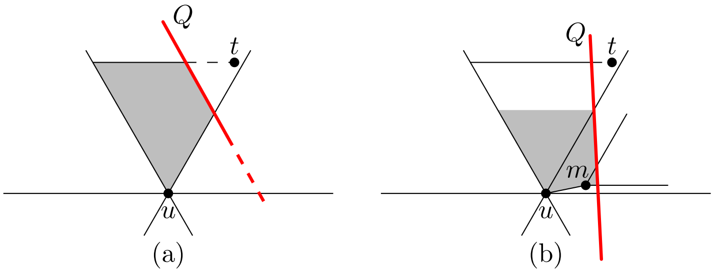


Figure 6 (a) When Q does not fully block the visibility of C_0, we maintain the invariant that the visible portion of the canonical triangle (gray region) must be empty along our routing. (b) The situation where we restart the obstacle avoidance algorithm at m.

In this case, when u initiates the obstacle avoidance phase, we either reach z or a vertex m that has no edges in C_1 and C_2 (see Figure 6b). This latter case can only occur when z lies in C_3^m. Note that this implies that Q intersects both boundaries of C_1^m. Therefore, we initiate a new obstacle avoidance phase from m where C_1 plays the role of C_0. By Lemma 2, the second invocation of the obstacle avoidance phase must reach z.

▶ **Lemma 3.** *When u has no valid edges in the cone containing the destination t, the general obstacle avoidance phase initiated by u reaches the right endpoint z of the closest constraint Q blocking visibility to t.*

We note that the above proof relies heavily on the fact that we have exactly 6 cones (and thus we are in the constrained Θ_6-graph). We have a specific example in which the routing strategy described above would fail for 14 cones (for some node, no edge will keep an invariant zone empty). Thus, a different obstacle avoidance method is needed when the number of cones is not 6.

3.2 Global Routing Strategy

We now have all the pieces in place to describe our routing strategy. Our routing strategy alternates between three phases: while not blocked by an obstacle, we use the classic Θ-routing algorithm. If the current vertex has no valid edges in the cone containing the destination, it must be blocked by a constraint Q. In this case, we enter the obstacle avoidance phase to reach the right endpoint of Q. Once we reach this endpoint, we check which of the two endpoints of Q is closer to the destination. If the closest point to destination is the other endpoint of Q, we enter the *alternative endpoint phase* to reach it. Note that the two endpoints of Q can see each other, so we can route between them using the strategy introduced in [2]. Once we have reached the endpoint of Q that is closest to the destination, we resume classic Θ-routing. We call this alternation between the three phases the *constrained Θ_6-routing strategy.*

3.3 Convergence

We now show that our routing algorithm always reaches the destination. First we give a proof of convergence which greatly overestimates the number of steps needed to reach the destination, but it turns out that first showing that the algorithm always reaches the destination simplifies the proof of bounding the number of steps.

▶ **Lemma 4.** *The constrained Θ_6-routing strategy always reaches the destination within a finite number of steps.*

Proof. By construction, each edge followed during the Θ-routing phase gets closer to the destination. Hence, each Θ-routing phase can consist of at most n steps. Similarly, an obstacle avoidance phase performs at most n steps, since each step brings the boundary of cone C_0 closer to the endpoint we are routing to. At the end of an obstacle avoidance phase, we may need an alternative endpoint phase which visits each vertex at most once [2].

Thus, in order to show termination it remains to bound the number of alternations between phases. Each invocation of an obstacle avoidance phase is tied to a single constraint Q. We bound the number of times Q can trigger an obstacle avoidance phase. Let z be the endpoint of Q that is closest to t. In order for Q to trigger another obstacle avoidance phase the routing path needs to first reach a vertex v such that v and t are in different halfplanes (with respect to the line containing Q). Since the routing path cannot cross the constraint Q itself, in the routing path between z and v we reach a vertex that is further from t than z is.

Since Θ-routing only gets closer to t, we must perform at least one obstacle avoidance phase with a different constraint Q'. Since an obstacle avoidance phase (together with the possible alternative endpoint phase) always ends at the endpoint z' of Q' that is closest to t, this implies that z' is further away from t than z. Let $Q_1, \ldots, Q_k$ be all the constraints sorted by decreasing distance of their closest endpoint to t. Let z_i be the endpoint of Q_i closest to t. Notice that Q_1 cannot invoke more than one obstacle avoidance phase since there are no constraints whose closest endpoint z_i is further from t than z_1. In general, this ordering implies that Q_i cannot invoke an obstacle avoidance phase more than 2^{i-1} times. Therefore, when there are k constraints, there can be at most $2^k - 1$ invocations of an obstacle avoidance phase. Since we take at most $3n$ steps between two obstacle avoidance steps, the total number of steps is upper bounded by $O(n \cdot 2^k)$. ◀

Note that the above reasoning shows that a single constraint could be visited many times, however, a simple argument shows that each constraint invokes at most one obstacle avoidance phase.

▶ **Lemma 5.** *Let Q be a constraint and let z be the endpoint of Q that is closest to t. Vertex z can be visited as the final vertex of at most one obstacle avoidance or alternative endpoint phase.*

Proof. When we reach z at the end of an obstacle avoidance or alternative endpoint phase, since the constrained Θ_6-routing strategy is memoryless at the end of this phase, it follows the same edge from z every time we reach it. This implies that z cannot be visited twice using an obstacle avoidance or alternative endpoint phase, since otherwise the path would cycle indefinitely, contradicting Lemma 4. ◀

This immediately gives a linear bound on the number of phase changes, implying a quadratic bound on the number of steps. We now use a more detailed analysis of the circumstances in which a vertex may be visited to tighten this further to $O(n)$.

▶ **Lemma 6.** *The constrained Θ_6-routing strategy always reaches the destination in $O(n)$ steps.*

Proof. Consider any vertex v and consider how we reached it. We will show that overall, no vertex is visited too many times.

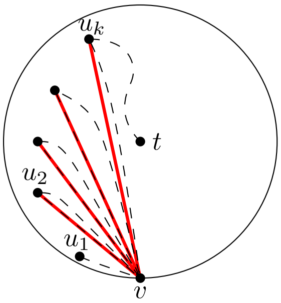


Figure 7 A vertex v can be visited $\Omega(n)$ times as the endpoint not closest to t. This implies that v is the endpoint of many constraints and in all of them it is further away from t than the other endpoint u_2, ..., u_k. For clarity, the disk centred at t passing through v is drawn (as solid black), and a possible routing path that visits v multiple times is also shown (in dashed black).

1) **v is reached during a Θ-routing phase.** Since the routing strategy in this phase is memoryless, we would make the same routing step from v every time we reach it. In particular, this would imply that v cannot be visited twice using a Θ-routing phase (otherwise, the path would cycle indefinitely, contradicting with Lemma 4). Hence, we conclude that v is visited once during a Θ-routing phase during the whole routing algorithm.

2) **v is reached during an avoidance phase of constraint Q.** We consider two subcases:

2.1) **v is not an endpoint of Q.** Let u be the vertex that initiated the avoidance phase and first consider the case in which Q completely blocks visibility of u in the cone containing t (see Figure 3b). In this situation, the same cone remains empty for all vertices along the path (including v). Note that there can be at most three constraints that fully block visibility of v in some cone. Thus, if v is visited more than three times as part of an obstacle avoidance path, two of them share the same cone. Both of these times, the obstacle avoidance and alternative endpoint phases would end up at z, the endpoint of Q closest to t, contradicting Lemma 5. Thus, we conclude that v can be reached this way at most three times.

It is possible that Q did not block the visibility in the cone completely (i.e., we initiated the obstacle avoidance phase because the edge was invalid, see Figure 3a). This situation is very similar to the case in which visibility was completely blocked. The only difference is that the choice of the edge we follow at v depends on the cone that contained t when we started this obstacle avoidance phase as well as on whether or not v has edges in the two adjacent cones. We again conclude that if v is visited more than a constant number of times in this way, the algorithm would route to the same neighbour of v, eventually ending at the same endpoint of Q and contradicting Lemma 5.

2.2) **v is an endpoint of Q.** As argued in Lemma 5, v can only be visited once during the whole execution of the algorithm if it is the endpoint that is closest to t. Similarly, if v is the endpoint that is furthest away from t, we know the algorithm enters the alternative endpoint phase and routes to the opposite endpoint of Q. Note that v could be visited several times this way (see Figure 7). However, notice that v can never be visited twice because of the same constraint Q, as this would imply that we visit the same closest endpoint twice as well, contradicting Lemma 5. Thus, during the entire execution of the algorithm, we can visit at most $3n - 6$ vertices as the endpoint of a constraint that is not closest to t.

3) **v is reached during an alternate endpoint phase.** Every time a vertex is part of a path in the alternate endpoint phase, Lemma 3 of [2] shows that at least one of its cones is empty.

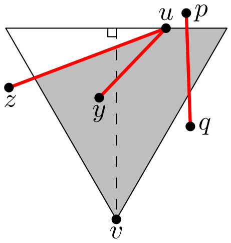


Figure 8 The constrained canonical triangle of v (gray). Constraint uz is used to clip the triangle. Constraint uy does not clip the triangle, since it does not cross the triangle boundary. Constraint pq does not clip the triangle, since it has no endpoint at u.

Hence, excluding case 2.2, each vertex is visited a constant number times. Since case 2.2 adds at most $3n - 6$ visited vertices during the entire execution of the algorithm, this implies that a total of $O(n)$ steps are executed as claimed. ◀

▶ **Theorem 7.** *There exists a 1-local $O(1)$-memory routing algorithm for the constrained Θ_6-graph that reaches the destination in $O(n)$ steps.*

Proof. The algorithm is 1-local by construction, since we consider only information about vertices the current vertex is connected to. The Θ-routing phase does not require any memory. The obstacle avoidance phase and alternative endpoint phase store a single vertex each and this information is discarded when the phase ends. Hence, the algorithm requires $O(1)$ memory. Lemma 6 shows that the algorithm terminates in $O(n)$ steps. ◀

4 Routing on the Visibility Graph

We now return our attention to our main goal: routing on the visibility graph. Since in the previous section we presented a routing algorithm for the constrained Θ_6-graph, we first show that we can use this algorithm to route on the visibility graph as well. Afterwards, we also describe how to modify the constrained Θ_6-routing algorithm to route on the visibility graph directly without locally determining the edges of the constrained Θ_6-graph.

We note that, unfortunately, the length of the paths resulting from these two approaches need not be related to the length of the shortest path in the visibility graph. Since we cannot determine locally which endpoint of a constraint is closest to t, the routing algorithms may follow a path to an endpoint arbitrarily far away, preventing us from being competitive.

4.1 Using the Constrained Θ_6-Graph

In order to use the constrained Θ_6-routing algorithm from the previous section, we need to determine locally at a vertex which of its visibility edges are part of the constrained Θ_6-graph. Since it is easy to locally determine at a vertex u if a vertex v is the closest vertex in one of its subcones, we focus on the situation where this is not the case and we thus have to determine at u if it is the closest vertex in one of the subcones of v. Let the *constrained canonical triangle* of v be $\triangle_{vu}$ clipped using the constraints intersecting the boundary of the canonical triangle with one endpoint at u (see Figure 8). Note that we can determine the constrained canonical triangle of v locally at u.

▶ **Lemma 8.** *Let u and v be two vertices such that v is not the closest vertex to u in any subcone of u. Edge uv is part of the constrained Θ_6-graph if and only if u does not have any visible vertices in the constrained canonical triangle of v.*

Proof. We first note that we can consider the subcone of v that contains u to be the full cone: If the constraint defining the subcone ends in the constrained canonical triangle, Lemma 1 implies that it also contains a vertex visible to u, correctly implying that uv is not an edge. If the constraint does not end in the constrained canonical triangle, the part of the constrained canonical triangle outside the subcone is not visible to u and hence it does not influence the decision at u.

It is easy to see that if u has any visible vertices in the constrained canonical triangle of v, uv is not an edge of the constrained Θ_6-graph: Consider the vertex x such that the smaller angle of ux and uv is minimized. Since the angle is minimized, u is not the endpoint of any constraints intersecting triangle uvx, so we can apply Lemma 1 to uvx. This gives us a vertex inside the constrained canonical triangle that is visible to v. Hence, u is not the closest visible vertex to v and thus uv is not an edge of the constrained Θ_6-graph.

Next we show that if u has no visible vertices in the constrained canonical triangle of v, uv is an edge of the constrained Θ_6-graph. We prove this by contradiction, so assume that uv is not an edge of the constrained Θ_6-graph. This implies that there exists a vertex x in the subcone of v that contains u that is closer to v than u is. Hence, x lies in the constrained canonical triangle. Applying Lemma 1 to uvx gives us a vertex inside the constrained canonical triangle that is visible to u, contradicting that u has no visible vertices in this region. ◀

4.2 Routing Directly on the Visibility Graph

In order to route directly on the visibility graph, instead of at each vertex computing the local neighbourhood in the constrained Θ_6-graph, the constrained Θ_6-routing algorithm needs to be modified. We do this in such a way that the vertices do not need to store any fixed cone orientations.

When a vertex s wants to send a message, it picks an arbitrary cone orientation and stores it in the message it sends. We note that a vertex can pick a different orientation of the cones for each message that it sends and this only requires a constant amount of storage. Since the orientation is stored in the message, vertices do not need to agree on a fixed orientation in advance, as every vertex along the routing path can extract the orientation from the message and use that for its decisions.

Like in the constrained Θ_6-routing algorithm, routing directly on the visibility graph works in three phases: Θ-routing, obstacle avoidance, and alternative endpoint. During the Θ-routing phase a vertex u simply sends the message to the closest vertex in the cone that contains t, again limiting the edges it is allowed to follow to the edges that end in $\triangle_{ut}$.

During the obstacle avoidance phase, we start by routing to either endpoint of the constraint blocking visibility to t. Since we are routing on the visibility graph, Lemma 1 tells us that there is a convex chain of visibility edges to these endpoints. Hence, in order to reach an endpoint of the constraint, we follow one of these convex chains. In order to determine the next edge on the chain at an intermediate vertex m, the message needs to store the predecessor of m on the chain and whether the path should continue to the next clockwise or counter-clockwise edge of m. The next edge along the convex chain at m is the edge that minimizes the angle with the line through m and the predecessor of m in the stored direction.

When we arrive at an endpoint of a constraint, we can determine the location of the other endpoint, since they are connected in the visibility graph. Using this information, we can determine if this constraint is the one that caused the obstacle avoidance phase by checking if it blocks visibility of u to t. If this is the case, we also determine which of the two endpoints is closer to t. If we are not yet at the endpoint closest to t, we start the alternative endpoint phase, which is now simplified to following the edge in the visibility graph to the other endpoint of the constraint.

▶ **Theorem 9.** *There exists a 1-local $O(1)$-memory routing algorithm for the visibility graph that reaches the destination in $O(n)$ steps.*

Proof. We first note that locality follows from the fact that we only need to consider the neighbours of the current vertex in each of the steps. The memory bound follows from the fact that we need to store only the orientation of the cones in the message, as well as the starting vertex of the obstacle avoidance phase.

It remains to bound the number of steps. This algorithm has properties similar to those of the constrained Θ_6-routing algorithm. First, the Θ-routing phase always gets closer to the destination and thus cannot repeat vertices. This implies that Lemma 4 also holds for this routing algorithm. This in turn implies that a vertex can be the closest endpoint of an obstacle avoidance or alternative endpoint phase at most once. Next, since the obstacle avoidance path is convex, this implies that this path visits a subset of the vertices visited by the obstacle avoidance phase of the constrained Θ_6-routing algorithm. Finally, the alternative endpoint phase consists of at most a single edge, hence this phase too is a subpath of its constrained Θ_6-routing counterpart. Hence, when we compare the path of this routing algorithm to the constrained Θ_6-routing path that uses the same cone orientation, the routing path on the visibility graph is a subpath of the constrained Θ_6-routing path. Hence, it takes at most $O(n)$ steps. ◀

5 Conclusion

We presented the first 1-local $O(1)$-memory routing algorithms for the visibility graph that do not require the computation of a planar subgraph. Unfortunately, our algorithms do not give guarantees on the length of the routing path, only on the number of edges used. Hence, designing an algorithm that is competitive with respect to the shortest path remains open.

Acknowledgements. We thank Luis Barba, Sangsub Kim, and Maria Saumell for fruitful discussions.

References

1 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. On plane constrained bounded-degree spanners. In *LATIN*, volume 7256 of *LNCS*, pages 85–96, 2012.

2 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Competitive local routing with constraints. *Journal of Computational Geometry*, 8(1):125–152, 2017.

3 Prosenjit Bose, Matias Korman, André van Renssen, and Sander Verdonschot. Constrained routing between non-visible vertices. In *COCOON*, 2017.

4 Prosenjit Bose and André van Renssen. Upper bounds on the spanning ratio of constrained theta-graphs. In *LATIN*, volume 8392 of *LNCS*, pages 108–119, 2014.

5 Ken Clarkson. Approximation algorithms for shortest path motion planning. In *STOC*, pages 56–65, 1987.

6 Gautam Das. The visibility graph contains a bounded-degree spanner. In *CCCG*, pages 70–75, 1997.

7 Sudip Misra, Subhas Chandra Misra, and Isaac Woungang. *Guide to Wireless Sensor Networks*. Springer, 2009.

8 Harald Räcke. Survey on oblivious routing strategies. In *Mathematical Theory and Computational Practice*, volume 5635 of *LNCS*, pages 419–429, 2009.

An FPTAS of Minimizing Total Weighted Completion Time on Single Machine with Position Constraint[*][†]

Gruia Călinescu[1], Florian Jaehn[2], Minming Li[3], and Kai Wang[4]

1 Dept. of Computer Science, Illinois Institute of Technology, Chicago, USA
calinescu@iit.edu

2 Management Science and Operations Research, Helmut Schmidt University – University of the Federal Armed Forces, Hamburg, Germany
florian.jaehn@hsu-hh.de

3 Dept. of Computer Science, City University of Hong Kong, Hong Kong SAR, China
minming.li@cityu.edu.hk

4 Dept. of Computer Science, City University of Hong Kong, Hong Kong SAR, China
kai.wang@my.cityu.edu.hk

Abstract

In this paper we study the classical scheduling problem of minimizing the total weighted completion time on a single machine with the constraint that one specific job must be scheduled at a specified position. We give dynamic programs with pseudo-polynomial running time, and a fully polynomial-time approximation scheme (FPTAS).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases FPTAS, Scheduling, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.19

1 Introduction

In general, a major challenge of scheduling problems is the determination of a job sequence for each machine involved. Especially in non-preemptive one machine settings without idle times, this is usually the only task to be performed. In this context, scheduling problems appear without restrictions on this sequence (e.g. $1||\sum T_j$, minimize total tardiness) or with restrictions on the sequence (e.g. $1|r_j, prec|\sum C_j$, minimize total completion time). Restrictions on the sequence are commonly either time dependent or linked to job pairs. Examples for time dependent restrictions are release dates, deadlines, or time dependent maintenance activities. Precedence constraints are a typical restriction based on job pairs. In this paper, we elaborate on a different restriction based on the position of a job in the sequence. To be more precise, we force one job to have a fixed position within the sequence of jobs.

The practical and theoretical motivation for such a scheduling problem is twofold. Firstly, such a job that has a fixed position in the sequence could be considered as a maintenance

[*] The work described in this paper was supported by a grant from Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11268616).

[†] A full version of the paper is available at https://arxiv.org/abs/1710.10904.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 19; pp. 19:1–19:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

operation. Classically, maintenance is also considered to be time dependent, e.g. by modeling predetermined machine unavailability intervals ([15, 1, 16]), by allowing a maximum time between two maintenances, which is often referred to as "tool changes" ([6, 7]), or maintenances may be inserted arbitrarily in order to reduce the processing times of the following jobs ([14, 18]). However, just lately, position dependent maintenance operations have been introduced by Drozdowski, Jaehn and Paszkowski [8]. Amongst others, they motivate position dependent maintenance activities with wear and tear of jet engines or aircraft wheels, which is rather caused by the number of flights (because of the climb flight and thrust reversal for the engines) than by the length of the flight. So the problem considered here can be seen as the special case in which exactly one position dependent maintenance activity is necessary.

Secondly, our problem is a special case of scheduling with non-negative inventory constraints, as was introduced by Briskorn et al. [2]. Here, each job either delivers or removes items of a homogeneous good to or from an inventory. Jobs that remove items can only be processed if the required number of items are available, i.e. only if the inventory level does not become negative. This problem relates to ours, in which a job is fixed to position k, as follows. The job fixed on position k can be considered as the only job removing items from the inventory, and $k-1$ jobs are required to deliver items before this job can be scheduled. If the parameter settings of the fixed job are chosen such that this job is to be scheduled as early as possible, it is forced to be on position k. Analogously, the fixed job can be modeled as the only one delivering to the inventory so that it must be scheduled the latest on position k. Parameter settings then need to ensure that it is not scheduled earlier.

In this paper we continue the work of [2] on problem $1|inv|\sum w_jC_j$. We consider one machine with the above mentioned inventory constraint with the objective of minimizing total weighted completion time. Briskorn et al. [2] show that this problem is strongly NP-hard in the general case and they propose various special cases, which are easily solvable and some which are still open. They especially differ between the sets of jobs that deliver to the inventory and that remove goods from the inventory. As mentioned before, we consider a special case in which one of the two sets only consists of one job. For this problem setting, we propose a fully polynomial time approximation scheme (FPTAS).

Several special cases and generalizations of problem $1|inv|\sum w_jC_j$ have been analyzed in the literature. Briskorn and Pesch [5] consider the generalization with a maximum inventory level. They show that even finding a feasible solution is NP-hard and they propose heuristics. Another generalization is analyzed by Kononov and Lin [13]. Here, each job consumes some items at the beginning of its processing time and it adds to the inventory a number of items at its completion time. They show NP-hardness of several special cases and present some approximation algorithms for further special cases. Morsy and Pesch [17] consider a special case in which all jobs delivering to the inventory must be equal (concerning processing time, weight, and inventory modification) and the remaining jobs must also share some characteristics. For this setting, a 2-approximation is presented. Optimality criteria and an exact branch-and-bound algorithm for the standard problem $1|inv|\sum w_jC_j$ are proposed by Briskorn et al. [3].

There are some problems discussed in the literature, which are closely related to $1|inv|\sum w_jC_j$. First of all, Briskorn and Leung [4] consider the problem with maximum lateness objective function. They propose some optimality criteria, lower bounds, and heuristics, which are then used in a branch-and-bound framework. There are various papers ([9, 10, 11, 12]) that analyze a "non-renewable resource constraint", which means that each job removes goods from the inventory, but the inventory is filled automatically at predetermined points in time. So contrary to the above mentioned inventory constraint, which was exclusively based on the job sequence, here the constraint is partially time based. The papers on this problem mostly focus on minimizing makespan. Only Kis [12] considers the same objective function and presents a strong NP-hardness proof and an FPTAS for a special case.

We formulate the problem with the constraint that a fixed amount of jobs must be finished when the special job (refer to as pivot job) starts. We present an FPTAS in this paper. First we propose a dynamic programming whose running time depends on job processing times. To obtain an FPTAS, we use rounding technique: we round the job processing times so that they are polynomial in n and $1/\epsilon$, then we obtain the optimal schedule for the rounded jobs via dynamic programming and apply that schedule to original jobs. However, the rounding approach does not guarantee $(1+\epsilon)$ - approximation. To make it work, we further discover an important property when the rounding technique fails: the job with the largest weight can not be scheduled after (or the same as) the job with the largest processing time. The reason behind is that when this property breaks, the objective value of the optimal schedule is large enough, which makes the dynamic programming solution good enough. With this property, on one hand we apply the rounding technique and on the other hand we put these two special jobs before or after the pivot job accordingly and solve the subproblem.

The remainder of the paper is organized as follows. The problem formulation is given in Section 2. In Section 3 we propose two dynamic programs to solve the problem, with running time polynomial in job processing times and job weights, respectively. Then we use the dynamic programming to design an FPTAS in Section 4. In Section 5, we present another FPTAS as a comparison. In Section 6 we conclude our work.

2 Formulation

The instance of the problem is a set of n jobs $J = \{1, 2, ..., n\}$, a specified job $c \in J$ and an integer $k \in [1, n]$. Each job $j \in J$ is defined by its weight w_j and its processing time s_j (or sometimes refereed to as workload, size). A schedule σ over instance J is an order of jobs, we write $i \preceq_\sigma j$ (resp. $i \succeq_\sigma j$) meaning that job i precedes (resp. succeeds) job j or jobs i, j are the same in schedule σ. The completion time of a job in a feasible schedule is the time when the job finishes. Assume that the machine is never idle unless there is no more job to be processed, we define $C_j^\sigma = \sum_{i \preceq_\sigma j} s_i$ as the *completion time* for each job $j \in J$ in schedule σ. Also, we denote $\delta_j = \frac{w_j}{s_j}$ as the *density* of job j. The objective is to minimize total weighted completion time on a single machine such that there are exactly $k-1$ jobs scheduled before job c, i.e. $\min_\sigma \sum_{j \in J} w_j C_j^\sigma \quad \text{s.t.} \quad k = |\{\ j \mid j \preceq_\sigma c,\ j \in J\ \}|$ where k is part of the input.

▶ **Theorem 1.** *In the optimal solution, jobs that are scheduled before (or after) job c must follow* Smith's order*.*

In classical *Smith's Rule* [19] (or Smith's order), jobs are executed in non-increasing order of its density δ_j. Smith's Rule has been proven to be optimal when there is no position constraint of job c. However, Smith'r Rule does not work in this problem as in the special case where w_c approaches to infinity and the jobs that are placed before the pivot job in the optimal solution should have the smallest processing times. In the sequel, we assume job c is indexed as n, and the remaining jobs $J \setminus \{c\}$ are sorted in Smith's order, i.e. $\delta_1 \geq \delta_2 \geq ... \geq \delta_{n-1}$.

3 Dynamic Programming with Side Constraints

In this section, we propose pseudo-polynomial dynamic programs to solve this problem. Given integer k, job $c \in J$ and sets of jobs J, $H \subset J \setminus \{c\}$, $B \subset J \setminus \{c\}$ such that $H \cap B = \emptyset$, we aim to find the optimal schedule such that

(i) jobs from H are scheduled before job c
(ii) jobs from B are scheduled after job c
(iii) there are exactly $k-1$ jobs scheduled before job c.

We say that job j is *assigned* when the order of job j and job c is determined and *unassigned* otherwise. Therefore, jobs $H \cup B$ are *assigned* and let $U = J \setminus (B \cup H \cup \{c\})$ be the *unassigned* jobs. Let $\hat{s_j} = \lceil \lambda \cdot s_j \rceil$ be the rounded job processing time of job $j \in J$ with a given parameter λ. We would see later that λ is polynomial in n, $1/\epsilon$, and linear in $1/s_{max}$ where s_{max} is the maximum job processing time among all unassigned jobs. In other words, we make sure that for each unassigned job j, $\hat{s_j}$ is polynomial in n and $1/\epsilon$. Similarly, we could also round job weights with a different λ , i.e. $\hat{w_j} = \lceil \lambda \cdot w_j \rceil$, $\forall j \in J$. In the following part of this section, we give two dynamic programs to solve the rounded jobs based on job processing time in Section 3.1 and based on job weight in Section 3.2, and denote $f_s(k, c, J, H, B, \lambda)$ and $f_w(k, c, J, H, B, \lambda)$ as the optimal schedule (the order of jobs) returned by the corresponding dynamic programming for the rounded jobs respectively.

3.1 Based On Job Processing Time

We propose a dynamic programming with pseudo-polynomial running time. That is, we assume for each job that its processing time has already been rounded into integers, and the running time of the dynamic program is polynomial in the number of jobs and the maximum job processing time.

Given a set of jobs $J^* \subseteq J$, we denote $[J^*] = \sum_{j \in J^*} \hat{s_j}$ as the total processing time of jobs J^*. For a feasible schedule γ, let $U(\gamma) = \{i \mid i \in U, i \prec_\gamma c\}$ be the subset of jobs in U which are scheduled before job c in schedule γ.

Let $n = |J|$ and $\hat{S} = [U]$. A *partial schedule* of jobs $J' \subseteq J$ assigns to each of these jobs J' a valid completion time, making sure that each job could be finished within that valid completion time (i.e. jobs do not overlap). First, we try every possibility of the completion time of job c, i.e. we aim to find the optimal schedule σ such that $[U(\sigma)] = L$ where we test every possibility of L from $\{0, 1, \ldots, \hat{S}\}$. Hence, we denote $C_c(L) = L + \hat{s_c} + [H]$ as the completion time of job c when L is fixed. Afterwards, we consider jobs $J' = \{1, ..., j\}$ and focus on two parameters e, E in the optimal schedule σ where $|J' \cap U(\sigma)| = e$ and $[J' \cap U(\sigma)] = E$. Finally, we test every possibility of e, E in the dynamic programming.

▶ **Definition 2.** Let $dp(e, E, j)$ be the total weighted completion time of jobs $J' = \{1, ..., j\}$ in an optimal partial schedule such that there are e jobs from $J' \cap U$ which are scheduled before job c with total processing time E, where $e \in \{0, \ldots, k-1\}$, $j \in \{0, \ldots, n-1\}$, $E \in \{0, 1, \ldots, L\}$. $dp(e, E, j)$ is taken to be infinity if no such partial schedule exists.

To find the optimal schedule of jobs $J^{'}$, we fix the schedule of job j and then solve the subproblem. We show that the completion time of job j could be calculated once job j is determined to be scheduled before or after job c. We put the proof of Lemma 3 in full version.

▶ **Lemma 3.** *For $j = 0$, we have $dp(e, E, j) = 0$ if $e = 0, E = 0$ and $dp(e, E, j) = \infty$ otherwise. For $j > 0$, we have $dp(e, E, j) =$*

$$\min \begin{cases} dp(e, E, j-1) + w_j \left(E + [H \cap J^{'}]\right), & \textit{if } j \in H \\ dp(e-1, E - \hat{s_j}, j-1) + w_j \left(E + [H \cap J^{'}]\right), & \textit{if } j \in U, e > 0, E \geq \hat{s_j} \\ dp(e, E, j-1) + w_j \left(C_c(L) + ([U \cap J^{'}] - E) + [B \cap J^{'}]\right), & \textit{if } j \in B \cup U \end{cases}$$

$$f_s(k, c, J, H, B, \lambda) = \min\nolimits_{L \in [0, \hat{S}]} \; w_c \cdot C_c(L) + dp(k-1-|H|, L, n-1)$$

Time Complexity. Note that the values $[H \cap J^{'}]$, $[B \cap J^{'}]$ and $[U \cap J^{'}]$ could be precomputed, and they will not change the overall running time. In other words, the running time depends on the unassigned jobs. The overall time complexity is $O(n^4 \hat{s}_{max}^2)$ where $\hat{s}_{max} = \max_{j \in U} \hat{s}_j$. Indeed, the dynamic programming has a table size $O(n^2 \hat{S})$, the computation of each value $dp(e, E, j)$ takes $O(1)$ operations, and the dynamic programming needs $O(\hat{S})$ time for $L \in \{0, 1, ..., \hat{S}\}$, thus in total the time complexity is $O(n^2 \hat{S}^2) = O(n^4 \hat{s}_{max}^2)$. It is important that $\hat{s}_{max}$ only depends on the unassigned jobs U.

3.2 Based On Job Weight

In this section, the unassigned jobs U are required to have integer weight, as the running time of the dynamic programming depends on the weights of the unassigned jobs. We assume for each job that its weight is already rounded to integer. As this problem is highly symmetrical, we show that the dynamic programming in Section 3.1 could be applied by Theorem 4. For each job $j \in J$, we create a corresponding job j^* with processing time w_j, weight s_j, and we obtain a new instance J^*, i.e. $\forall j^* \in J^*,\ w_{j^*} = s_j,\ s_{j^*} = w_j$.

▶ **Theorem 4.** *The reverse order of* $f_s(n+1-k, c^*, J^*, B^*, H^*, \lambda)$ *is* $f_w(k, c, J, H, B, \lambda)$.

Proof. We denote $obj(I, k, \sigma)$ as the objective value of schedule σ for the jobs I with parameter k (position constraint parameter). Let π be any feasible schedule for jobs J with parameter k, and let $\pi^{'}$ be the reverse of π, i.e. $i \preceq_\pi j$ if and only if $j \preceq_{\pi^{'}} i$. We prove that

$$obj(J, k, \pi) = obj(J^*, n+1-k, \pi^{'})$$

Firstly, in schedule π there are $k-1$ jobs which are scheduled before job c since π is feasible for J with parameter k. Therefore, in schedule $\pi^{'}$ for J^* there are $n-k$ jobs which are scheduled before job c^* by definition of $\pi^{'}$. Moreover, in schedule π jobs H are scheduled before job c, then in schedule $\pi^{'}$ jobs H^* are scheduled after job c^*. Similar analysis can be used for jobs B. Consequently, schedule $\pi^{'}$ is a feasible schedule for J^* with parameter $n+1-k$. Then, we prove the theorem by the equation: $obj(J, k, \pi) = \sum_{j \in J} w_j \sum_{i \preceq_\pi j} s_i = \sum_{i \in J} s_i \sum_{i \preceq_\pi j} w_j = \sum_{i \in J} w_{i^*} \sum_{i \preceq_\pi j} s_{j^*} = \sum_{i \in J} w_{i^*} \sum_{j \preceq_{\pi^{'}} i} s_{j^*} = obj(J^*, n+1-k, \pi^{'})$. In the first equality, we formulate the objective of schedule π for J. In the second equality, we reorganize the summation. In the third equality, for each $j \in J$ we substitute w_j by s_{j^*} and s_j by w_{j^*} as they have equal value. In the fourth equality, we replace π by $\pi^{'}$.

Consequently, the reverse order of the optimal solution for jobs J^* with parameter $n+1-k$ is optimal for jobs J with parameter k. ◀

4 Fully Polynomial-Time Approximation Scheme (FPTAS)

In this section, we present the FPTAS algorithm. Recall that the dynamic programming in previous section gives the optimal solution while the running time depends on job processing times (or job weights). The straightforward idea is to round the job processing times such that they are polynomial in n and $1/\epsilon$ and then solve the rounded jobs via dynamic programming. However, this technique does not guarantee $(1+\epsilon)$ - approximation where we will show an example. Later, we extract information from this failure and design an FPTAS.

Rounding Technique

For each job $j \in J$, we round job processing time with parameter λ, i.e. $\hat{s}_j = \lceil \lambda \cdot s_j \rceil$ with $\lambda = \frac{h(n, \frac{1}{\epsilon})}{s_{max}}$ where s_{max} is the maximum processing time of all jobs J and $h(n, \frac{1}{\epsilon})$ is a function which is polynomial in n and $1/\epsilon$. We obtain the optimal schedule (denote by σ) for the rounded jobs via dynamic programming and analyze the performance of schedule σ for jobs J. Let π be the optimal schedule for jobs J. The objective of σ could be bounded:

$$\begin{aligned}&\textstyle\sum_{j\in J} w_j \sum_{i\preceq_\sigma j} s_i \leq \sum_{j\in J} w_j \sum_{i\preceq_\sigma j} \hat{s}_i/\lambda \leq \sum_{j\in J} w_j \sum_{i\preceq_\pi j} \hat{s}_i/\lambda \\ &\leq \textstyle\sum_{j\in J} w_j \sum_{i\preceq_\pi j} (\lambda s_i + 1)/\lambda = \mathrm{OPT}(J) + \sum_{j\in J} w_j \sum_{i\preceq_\pi j} 1/\lambda \\ &\leq \mathrm{OPT}(J) + (n/\lambda)\cdot \textstyle\sum_{j\in J} w_j\end{aligned} \tag{1}$$

where in the first and third inequality we use $\lambda s_j \leq \hat{s}_j \leq \lambda s_j + 1$, and in the second inequality we apply the fact that σ is optimal for the rounded jobs. Similarly, when we round job weights with a different parameter λ, i.e. $\hat{w}_j = \lceil \lambda \cdot w_j \rceil, \forall j \in J$, we would have

$$\textstyle\sum_{j\in J} w_j \sum_{i\preceq_\sigma j} s_i \leq \mathrm{OPT}(J) + (n/\lambda)\cdot \sum_{i\in J} s_i \tag{2}$$

The error $(n/\lambda)\cdot\sum_{j\in J} w_j$ in Equation (1) may not be bounded by $\epsilon\cdot\mathrm{OPT}(J)$, where one would see from the following example. In the example, we have 3 jobs where $s_1 = 1, s_2 = 2$, $s_3 \gg 2h(n, \frac{1}{\epsilon})$ and $w_1 = s_3, w_2 = w_1 + 1, w_3 = 1$. After rounding, $\hat{s}_1 = \hat{s}_2 = 1$ as $s_{max} = s_3$, therefore the optimal schedule for the rounded jobs will be $\sigma = (2 \prec 1 \prec 3)$, while the optimal schedule for original jobs is $\pi = (1 \prec 2 \prec 3)$. Therefore, the approximation ratio is $\frac{w_2*2+w_1*(1+2)+1*(1+2+s_3)}{w_1*1+w_2*(1+2)+1*(1+2+s_3)} \geq 17/16$, which is a constant.

Note that the error in Equation (1) is $(n/\lambda)\cdot\sum_{j\in J} w_j = \frac{ns_{max}\cdot\sum_{j\in J} w_j}{h(n,\frac{1}{\epsilon})}$. This error may not be bounded by $\epsilon\cdot\mathrm{OPT}(J)$ if the objective value of optimal solution is small, comparing to the product of maximum job processing time and maximum job weight. Therefore, we focus on two such special jobs, job $u = \arg\max_{j\in J}\{s_j\}$ the job of largest processing time, and job $v = \arg\max_{j\in J}\{w_j\}$ the job of largest weight. Note that $s_{max} = s_u$ and that if job v is scheduled after job u or $u = v$ in the optimal solution, i.e. $v \succeq_\pi u$, we will have $\mathrm{OPT}(J) \geq w_v s_u$, then the error in Equation (1) could be bounded when we take $h(n, \frac{1}{\epsilon}) = n^2 \cdot \frac{1}{\epsilon}$:

$$(n/\lambda)\cdot\sum_{j\in J} w_j \leq n^2 w_v/\lambda = \frac{n^2 w_v \cdot s_u}{h(n,\frac{1}{\epsilon})} \leq \frac{n^2}{h(n,\frac{1}{\epsilon})}\cdot\mathrm{OPT}(J) \leq \epsilon\cdot\mathrm{OPT}(J)$$

Therefore, when the rounding technique fails, we would have $v \prec_\pi u$, i.e. the job of largest weight must be scheduled before the job of largest processing time. This property from the failure of rounding technique plays an important role in designing the FPTAS algorithm.

FPTAS Algorithm

From the above analysis, the rounding technique could possibly fail to return a good solution, which we never know. In case that the failure happens, we would assign some jobs based on the property from such failure that the objective value of the optimal schedule is small (i.e. the job of largest weight must be scheduled before the job of largest processing time). Afterwards we run the rounding technique again, and still a good solution may not be returned. Indeed, we could recursively assign jobs and apply the rounding technique. However, as more and more jobs are assigned, the unassigned jobs will have small weight and processing time, which will not reflect the objective value of the optimal schedule. In other words, the above property will not hold any more. Instead, we would fix the position of one job when such case happens, i.e. the job weight or job processing time of the unassigned job is small enough.

The FPTAS algorithm has many rounds. In each round, we aim to fix the position of one job. More precisely, we make this job as the first job (or last job), then we take the remaining

Algorithm 1 FPTAS Algorithm $F\langle c, k, H, B, U\rangle$

```
Input: Consisting of specified job $c$, specified value $k$, set of jobs $H$ (resp. $B$) assigned to
    be scheduled before (resp. after) $c$, and set of unassigned jobs $U$. Here, $\{c\}, H, B, U$ are
    pairwise disjoint, and $H = \emptyset$ if and only if $B = \emptyset$.
Output: A sequence $\chi$ of jobs $U \cup H \cup B \cup \{c\}$ with exactly $k-1$ jobs scheduled before
    job $c$, or $\emptyset$ if no such sequence exists.
$J_r \leftarrow U \cup H \cup B \cup \{c\}$, $u \leftarrow \arg\max_{j \in U} \{s_j\}$, $v \leftarrow \arg\max_{j \in U} \{w_j\}$
$\chi \leftarrow$ an arbitrary feasible schedule of $J_r$          ▷ best from following cases
call CheckFeasibility()
call FixJob()
call RepeatSize()                                ▷ If $v \succ c$ in the optimal schedule
call RepeatWeight()                              ▷ If $u \prec c$ in the optimal schedule
if $u \neq v$ then                               ▷ If $v \prec c \prec u$ in the optimal schedule
    $\sigma \leftarrow F\langle c, k, H \cup \{v\}, B \cup \{u\}, U \setminus \{u, v\}\rangle$, Update $\chi \leftarrow best\{\chi, \sigma\}$
end if
Return $\chi$
______________________________________________________________________
procedure CheckFeasibility                       ▷ Check Feasibility
    if $|H| > k-1$ or $|H| + |U| < k-1$ then
        Return $\emptyset$
    else if $|H| = k-1$ then                     ▷ termination case
        $B \leftarrow B \cup U$
    else if $|H| + |U| = k-1$ then               ▷ termination case
        $H \leftarrow H \cup U$
    end if
    Sort jobs $H$ and $B$ by Smith's order respectively
    $\chi \leftarrow (H, c, B)$, Return $\chi$.
end procedure
```

jobs as a new instance (update constraint parameter k accordingly) and start over. We guarantee that the performance of the solution lose by a factor of $(1 + \epsilon/n)$ each time when we fix one job. Let J_r be the remaining jobs in current round, i.e. jobs $J \setminus J_r$ are already fixed. We would take J_r as an instance, and let π be the optimal schedule of jobs J_r. In order to find and fix one job from J_r, the algorithm will go into many iterations and assign jobs into sets $H \subset J_r, B \subset J_r$ such that either $H = B = \emptyset$ or $H \neq \emptyset,\ B \neq \emptyset$, where jobs H (resp. B) are determined to be scheduled before (resp. after) job c. Let $U = J_r \setminus (H \cup B \cup \{c\})$ be the unassigned jobs. Let $S = \sum_{j \in J_r} s_j$, $W = \sum_{j \in J_r} w_j$ and $S^{'} = \sum_{j \in U} s_j$, $W^{'} = \sum_{j \in U} w_j$. The algorithm handles the problem separately according to the following inequalities.

$$S^{'} \leq \epsilon S/n \tag{3}$$

$$W^{'} \leq \epsilon W/n \tag{4}$$

▶ **Lemma 5.** *Assume that the optimal schedule assign jobs H (resp. jobs B) before (resp. after) job c, if inequality (3) or (4) holds, we are able to either*

(i) *obtain a feasible schedule with $(1+\epsilon)$-approximation, or*

(ii) *fix one job from J_r as the first job or last job by losing at most a factor of $(1+\epsilon/n)$ comparing to the optimal schedule of J_r.*

Algorithm 2 Algorithm 1 (continued)

```
procedure FixJob                                                              ▷ Fix One Job
    $S \leftarrow \sum_{j \in J_r} s_j$, $W \leftarrow \sum_{j \in J_r} w_j$, $S^{'} \leftarrow \sum_{j \in U} s_j$, $W^{'} \leftarrow \sum_{j \in U} w_j$
    if $S^{'} \leq \epsilon S/n$ and $B = \emptyset$ then                        ▷ termination case
        $H^{'} \leftarrow$ the first $k-1$ jobs from $U$ by non-increasing order of job weight.
        Return $F\langle c, k, H^{'}, U \setminus H^{'}, \emptyset \rangle$
    else if $W^{'} \leq \epsilon W/n$ and $H = \emptyset$ then                   ▷ termination case
        $H^{'} \leftarrow$ the first $k-1$ jobs from $U$ by non-decreasing order of job processing time.
        Return $F\langle c, k, H^{'}, U \setminus H^{'}, \emptyset \rangle$
    else if $S^{'} \leq \epsilon S/n$ and $B \neq \emptyset$ then     ▷ place job $i$ as the last job, i.e. $i \succeq J_r$
        $i \leftarrow \arg\min_{j \in B}\{\delta_j\}$, $\chi' \leftarrow F\langle c, k, \emptyset, \emptyset, U \cup B \cup H \setminus \{i\}\rangle$.
        Return $(\chi', i)$
    else if $W^{'} \leq \epsilon W/n$ and $H \neq \emptyset$ then     ▷ place job $i$ as the first job, i.e. $i \preceq J_r$
        $i \leftarrow \arg\max_{j \in H}\{\delta_j\}$, $\chi' \leftarrow F\langle c, k-1, \emptyset, \emptyset, U \cup B \cup H \setminus \{i\}\rangle$.
        Return $(i, \chi')$
    end if
end procedure
------------------------------------------------------------------------------------------------
procedure RepeatSize                                                ▷ round job processing time
    copy $U, H, B$                                                           ▷ make copy of jobs
    while $|U| + |H| \geq k$ do
        $p \leftarrow \arg\max_{j \in U}\{s_j\}$, $\lambda \leftarrow \frac{n^3}{\epsilon^2 s_p}$,
        $\sigma \leftarrow f_s(k, c, J_r, H, B, \lambda)$, Update $\chi \leftarrow best\{\chi, \sigma\}$.
        $B \leftarrow B \cup \{p\}$, $U \leftarrow U \setminus \{p\}$.
    end while
    $\sigma \leftarrow F\langle c, k, H \cup U, B, \emptyset \rangle$, Update $\chi \leftarrow best\{\chi, \sigma\}$
end procedure
------------------------------------------------------------------------------------------------
procedure RepeatWeight                                                       ▷ round job weight
    copy $U, H, B$                                                           ▷ make copy of jobs
    while $|H| < k - 1$ do
        $q \leftarrow \arg\max_{j \in U}\{w_j\}$, $\lambda \leftarrow \frac{n^3}{\epsilon^2 w_q}$,
        $\sigma \leftarrow f_w(k, c, J_r, H, B, \lambda)$, Update $\chi \leftarrow best\{\chi, \sigma\}$.
        $H \leftarrow H \cup \{q\}$, $U \leftarrow U \setminus \{q\}$.
    end while
    $\sigma \leftarrow F\langle c, k, H, B \cup U, \emptyset \rangle$, Update $\chi \leftarrow best\{\chi, \sigma\}$
end procedure
```

Proof. In the following cases, we claim that i) could be achieved if Case 1) or 2) happens and ii) could be achieved if Case 3) or 4) happens (refer to Algorithm 1 procedure FixJob).

Case 1) If $S^{'} \leq \epsilon S/n$ and $B = H = \emptyset$, we assign jobs $H^{'}$ before job c, jobs $U \setminus H^{'}$ after job c and terminate the algorithm, where H' are the first $k-1$ jobs from U by non-increasing order of job weight. Let χ be the corresponding schedule. In the optimal schedule π, let $L \subset J_r$ (resp. $R \subset J_r$) be the set of jobs scheduled before (resp. after) job c. In schedule χ, we use the corresponding notation $\tilde{L}, \tilde{R}$. Schedule $\psi = (L \cap \tilde{L}, R \cap \tilde{L}, \{c\}, \tilde{R})$ is obtained based on schedule χ by advancing and rearranging jobs $L \cap \tilde{L}$ as the order in the optimal schedule, hence the completion time of jobs $L \cap \tilde{L}$ in schedule ψ is at most that in the optimal schedule, i.e. $C_j^\psi \le C_j^\pi$, $\forall j \in L \cap \tilde{L}$. Since jobs $\tilde{L}$ (resp. $\tilde{R}$) in schedule χ are ordered by Smith Rule, the objective value of χ is at most that of ψ. Note that $S = S' + s_c$ as $B = H = \emptyset$, then we have $C_j^\psi \le S - s_c \le \epsilon S/n$, $\forall j \in R \cap \tilde{L}$ because these jobs are scheduled before job c, and $C_j^\psi \le S$, $\forall j \in \tilde{R}$. Thus the objective value of ψ is at most:

$$\begin{aligned}
&\textstyle\sum_{j\in L\cap\tilde{L}} w_j C_j^\pi + \sum_{j\in R\cap\tilde{L}} w_j(\epsilon S/n) + \sum_{j\in\tilde{R}\cup\{c\}} w_j S\\
&\le \textstyle\sum_{j\in L\cap\tilde{L}} w_j C_j^\pi + (1+\epsilon/n)S\sum_{j\in R\cup\{c\}} w_j\\
&\le \tfrac{1+\epsilon/n}{1-\epsilon/n}\textstyle\sum_{j\in J_r} w_j C_j^\pi
\end{aligned}$$

where in the first inequality we apply $\sum_{j\in\tilde{R}} w_j \le \sum_{j\in R} w_j$ as jobs $\tilde{L}$ are selected by job weight from jobs U, and in the second inequality we apply $C_j^\pi \ge s_c \ge S(1-\epsilon/n)$, $\forall j \in R \cup \{c\}$. As $\frac{1+\epsilon/n}{1-\epsilon/n} \le 1+\epsilon$ for $n \ge 3$ (one would enumerate all possible solutions for $n \le 2$). The claim follows.

Case 2) If $W^{'} \le \epsilon W/n$ and $B = H = \emptyset$, we assign jobs $H^{'}$ before job c, jobs $U \setminus H^{'}$ after job c and terminate the algorithm, where H' are the first $k-1$ jobs from U by non-decreasing order of job processing time. A similar argument could be constructed as Case 1).

Case 3) If $S^{'} \le \epsilon S/n$ and $B \ne \emptyset$, we place job $i = \arg\min_{j\in B}\{\delta_j\}$ as the last job among J_r and reduce to subproblem by taking the remaining jobs $J_r \setminus \{i\}$ as a new instance. Let χ be the schedule transformed from π by placing job i after jobs $J_r \setminus \{i\}$. Schedule χ is feasible because job i must be scheduled after job c in the optimal schedule as $i \in B$. Hence, after transformation the completion time of any job of $J_r \setminus \{i\}$ in schedule χ is at most that in schedule π. By assumption, in the optimal schedule π, job i must be scheduled after all jobs in $(H \cup B \cup \{c\}) \setminus \{i\}$, we have $C_i^\pi \ge \sum_{j\in J_r\setminus U} s_j = S - S^{'}$ and $C_i^\chi = S$. Therefore,

$$\frac{C_i^\chi}{C_i^\pi} \le \frac{S}{S-S^{'}} \le 1 + \frac{\epsilon}{n-\epsilon}$$

Case 4) If $W^{'} \le \epsilon W/n$ and $H \ne \emptyset$, we place job $i = \arg\max_{j\in H}\{\delta_j\}$ as the first job among J_r. A similar argument could be constructed as Case 3). ◀

▶ **Lemma 6.** *Assume the rounding technique fails to return* $(1+\epsilon)$*-approximation solution every time, then either inequality* (3) *or* (4) *will hold after at most* n *iterations.*

Proof. Initially, we have $H = B = \emptyset, U = J_r \setminus \{c\}$. Suppose $S^{'} > \epsilon S/n$ and $W^{'} > \epsilon W/n$. Let job $u = \arg\max_{j\in U}\ \{s_j\}$ be the job of largest processing time among unassigned jobs, and let $v = \arg\max_{j\in U}\ \{w_j\}$. In each iteration, we first apply the rounding technique (round job processing time) with parameter $\lambda = \frac{n^3}{\epsilon^2 s_u}$. Note that the time complexity of dynamic programming in Section 3 only depends on unassigned jobs.

If $v \succeq_\pi u$, we claim that the rounding technique will return $(1+\epsilon)$-approximation solution due to $W^{'} > \epsilon W/n$, as the error in Equation (1) could be bounded

$$(n/\lambda)\cdot\sum_{j\in J_r} w_j = \frac{\epsilon^2 s_u}{n^2}\cdot W < \frac{\epsilon s_u}{n}\cdot W^{'} \le \epsilon\cdot w_v s_u \le \epsilon\cdot \mathrm{OPT}(J_r)$$

Otherwise, $v \prec_\pi u$, we solve by three cases.

Case 1) If $c \prec_\pi v \prec_\pi u$. We assign job u into set B (as the optimal does) and remove job u from U, i.e. $U^{'} = U \setminus \{u\}$. Then we apply the rounding technique (round job processing time) again with $\lambda^{'} = \frac{n^3}{\epsilon^2 s_{u^{'}}}$ where $u^{'} = \arg\max_{j\in U'}\ \{s_j\}$. Note that job v is still the job of largest weight among unassigned jobs $U^{'}$. We claim that we would have $v \prec_\pi u^{'}$ if the rounding technique fails again. Similarly, if $v \succeq_\pi u^{'}$, we have

$$(n/\lambda^{'}) \cdot \sum_{j \in J_r} w_j = \frac{\epsilon^2 s_{u'}}{n^2} \cdot W < \frac{\epsilon s_{u'}}{n} \cdot W^{'} \leq \epsilon \cdot w_v s_{u'} \leq \epsilon \cdot \mathrm{OPT}(J_r)$$

That is, the rounding technique returns $(1+\epsilon)$-approximation solution and the claim follows. Therefore, we have $c \prec_\pi v \prec_\pi u^{'}$ if the rounding technique fails again, which implies that the algorithm will stay on Case 1). Hence, we continue assigning job $u^{'}$ into set B until at some moment $v = u^{'}$ (refer to Algorithm 1 procedure RepeatSize). Consequently, at least one rounding technique will succeed.

Case 2) If $v \prec_\pi u \prec_\pi c$. We apply the rounding technique of rounding job weights by taking $\lambda = \frac{n^3}{\epsilon^2 w_v}$ (refer to Algorithm 1 procedure RepeatWeight). The error in Equation (2) could be bounded due to $S^{'} > \epsilon S/n$:

$$(n/\lambda) \cdot \sum_{j \in J_r} s_j = \frac{\epsilon^2 w_v}{n^2} \cdot S < \frac{\epsilon w_v}{n} \cdot S^{'} \leq \epsilon \cdot w_v s_u$$

A similar argument could be constructed to show that once Case 2) is triggered, the algorithm will stay on Case 2) and at least one rounding technique will succeed.

Case 3) If $v \prec_\pi c \prec_\pi u$. We assign job v into set H and job u into set B, and continue to the next iteration (refer to Algorithm 1 line 8).

One would see that there will be at most n iterations to assign all jobs, i.e. the procedure RepeatSize and RepeatWeight in Algorithm 1 will be called at most n times. ◀

▶ **Lemma 7.** *Algorithm 1 is $(1 + 3\epsilon)$ - approximation with time complexity $O(n^{13}/\epsilon^4)$.*

Proof. We first prove that the algorithm will give $(1 + 3\epsilon)$ - approximation solution. The solution returned by the algorithm comes from either a successful rounding technique, or the termination case in the procedure FixJob (Line 27 and 30), and before that the algorithm may have already fixed some jobs $J \setminus J_r$ (Line 33 and 36). By Lemma 5, the termination case will return $(1 + \epsilon)$-approximation solution, also a successful rounding technique will return $(1 + \epsilon)$-approximation solution, comparing to $\mathrm{OPT}(J_r)$. When fixing one job, we would lose a factor of $(1 + \epsilon/n)$ by Lemma 5, comparing $\mathrm{OPT}(J_r)$. One may think about transforming the optimal solution of jobs J into our solution by fixing one job each time, then each time we still lose a factor of $(1 + \epsilon/n)$ comparing to $\mathrm{OPT}(J)$. Therefore, the total approximation of fixing jobs will increase exponentially, which is $(1 + \epsilon/n)^n = 1 + \epsilon + o(\epsilon^2)$. Finally, the overall approximation is $(1 + \epsilon + o(\epsilon^2))(1 + \epsilon) < 1 + 3\epsilon$.

We show the time complexity of the algorithm. After rounding, the largest job processing time or job weight of unassigned jobs is at most $O(n^3/\epsilon^2)$ as we take $\lambda = \frac{n^3}{\epsilon^2 s_u}$ when rounding job processing time or $\lambda = \frac{n^3}{\epsilon^2 w_v}$ when rounding job weight. Therefore, each dynamic programming has running time $O(n^{10}/\epsilon^4)$, In each iteration, the dynamic programming procedure RepeatSize and RepeatWeight is called once, each procedure executing dynamic programming at most n times. We need to try $O(n)$ iterations to fix one job (Line 33 and 36) or terminate the algorithm (Line 27 and 30) We need to fix at most n jobs. Finally, the time complexity is $O(n^{13}/\epsilon^4)$. ◀

5 Different Approach

In this section, we show that a different FPTAS could be constructed based on the approach by Woeginger [20]. Woeginger proposed some conditions to identify whether a dynamic programming could be transformed into an FPTAS, using the method of trimming state space.

First, we present a different dynamic programming, and then show that the conditions are satisfied. We assume that jobs have integer weights and integer processing times. Recall that job c is indexed as n, and the remaining jobs $J \setminus \{c\}$ are sorted by Smith's order. We start from the schedule which only contains job c, and then add remaining jobs into the schedule one by one. The dynamic programming algorithm works with vector sets $\mathrm{VS}_1, ..., \mathrm{VS}_{n-1}$ where in phase j $(1 \le j \le n-1)$ job j is considered and VS_j is computed from VS_{j-1}. A state vector $[i, X, Y, Z]$ in VS_j encodes a partial schedule without idle time for jobs $\{1, 2, ..., j\} \cup \{c\}$: i is the number of jobs that are scheduled before job c, X (resp. Y) is the total processing time (resp. total weights) of jobs before (resp. after and include) job c, and Z is the objective value for the partial schedule.

Initialization. Set $\mathrm{VS}_0 = \{[0, 0, w_c, w_c s_c]\}$.

Phase j. For every vector $[i, x, y, z]$ in VS_{j-1}, put the vectors $[i, x, y + w_j, z + w_j(s_c + \sum_{i=1}^{j} s_i)]$ and $[i + 1, x + s_j, y, z + w_j(x + s_j) + ys_j]$ into VS_j.

Output. return the vector $[i, x, y, z] \in \mathrm{VS}_{n-1}$ that minimizes the z value such that $i = k-1$.

Note that in phase j, by Smith's rule job j can only be scheduled at the end or right before job c. If job j is scheduled at the end, we just append job j into the schedule, then the objective value only increases by the weighted completion time of job j, i.e. $w_j(s_c + \sum_{i=1}^{j} s_i)$. Otherwise job j is scheduled right before job c. When we insert job j right before job c, besides the weighted completion time of job j, the completion time of those jobs that are scheduled after job j will increase by s_j, therefore the objective value will increase by $w_j(x + s_j) + ys_j$. Since the coordinates of all vectors in all sets VS_j are integers, the cardinality of every vector set VS_j is bounded from above by $O(nW^2S^2)$, therefore the dynamic programming algorithm has a pseudo-polynomial time complexity of $O(n^2W^2S^2)$ where $W = \sum_{j \in J} w_j$ and $S = \sum_{j \in J} s_j$.

We show that an FPTAS exists from the conditions. For $j = 1, ..., n$ we define the input vector $X_j = [w_j, s_j]$. Let $\mathcal{F} = \{F_1, F_2\}$ where

$$\begin{aligned} F_1(w_j, s_j, i, x, y, z) &= [i, x, y + w_j, z + w_j(s_c + \textstyle\sum_{i=1}^{j} s_i)] \\ F_2(w_j, s_j, i, x, y, z) &= [i + 1, x + s_j, y, z + w_j(x + s_j) + ys_j] \end{aligned}$$

Let degree-vector $D = [0, 1, 1, 1]$ and objective function $G(i, x, y, z) = z$ if $i = k - 1$ and ∞ otherwise. Then according to the approach by Woeginger [20], this dynamic programming is ex-benevolent and an FPTAS could be constructed. Especially, the time complexity of the FPTAS is $O(n^2 \log_\Delta^2 W \log_\Delta^2 S) = O(n^6 \log^2 W \log^2 S/\epsilon^4)$ where $\Delta = 1 + \epsilon/n$.

6 Conclusion and Discussion

We study the classical scheduling problem where one specific job must be scheduled at a specified position. We give pseudo-polynomial time dynamic programs to solve this problem, which are polynomial in job processing time and job weight, respectively. Moreover, we design a fully polynomial-time approximation scheme (FPTAS) of $(1 + 3\epsilon)$-approximation with running time $O(n^{13}/\epsilon^4)$. Our first method of using rounding technique based on pseudo-polynomial dynamic programs and fixing items one at a time in case the rounding fails may have further applications. For the new approach of FPTAS in Section 5, the running time depends on the job weight ($\log W$) and job processing time ($\log S$), while our first FPTAS algorithm does not.

It remains open whether this problem is NP-hard or not. It will also be interesting to study the multiple position constraints, i.e. two or more jobs have fixed positions. In this setting, our dynamic programming algorithm could be easily generalized and the rounding technique will not change too much while it is difficult to design the FPTAS algorithm to avoid exponential number of recursions.

References

1 Jacek Błażewicz, Paolo Dell'Olmo, Maciej Drozdowski, and Przemysław Mączka. Scheduling multiprocessor tasks on parallel processors with limited availability. *European Journal of Operational Research*, 149(2):377–389, 2003.

2 Dirk Briskorn, Byung-Cheon Choi, Kangbok Lee, Joseph Leung, and Michael Pinedo. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619, 2010.

3 Dirk Briskorn, Florian Jaehn, and Erwin Pesch. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of scheduling*, 16(1):105–115, 2013.

4 Dirk Briskorn and J YT Leung. Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, 64(12):1851–1864, 2013.

5 Dirk Briskorn and Erwin Pesch. Variable very large neighbourhood algorithms for truck sequencing at transshipment terminals. *International Journal of Production Research*, 51(23-24):7140–7155, 2013.

6 Jen-Shiang Chen. Optimization models for the tool change scheduling problem. *Omega*, 36(5):888–894, 2008.

7 A Costa, FA Cappadonna, and S Fichera. Minimizing the total completion time on a parallel machine system with tool changes. *Computers & Industrial Engineering*, 91:290–301, 2016.

8 M Drozdowski, F Jaehn, and R Paszkowski. Scheduling position dependent maintenance operations. *Operations Research*, 2016. to appear.

9 Péter Györgyi and Tamás Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17(2):135–144, 2014.

10 Péter Györgyi and Tamás Kis. Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research*, 235(1):319–336, 2015.

11 Péter Györgyi and Tamás Kis. Reductions between scheduling problems with non-renewable resources and knapsack problems. *Theoretical Computer Science*, 565:63–76, 2015.

12 Tamás Kis. Approximability of total weighted completion time with resource consuming jobs. *Operations Research Letters*, 43(6):595–598, 2015.

13 Alexander V Kononov and Bertrand MT Lin. Minimizing the total weighted completion time in the relocation problem. *Journal of Scheduling*, 13(2):123–129, 2010.

14 Mikhail A Kubzin and Vitaly A Strusevich. Planning machine maintenance in two-machine shop scheduling. *Operations Research*, 54(4):789–800, 2006.

15 Chung-Yee Lee. Machine scheduling with an availability constraint. *Journal of global optimization*, 9(3-4):395–416, 1996.

16 Chinyao Low, Min Ji, Chou-Jung Hsu, and Chwen-Tzeng Su. Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, 34(2):334–342, 2010.

17 Ehab Morsy and Erwin Pesch. Approximation algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 18(6):645–653, 2015.

18 Kabir Rustogi and Vitaly A Strusevich. Simple matching vs linear assignment in scheduling models with positional effects: A critical review. *European Journal of Operational Research*, 222(3):393–407, 2012.

19 Wayne E Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

20 Gerhard J Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12(1):57–74, 2000.

An Efficient Fixed-Parameter Algorithm for the 2-Plex Bipartition Problem*

Li-Hsuan Chen[†1], Sun-Yuan Hsieh[2], Ling-Ju Hung[‡3], and Peter Rossmanith[4]

1 **Department of Computer Science and Information Engineering**
National Cheng Kung University, Tainan, Taiwan
clh100p@cs.ccu.edu.tw

2 **Department of Computer Science and Information Engineering**
National Cheng Kung University, Tainan, Taiwan
hsiehsy@mail.ncku.edu.tw

3 **Department of Computer Science and Information Engineering**
National Cheng Kung University, Tainan, Taiwan
hunglc@cs.ccu.edu.tw

4 **Department of Computer Science, RWTH Aachen, Aachen, Germany**
rossmani@cs.rwth-aachen.de

Abstract

Given a graph $G = (V, E)$, an *s-plex* $S \subseteq V$ is a vertex subset such that for $v \in S$ the degree of v in $G[S]$ is at least $|S| - s$. An *s-plex bipartition* $\mathcal{P} = (V_1, V_2)$ is a bipartition of $G = (V, E)$, $V = V_1 \uplus V_2$, satisfying that both V_1 and V_2 are s-plexes. Given an instance $G = (V, E)$ and a parameter k, the s-Plex Bipartition problem asks whether there exists an s-plex bipartition of G such that $\min\{|V_1|, |V_2|\} \leq k$. The s-Plex Bipartition problem is NP-complete. However, it is still open whether this problem is fixed-parameter tractable. In this paper, we give a fixed-parameter algorithm for 2-Plex Bipartition running in time $O^*(2.4143^k)$. A graph $G = (V, E)$ is called *defective (p, d)-colorable* if it admits a vertex coloring with p colors such that each color class in G induces a subgraph of maximum degree at most d. A graph G admits an s-plex bipartition if and only if the complement graph of G, $\bar{G}$, admits a defective $(2, s - 1)$-coloring such that one of the two color classes is of size at most k. By applying our fixed-parameter algorithm as a subroutine, one can find a defective $(2, 1)$-coloring with one of the two colors of minimum cardinality for a given graph in $O^*(1.5539^n)$ time where n is the number of vertices in the input graph.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases 2-plex, 2-plex bipartition, bounded-degree-1 set bipartition, defective $(2, 1)$-coloring

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.20

[*] Parts of this work were supported by the Ministry of Science and Technology of Taiwan under grants MOST 105–2221–E–006–164–MY3 and MOST 103–2221–E–006–135–MY3.

[†] Li-Hsuan Chen is supported by the Ministry of Science and Technology of Taiwan under grant MOST 106–2811–E–006–008

[‡] Ling-Ju Hung is supported by the Ministry of Science and Technology of Taiwan under grant MOST 106–2811–E–006–055.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 20; pp. 20:1–20:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a graph $G = (V, E)$, an *s-plex* $S \subseteq V$ is a vertex subset such that for $v \in S$ the degree of v in $G[S]$ is at least $|S| - s$. The notion of s-plexes is a degree relaxed variant of cliques and was defined to study the cohesiveness of subgroups in social networks [24]. It is NP-hard to find an s-plex of maximum cardinality in general graphs [3, 19]. Variants algorithms (*e.g.,* [2, 3, 4, 6, 7]) were designed for finding an s-plex of maximum cardinality in a given graph. Note that the maximum 1-plex problem is exactly the maximum clique problem. The maximum s-plex is trivial when $s = |V|$. An *s-plex bipartition* $\mathcal{P} = (V_1, V_2)$ is a bipartition of $G = (V, E)$, $V = V_1 \uplus V_2$, satisfying that both V_1 and V_2 are s-plexes. Given an instance $G = (V, E)$ and a parameter k, the s-Plex Bipartition problem asks whether there exists an s-plex bipartition of G such that $\min\{|V_1|, |V_2|\} \leq k$.

s-Plex Bipartition
Instance: A graph $G = (V, E)$
Parameter: An integer $k \geq 0$
Question: Does there exist a bipartition $\mathcal{P} = (V_1, V_2)$ such that both V_1 and V_2 are s-plexes and $\min\{|V_1|, |V_2|\} \leq k$?

Graph coloring is often used to model scheduling problems [18, 20]. Given a set of jobs J, one can construct a conflict graph $G = (V, E)$ where $V = J$ and for two jobs $u, v \in V$ having schedule conflicts, there is an edge $uv \in E$. We say that G admits a *proper p*-coloring if vertices in G can be colored with p colors and no two adjacent vertices in G are in the same color class. If vertices (jobs) are in the same color class, then those jobs can be done simultaneously without any conflict. However, the ordinary coloring may be too restricted to model a real scheduling problem in which jobs could tolerate some threshold of conflicts. This gives a more general coloring problem called *defective (p, d)-coloring* introduced in [1, 8, 16]. A vertex subset $S \subseteq V$ is called a *bounded-degree-d set* if the maximum degree of $G[S]$ is at most d. A graph $G = (V, E)$ is called *(p, d)-colorable* if it admits a vertex coloring with p colors such that each color class in G is a bounded-degree-d set. Here d means *defects* and the threshold of conflicts.

Defective (p, d)-Coloring
Input: A graph $G = (V, E)$
Question: Does there exist a (p, d)-coloring of G?

The notation $\chi_d(G)$ called the *defective chromatic number* of G is to denote the minimum p such that G is (p, d)-colorable and $\chi_0(G)$ is the usual chromatic number of the graph G. We see that a defective $(p, 0)$-coloring is a proper coloring.

The s-Plex Bipartition problem is important because it is related to the Defective (p, d)-Coloring. It is not hard to see that a graph G admits an s-plex bipartition if and only if the complement graph of G, $\bar{G}$, is defective $(2, s-1)$-colorable. The problem to determine whether an input graph is defective $(2, 0)$-colorable is equivalent to the recognition of bipartite graphs and can be done in linear time. Surprisingly, the Defective $(2, 1)$-coloring problem is NP-complete for general graphs [9] and even for planar graphs [10] and for graphs of maximum degree 4 [10]. This generalizes that Defective $(2, d)$-coloring is NP-complete for any $d \geq 1$ in general graphs and planar graphs. Moreover, the Defective (p, d)-Coloring is NP-complete for all $p \geq 3$ and $d \geq 0$ in general graphs [10]. To determine whether a planar graph is defective $(3, 1)$-colorable is also NP-complete [10]. It was proved that for any constant d, there exists an $\epsilon > 0$ such that $\chi_d(G)$ can not be approximated within a factor of n^ϵ unless P=NP [10].

Lovász [21] showed that for any positive integer p, any graph $G = (V, E)$ of maximum degree $\Delta(G)$ admits a defective $(p, \lfloor \Delta(G)/p \rfloor)$-coloring and the coloring can be found in time $O(\Delta(G) \cdot |E|)$. The defective chromatic number of planar graphs has been well-studied in [8, 14, 15, 17, 23, 25]. It was proved that any planar graph admits a defective $(3, 2)$-coloring and can be found in $O(n^2)$ time [8]. Poh [23] and Goddard [15] showed that any planar graph admits a special defective $(3, 2)$-coloring in which each color class is the disjoint union of paths.

A problem is *fixed-parameter tractable* (FPT) if given any instance of size n and a positive integer k, one can give algorithms to solve it in time $f(k) \cdot poly(n)$ where $f(k)$ is a computable function only depending on k. Those algorithms are called *fixed-parameter algorithms*. There are many results about fixed-parameter algorithms introduced in [11, 12].

A fixed-parameter algorithm based on *branch-and-reduce* strategy consists of a collection of *reduction rules* and *branching rules*. Given a problem instance (G, k) with the parameter k, reduction rules are used to obtain a smaller problem instance (G', k') in polynomial time such that $|G'| < |G|$ or $k' < k$. The branching rules are used to recursively solve the smaller instances of the problem with smaller parameter. We analyze each branching rule and use the worst-case time complexity over all branching rules as an upper bound of the running time. Search trees are often used to illustrate the execution of a branching algorithm. The root of a search tree represents the input of the problem, every child of the root represents a smaller instance reached by applying a branching rule associated with the instance of the root. One can recursively assign a child to a node in the search tree when applying a branching rule. Notice that we do not assign a child to a node when applying a reduction rule. The running time of a branching algorithm is usually measured by the maximum number of leaves in its corresponding search tree.

Let b be any branching rule. When rule b is applied, the current instance (G, k) is branched into $r \geq 2$ instances (G_i, k_i) where $|G_i| \leq |G|$ and $k_i = k - t_i$. Notice that fixed-parameter algorithms return "No" when the parameter $k \leq 0$. We call $\mathbf{b} = (t_1, t_2, \ldots, t_r)$ the *branching vector* of branching rule b. This can be formulated in a linear recurrence

$$T(k) \leq T(k - t_1) + T(k - t_2) + \cdots + T(k - t_r)$$

where $T(k)$ is the number of leaves in the search tree depending on the parameter k. The running time of the branching algorithm using only branching rule b is $O(poly(n) \cdot T(k)) = O^*(c^k)$[1], where c is the unique positive real root of $x^k - x^{k-t_1} - x^{k-t_2} - \cdots - x^{k-t_r} = 0$ [13]. The number c is called the *branching number* of the branching vector $(t_1, t_2, \ldots, t_r)$.

To the best of our knowledge, whether s-Plex Bipartition admits a fixed-parameter algorithm is an open problem. We first give a simple fixed-parameter algorithm to solve the s-Plex Bipartition problem runs in time $O^*((s + 1)^k)$ by reducing the problem to Minimum Ones $(s + 1)$-SAT.

The following Bounded-Degree-d Set Bipartition problem is equivalent to the Defective $(2, d)$-Coloring problem. Notice that G admits a s-plex bipartition if and only if $\bar{G}$ has a bounded-degree-$(s - 1)$ set bipartition. Moreover, G has a bounded-degree-d set bipartition if and only if G is defective $(2, d)$-colorable and there exists one color class having at most k vertices.

[1] For functions f and g we write $f(k, n) = O^*(g(k))$ if $f(k, n) = O(g(k) \cdot poly(n))$, where $poly(n)$ is a polynomial.

Bounded-Degree-d Set Bipartition (BD-dSB)
Instance: A graph $G = (V, E)$
Parameter: An integer $k \geq 0$
Question: Does there exist a vertex bipartition (B, W) with $V = B \uplus W$ such that both B and W are bounded-degree-d sets and $|B| \leq k$?

We first show that the BD d-SB problem can be reduced to the following Minimum Ones $(d+2)$-SAT problem and can be solved in time $O^*((d+2)^k)$.

Minimum Ones $(d+2)$-SAT
Instance: A $(d+2)$-CNF formula F.
Parameter: An integer $k \geq 0$
Question: Does there exist a 0/1 satisfying assignment for F such that the number of ones is at most k?

The BD-dSB can be formulated with $(d+2)$-CNF formula F as follows.

- For each vertex v in the input graph G, create a variable x_v.
- For each $(d+2)$ star S_{d+2} in G with center u and $(d+1)$ leaves $v_1, v_2, \ldots, v_{d+1}$ being $(d+1)$ neighbors of u in G, create two clauses $(x_u \vee x_{v_1} \vee \cdots \vee x_{v_d} \vee x_{v_{d+1}})$ and $(\bar{x}_u \vee \bar{x}_{v_1} \vee \ldots \bar{x}_{v_d} \vee \bar{x}_{v_{d+1}})$ in $(d+2)$-CNF formula which means all $u, v_1, \ldots, v_{d+1}$ cannot be colored all black or colored all white.

If the variable $x_u = 1$ means that the vertex u is colored black and the variable $x_u = 0$ means that the vertex u is colored white. It is not hard to see that F has a 0/1 satisfying assignment such that the number of ones is at most k if and only if the input graph G of the BD-dSB problem admits a vertex bipartition (B, W) with $V = B \uplus W$ such that both B and W are bounded-degree-d sets and $|B| \leq k$. Thus, to solve the BD-dSB problem can be reduced to solve the Minimum One $(d+2)$-SAT problem.

The Minimum one $(d+2)$-SAT problem can be solved in time $O^*((d+2)^k)$ by the following algorithm where k is the number of true variables.

- For each clause $(x_1 \vee \cdots \vee x_{d+2})$, the algorithm branches $d+2$ cases, *i.e.*, for $i = 1, \ldots, d+2$, let $x_i = 1$ and $k := k - 1$.

The above algorithm for Minimum Ones $(d+2)$-SAT runs in time $O^*((d+2)^k)$ which shows that the s-plex bipartition problem considered in this paper can be solved in time $O^*((s+1)^k)$, *i.e.*, fixed-parameter tractable for constant s. For $s = 2$, this algorithm runs in time $O^*(3^k)$.

In this paper, we design more customized fixed-parameter algorithm for 2-Plex Bipartition and improve the running time to be $O^*(2.4143^k)$ where $k = \min\{|V_1|, |V_2|\}$, (V_1, V_2) is a bipartition of V satisfying that both V_1 and V_2 are 2-plexes. By applying this algorithm as a subroutine, we give the first exact algorithm to find a Defective $(2,1)$-Coloring with one of the two colors of minimum cardinality for a given graph with running time $O^*(1.5539^n)$ where n is the number of vertices in the input graph. In the following, we define a problem related to Bounded-Degree-1 Set Bipartition.

Color Constrained Bounded-Degree-1 Set Bipartitioning (CCBD-1SB)
Instance: A graph $G = (B \uplus U, E)$
Parameter: An integer $k \geq 0$
Question: Does there exist a vertex bipartition $U = (B', W')$ such that both $B \cup B'$ and W' are bounded-degree-1 sets and $|B \cup B'| \leq k$?

Notice that if $B = \emptyset$, the CCBD-1SB problem is equivalent to the BD-1SB problem. In the rest of the paper, we solve the CCBD-1SB problem.

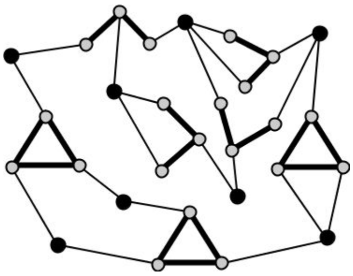


Figure 1 A partially black triple graph where grey nodes denote vertices in U and black nodes denote vertces in B. We use thick lines to denote edges with both endvertices in U and thin lines to denote edges with one endvertex in B and the other endvertex in U.

We close the section with some notation definitions. Let $G = (V, E)$ be a simple graph. For a vertex v in G, we use $N_G(v)$ to be the set of vertices adjacent to v. A path P_h in G is a path consisting of h vertices. We use C_h to denote a cycle of h vertices. If we use $(v_1, v_2, \dots, v_h)$ to denote a P_h, then it means that $v_i v_{i+1} \in E$ for $1 \leq i \leq h-1$. If we use $(v_1, v_2, \dots, v_h)$ to denote a C_h, then it means that $v_i v_{i+1} \in E$ for $1 \leq i \leq h-1$ and $v_1 v_h \in E$. For a vertex set $X \subseteq V$, let $G[X] = (X, E(X))$ where $E(X) = \{uv \in E \mid u, v \in X\}$. For $X, Y \subseteq V$, use $X \uplus Y$ to be $X \cup Y$ satisfying $X \cap Y = \emptyset$.

2 A fixed-parameter algorithm for partially black triple graphs

In this section, we define a graph class called partial black triple graphs $\mathcal{C}$ and give a fixed-parameter algorithm for the Color Constrained Bounded-Degree-1 Set Bipartition problem in partial black triple graphs running in time $O(2^k)$.

▶ **Definition 1.** A graph $G = (V = B \uplus U, E)$ is called a partially black triple graph if the following conditions hold.

(i) Vertices in B are colored black and form an independent dominating set[2] in G.

(ii) Vertices in U are uncolored. Each vertex $v \in U$ is adjacent to exactly one black vertex in B.

(iii) Each connected component in $G[U]$ is either a P_3 or a C_3.

Next we define the Color Constrained Bounded-Degree-1 Set Bipartition problem in partially black triple graphs.

CCBD-1SB in Partially Black Triple Graphs

Instance: A partially black triple graph $G = (B \uplus U, E)$

Parameter: An integer $k \geq 0$

Question: Does there exist a vertex bipartition $U = B' \uplus W'$ such that both $B \cup B'$ and W' are both bounded-degree-1 sets and $|B \cup B'| \leq k$?

Suppose that we use two colors black and white to color vertices U. A black-and-white coloring of U is said to be *feasible* if all black vertices in G form a bounded-degree-1 set and all white vertices in G form a bounded-degree-1 set.

[2] A vertex set $D \subseteq V$ is an *independent dominating set* in a graph $G = (V, E)$ if no two vertices in D are adjacent in G and each vertex $v \in V \setminus D$ is adjacent at least one vertex of D.

▶ **Lemma 2.** *Let $G = (B \uplus U, E)$ be a partially black triple graph that admits a vertex bipartition of $U = B' \uplus W'$ such that both $B \cup B'$ and W' are both bounded-degree-1 sets and $|B \cup B'| \leq k$. Then there exists $S \subseteq U$ of minimum cardinality such that $U \setminus S$ is a bounded-degree-1 set and $|S| \leq k - |B|$. Moreover, S can be found in polynomial time.*

Proof. Since G admits a bounded-degree-1 bipartition $(B \cup B', W')$ such that $|B'| \leq k - |B|$ and S is a subset of U of minimum cardinality such that $U \setminus S$ is a bounded-degree-1 set, we see that $|S| \leq |B'| \leq k - |B|$. According to the definition of partially black triple graph, each connected components of $G[U]$ is either a P_3 or a C_3. The set S can be obtained by picking exactly one vertex of degree one in each P_3 and exactly one vertex in each C_3 in $G[U]$. Thus the set S can found in polynomial time. This completes the proof. ◀

▶ **Lemma 3.** *Let $G = (B \uplus U, E)$ be a partially black triple graph. To determine whether there exists a vertex bipartition of $U = B' \uplus W'$ such that both $B \cup B'$ and W' are bounded-degree-1 sets and $|B \cup B'| \leq k$ can be done in time $O^*(2^{k-|B|})$.*

Proof. Let $S \subseteq U$ of minimum cardinality such that $G[U \setminus S]$ is of maximum degree one. By Lemma 2, if G admits a vertex bipartition of $U = B' \uplus W'$ such that both $B \cup B'$ and W' are both bounded-degree-1 sets and $|B \cup B'| \leq k$, then $|S| \leq k - |B|$. Thus, if $|S| > k - |B|$, our algorithm can simply return that no such bipartition of U exists in polynomial time.

Suppose that $|S| \leq k - |B|$. Our algorithm enumerates $2^{|S|} \leq 2^{k-|B|}$ possibilities to partition $S = S_B \uplus S_W$ where $S_B \subseteq B'$ and $S_W \subseteq W'$. Notice that S is an independent set in G. This implies that S_B and S_W are both independent sets in G. If $S_B \cup B$ is not a bounded-degree-1 set, then the partition of S is not feasible. Assume that $S_B \cup B$ is a bounded-degre-1 set in G. We say vertices in $B \cup S_B$ are colored black and vertices in S_W are colored white.

Notice that vertices in S are collected by picking exactly one vertex of degree one in each P_3 and exactly one vertex in each C_3 in $G[U]$. Let $U' = U \setminus S$. It is easy to see that $G[U']$ is a 1-regular graph.

In the rest of the proof, we call vertices in U' uncolored and we call U' the uncolored set. Once an uncolored vertex is colored black or white, it is removed from the uncolored set U'. The rest of the problem is to color vertices U' black or white.

If the following cases exist, we can simply color a vertex white or black.

- If $v \in U'$ is adjacent to two black vertices, then v must be colored white. Remove v from U'.
- If $v \in U'$ is adjacent to two white vertices, then v must be colored black. Remove v from U'.
- If $v \in U'$ is adjacent to a black vertex x and x has a black neighbor, then v must be colored white. Remove v from U'.
- If $v \in U'$ is adjacent to a white vertex x and x has a white neighbor, then v must be colored black. Remove v from U'.
- If there exists $v \in U'$ of degree two in G, $N_G(v) = \{x, y\}$, satisfying that x is black, y is white, and vertices in $N_G(y) \setminus \{v\}$ are all black, then color v white. Remove v from U'.

Suppose the above cases do not exist. We see that $G[U']$ is a 1-regular graph and every vertex $v \in U'$ is either of degree two or three in G. Moreover, in G each uncolored vertex $v \in U'$ of degree two is adjacent to an uncolored vertex and a black vertex and each uncolored vertex $v \in U'$ of degree three is adjacent to an uncolored vertex, a black vertex, and a white vertex.

▶ **Claim 4.** *If each uncolored vertex $v \in U'$ of degree two is adjacent to an uncolored vertex and a black vertex; and each uncolored vertex $v \in U'$ of degree three is adjacent to an uncolored vertex, a black vertex, and a white vertex, then any feasible coloring of $U' = B' \uplus W'$ must satisfy $|B'| = |W'|$.*

Proof of Claim 4. Let $u, v \in U'$ be two adjacent vertices in G. Note that both u, v are adjacent to exactly one black vertex and at most one white vertex. If both u and v are colored black, then there exists a P_3 (u, v, x) such that all three vertices are colored black where x is the white vertex adjacent to u, a contradiction to the fact that this is a feasible coloring. Since at least one of u and v is adjacent to a white vertex, say u, if both u and v are colored white, then there exists a P_3 (u, v, x) such that all three vertices are colored white where x is the white vertex adjacent to u, a contradiction to the fact that this is a feasible coloring. Thus, one of u and v must be colored black and the other must be colored white. This shows that any feasible coloring of $U' = B' \uplus W'$ must satisfy $|B'| = |W'|$. This completes the proof. ◀

Because any two adjacent vertices $u, v \in U'$ must be colored different colors and any feasible coloring of $U' = B' \uplus W'$ must satisfy $|B'| = |W'|$, we can formulate the rest problem as a 2-SAT problem. Here is the way to formulate the problem with a 2-CNF formula.

- For each $v \in U'$, create a variable x_v.
- For any two adjacent vertices $u, v \in U'$, create two clauses $(x_u \vee x_v)$ and $(\bar{x}_u \vee \bar{x}_v)$ in the 2-CNF formula. The two clauses mean that x_u and x_v cannot be both true or both false. We use $x_u = 1$ to denote that u is colored black and $x_u = 0$ to denote that u is colored white.
- For any black vertex z, if there exists $u, v \in U'$ and (u, v, z) is a P_3 in G, create a clause $(\bar{x}_u \vee \bar{x}_v)$ in the 2-CNF formula. This means x_u and x_v cannot be both true. Both u and v cannot be both colored black.
- For any white vertex z, if there exists $u, v \in U'$ and (u, v, z) is a P_3 in G, create a clause $(x_u \vee x_v)$ in the 2-CNF formula. This means x_u and x_v cannot be both false. Both u and v cannot be both colored white.

It is not hard to see that the 2-SAT formula returns true if and only if U' admits a feasible coloring with $U' = B' \uplus W'$ satisfying $|B'| = |W'|$. Since 2-SAT can be solved in polynomial time [22], to determine whether U admits a feasible 2-coloring can be done in polynomial time.

Notice that the algorithm enumerates all $2^{|S|} \leq 2^{k-|B|}$ possibilities of black and white colorings of vertices in S. This produces at most $2^{k-|B|}$ subproblems. From the above cases analysis, all these coloring subproblems can be solved in polynomial time. This shows that to determine whether there exists a vertex bipartition of $U = B' \uplus W'$ such that both $B \cup B'$ and W' are bounded-degree-1 sets and $|B \cup B'| \leq k$ can be done in time $O^*(2^{k-|B|})$. This completes the proof. ◀

3 A fixed-parameter algorithm for general graphs

In this section, we give a branch-and-reduce algorithm for the CCBD-1SB problem to solve the CCBD-1SB problem running in time $O^*(2.4143^k)$ where k is the number of black vertices. Given an input graph $G = (B \uplus U, E)$, of the CCBD-1SB problem, the algorithm outputs a vertex bipartition $U = (B', W')$ such that both $B \cup B'$ and W' are bounded-degree-1 sets, if the bipartition exists. Note that U is called the set of uncolored vertices, vertices in U are called *uncolored*, and vertices in B are called black. The algorithm consists of reduction rules and branching rules and repeats the execution of the first applicable rule in the sequence. Thus, inside a given case, the hypotheses of all previous rules are assumed to be inapplicable. Notice that if we say an uncolored vertex is colored black or white in the algorithm, then the vertex will be removed from the uncolored set U but the vertex remains in the graph G. We say a vertex is removed from G that means it is deleted from the input graph.

3.1 Reduction Rules

Let $G = (B \uplus U, E)$ be the input graph of the CCBD-1SB problem and k be the input parameter. We first give some reduction rules applied in the fixed-parameter algorithm.

Reduction Rules

- **Too many colored neighbor rule.** If there exists an uncolored vertex having two black and two white neighbors, then return "No."
- **Same color P_3 rule.** If there exists a P_3 with three vertices getting the same color, then return "No."
- **Two adjacent vertex rule.** If black (white) vertices u, v are adjacent, then any vertex x adjacent to u, v must be colored white (black). Remove x from U and remove u, v from G if both u, v are white.
- **Isolated white rule.** If a white vertex v only adjacent to black vertices, then remove v from G.
- **Two same color neighbors rule.** If there is an uncolored vertex $v \in U$ with two black (white) neighbors, then v must be colored white (black). Remove v from U.
- **One uncolored rule.** If there is an uncolored vertex $v \in U$ without uncolored neighbors satisfying one of the following conditions,

  1. v has no white neighbors, or
  2. v has exactly one white neighbor u and at most one black neighbor, and all neighbors of u are colored black.

  Then v must be colored white. Remove v from U.
- **Two uncolored rule.** If there are two adjacent uncolored vertices $u, v \in U$ in G having no white neighbors and no other uncolored neighbors, then u, v must be colored white. Remove u, v from U. Notice that if G has no white vertices and the **Two uncolored rule** is not applicable, there is no connected component which is a P_2 in $G[U]$.
- **No black neighbor triple rule.** If there exists three uncolored vertices $x, y, z \in U$ inducing a P_3 or a C_3 satisfying that x is not adjacent to any black vertex and x, y, z has no uncolored neighbors other than x, y, z and x, y, z are not adjacent to any white vertex in G, then color x black and color y, z white. When the **No black neighbor triple rule** can not be applied, we see that each vertex in a connected component with exactly three vertices in $G[U]$ has a black neighbor in G.

▶ **Lemma 5.** *Two adjacent vertex rule is valid.*

Proof. Suppose that both u and v are black. If a black vertex x is adjacent to u or v, then there exists a P_3 (u, v, x) or (v, u, x) such that all three vertices are colored black, a contradiction to the fact that this is a feasible coloring. Thus, all vertices adjacent to u, v must be colored white. Suppose that both u and v are white. If a white vertex x is adjacent to u or v, then there exists a P_3 (u, v, x) or (v, u, x) such that all three vertices are colored white, a contradiction to the fact that this is a feasible coloring. Thus, all vertices adjacent to u, v must be colored black. This completes the proof. ◀

▶ **Lemma 6.** *Isolated white rule is valid.*

Proof. Let v be a white vertex only adjacent to black vertices and all its neighbors are colored. Since the rest of uncolored vertices to be colored black or white are not affected by the color of v, v can be removed from G safely. This completes the proof. ◀

▶ **Lemma 7.** *Two same color neighbors rule is valid.*

Proof. Suppose that v is an uncolored vertex having two black neighbors. If v is black, then the set of black vertices is not a bounded-degree-1 set because the degree of v is two in the induced subgraph of black vertices. Suppose that v is an uncolored vertex having two white neighbors. If v is white, then the set of white vertices is not a bounded-degree-1 set because the degree of v is two in the induced subgraph of white vertices. This completes the proof. ◀

▶ **Lemma 8.** *One uncolored rule is valid.*

Proof. Let v be an uncolored vertex having no uncolored neighbors, *i.e.,* v is an isolated vertex in $G[U]$. Notice that v is adjacent to at most one black vertex in G. Let $\mathcal{P}$ be an optimal solution of CCBD-1SB with minimum number of black vertices and the number of black vertices is at most k. Suppose that v is not adjacent to any white vertex and v is colored black in $\mathcal{P}$. We then obtain a solution $\mathcal{P}'$ from $\mathcal{P}$ by recoloring v white, a contradiction to the assumption that $\mathcal{P}$ is an optimal solution with minimum number of black vertices. Suppose that v is adjacent to exactly one white neighbor u and u is only adjacent to black vertices. If v is colored black in $\mathcal{P}$, we then obtain a solution $\mathcal{P}'$ from $\mathcal{P}$ by recoloring v white, a contradiction to the assumption that $\mathcal{P}$ is an optimal solution with minimum number of black vertices. This completes the proof. ◀

▶ **Lemma 9.** *Two uncolored rule is valid.*

Proof. Let x, y be two adjacent uncolored vertices having no white neighbors and no other uncolored neighbors, *i.e.,* $\{u, v\}$ induces a P_2 in $G[U]$ and forms a connected component in $G[U]$. Note that u and v are only adjacent to black vertices. Let $\mathcal{P}$ be an optimal solution of CCBD-1SB with minimum number of black vertices and the number of black vertices is at most k. If one of u, v is black, say u, a solution $\mathcal{P}'$ can be obtained from $\mathcal{P}$ by recoloring u white, a contradiction to the assumption that $\mathcal{P}$ is an optimal solution with minimum number of black vertices. This completes the proof. ◀

▶ **Lemma 10.** *No black neighbor triple rule is valid.*

Proof. Let $\{x, y, z\}$ be a connected component in $G[U]$ satisfying that x is not adjacent to any black vertex in G. Notice that $\{x, y, z\}$ induces either a P_3 or a C_3 in $G[U]$ and y and z are adjacent to at most one black neighbor and no white vertex is adjacent to x, y, z in G. Suppose that there is a solution $\mathcal{P}$ of CCBD-1SB that x is white. Since $\{x, y, z\}$ induce a P_3 or a C_3, at least one of y and z must be black in $\mathcal{P}$, otherwise there is a degree-two vertex in the induced subgraph of white vertices. We see that a feasible solution $\mathcal{P}'$ can be obtained from $\mathcal{P}$ by recoloring x black and y and z white. Moreover, $\mathcal{P}'$ and $\mathcal{P}$ have same number of black vertices. This completes the proof. ◀

3.2 Branching Rules

Suppose all the reduction rules are not applicable. The algorithm then applies the following branching rules.

- **One white neighbor rule.** There is an uncolored vertex v having exactly one white neighbor u and satisfying that v or u has an uncolored neighbor x. Then the algorithm branches on each of the following cases:
  - v is black and $k := k - 1$; or
  - v is white and x is black and $k := k - 1$.

  The branching vector of this rule is $(1, 1)$ and the branching number is 2. Notice that if **One white neighbor rule** is not applicable, no white vertices is adjacent to an uncolored vertex. Moreover, if the **One uncolored rule** and **One white neighbor rule** can not be applied, there is no isolated vertices in $G[U]$.
- **Standard branching rule.** There is an uncolored vertex v with at least three uncolored neighbors $v_1, v_2, \ldots, v_h$. Then the algorithm branches on each of the following cases:
  - v is black and $k := k - 1$;
  - v is white and $v_1, v_2, \ldots, v_h$ are black and $k := k - h$; or
  - branch on each $1 \leq i \leq h$, let v, v_i be white and $\{v_1, v_2, \ldots, v_h\} \setminus \{v_i\}$ be black and $k := k - h + 1$.

  The branching vector of this rule is $(1, h, h - 1, \ldots, h - 1)$ where the cardinality of the branching vector is $h + 2$ and $h \geq 3$. The worst cases happens when $h = 3$ and its branching vector is $(1, 3, 2, 2, 2)$ and the branching number is 2.4143. Note that if the **Standard branching rule** is not applicable, the maximum degree of $G[U]$ is at most two.
- **Four path end rule.** There exists a P_4 (v_1, v_2, v_3, v_4) in $G[U]$ satisfying that v_1 has only one uncolored vertex v_2 and v_2 has only two uncolored neighbors v_1, v_3. The algorithm branches on each of the following cases:
  - v_2 is black and $k := k - 1$;
  - v_1, v_2 are white and v_3 is black and $k := k - 1$; or
  - v_2, v_3 are white and v_1, v_4 are black and $k := k - 2$.

  The worst case branching vector of this rule is $(1, 1, 2)$ and the branching number is 2.4143. When all the reduction rules, **One white neighbor rule**, **Standard branching rule**, and **Four path end rule** are not applicable, we see that any path in $G[U]$ has exactly three vertices.
- **At least four cycle rule.** There exists a C_h $(v_1, v_2, v_3, \ldots, v_h)$ in $G[U]$ where $h \geq 4$. Then the algorithm branches on each of the following cases:
  - v_2 is black and $k := k - 1$;
  - v_2 is white and v_1, v_3 are black and $k := k - 2$;
  - v_2, v_3 are white and v_1, v_4 are black and $k := k - 2$; or
  - v_1, v_2 are white and v_3, v_h are black and $k := k - 2$.

  The worst case branching vector of this rule is $(1, 2, 2, 2)$ and the branching number is 2.3028. When the **At least four cycle rule** can not be applied, we see that any cycle in $G[U]$ has at most three vertices.

  When all reduction rules and all the above branching rules are not applicable, we see that the maximum degree of $G[U]$ is two. Moreover, all connected components in $G[U]$ are of three vertices, *i.e.,* they are either P_3 or C_3. Since the reduction rule, **No black neighbor triple rule** is not applicable, each of the uncolored vertex is adjacent to exactly one black vertex. By definition, the remaining graph is a partially black triple graph.

- **Partially black triple rule.** If $G = (B \uplus U, E)$ is a partially black triple graph, use the algorithm in Section 2 to solve the problem in time $O^*(2^{k-|B|})$.

Now we prove the correctness of the branching rules.

▶ **Lemma 11.** *One white neighbor rule is valid.*

Proof. Let v be an uncolored vertex having exactly one white neighbor u. Let x be an uncolored vertex adjacent to v or u. We see that v is either colored black or colored white in any optimal solution. If v is colored white, then x cannot be colored white, otherwise the set of white vertices in the optimal solution is not a bounded-degree-1 set, a contradiction. Thus, if v is colored white, then x must be colored black. This completes the proof. ◀

▶ **Lemma 12.** *Standard branching rule is valid.*

Proof. For an uncolored vertex v having h uncolored neighbors, $h \geq 3$, the Standard branching rule branches all $(h + 2)$ possibilities to color v and all its uncolored neighbors. In an optimal solution, either v is colored black or colored white. If v is white, either all its neighbors are all colored black or exactly one of its neighbor is colored white. By Lemma 5, if v and one of its neighbor are colored white, then all the uncolored neighbors of v must be colored black. This completes the proof. ◀

▶ **Lemma 13.** *Four path end rule is valid.*

Proof. Notice that if all the reduction rules and **One white neighbor rule** and **Standard branching rule** are not applicable, then G has no white vertices, each uncolored vertex v in G has at most one black neighbor, and each vertex in $G[U]$ is of degree at most two. Suppose that (v_1, v_2, v_3, v_4) is a P_4 in G and all of them are uncolored. Let $\mathcal{P}$ be an optimal solution of CCBD-1SB with minimum number of black vertices at most k. We see that v_2 is either colored black or white. Suppose that v_2 is colored white in the optimal solution $\mathcal{P}$. If v_1 is colored white in the optimal solution $\mathcal{P}$, then v_3 must be colored black. If v_3 is colored white in the optimal solution $\mathcal{P}$, then both v_1 and v_4 must be colored black. Suppose that v_2 is colored white and both v_1 and v_3 are colored black in the optimal solution $\mathcal{P}$. Since all the neighbors of v_2 are colored black and v_2 has no white neighbors, we can always obtain a solution $\mathcal{P}'$ from $\mathcal{P}$ by recoloring v_1 white. We see that $\mathcal{P}'$ has less black vertices than $\mathcal{P}$, a contradiction to the assumption that $\mathcal{P}$ is an optimal solution with minimum number of black vertices. This completes the proof. ◀

▶ **Lemma 14.** *At least four cycle rule is valid.*

Proof. Notice that if all the reduction rules and **One white neighbor rule**, **Standard branching rule**, and **Four path end rule** are not applicable, then G has no white vertices, each uncolored vertex v in G has at most one black neighbor, each vertex in $G[U]$ is of degree at most two, and $G[U]$ consists of cycles and P_3s. Suppose that $\{v_1, v_2, v_3, \ldots, v_h\}$ induces a cycle in G such that $(v_i, v_{i+1}), (v_1, v_h) \in E$ for $1 \leq i \leq h - 1$, and all of them are uncolored. Let $\mathcal{P}$ be an optimal solution of CCBD-1SB with minimum number of black vertices at most k. We see that v_2 is either colored black or white. Suppose that v_2 is colored white in $\mathcal{P}$. Then either both v_1 and v_3 are black in $\mathcal{P}$ or one of v_1 and v_3 is colored white in $\mathcal{P}$. If v_1 is colored white in the optimal solution $\mathcal{P}$, then v_h and v_3 must be colored black. If v_3 is colored white in the optimal solution $\mathcal{P}$, then v_1 and v_4 must be colored black. This completes the proof. ◀

Notice the worst branching number 2.4143 is obtained from **Standard branching rule** and **Four path end rule**. We now conclude this section with the following theorem and corollaries.

▶ **Theorem 15.** *The CCBD-1SB problem can be solved in* $O^*(2.4143^k)$ *time.*

▶ **Corollary 16.** *The* 2-PLEX BIPARTITION *problem and the* BOUNDED-DEGREE-1 SET BIPARTITION *problem can be solved in* $O^*(2.4143^k)$ *time.*

▶ **Corollary 17.** *The* 2-PLEX BIPARTITION *problem and the* BOUNDED-DEGREE-1 SET BIPARTITION *problem can be solved in* $O^*(1.5539^n)$ *time where* n *is the number of vertices in the input graph.*

Proof. By Corollary 16, the 2-PLEX BIPARTITION problem and the BOUNDED-DEGREE-1 SET BIPARTITION problem can be solved in $O^*(2.4143^k)$ time. According to the fact that $k \leq n/2$, we can design an exact algorithm by using the fixed-parameter algorithm for the 2-PLEX BIPARTITION problem and the BOUNDED-DEGREE-1 SET BIPARTITION problem as a subroutine. The running time of the exact algorithm is $O^*(2.4143^{n/2}) = O^*(1.5539^n)$. This completes the proof. ◀

4 Concluding remarks

In this paper, we give a fixed-parameter algorithm to solve the 2-plex bipartition problem in time $O^*(2.4143^k)$ where $k \leq n/2$ is an input parameter. It is of interesting to see whether there exist more efficient fixed-parameter algorithms to solve the s-plex bipartition problem for a constant $s \geq 2$. Moreover, it is even more interesting to see whether there exist fixed-parameter algorithms to solve the s-plex t-partition problem that asks whether the vertices of input the graph can be partitioned into t parts such that each part is an s-plex.

References

1 J. Andrews and M. Jacobson, On a generalization of chromatic number, *Congressus Numerantium*, **47** (1985), 33–48.

2 B. Balasundaram, Cohesive subgroup model for graph-based text mining, *Proceedings of the 2008 IEEE Conference on Automation Science and Engineering*, pp. 989–994, 2008.

3 B. Balasundaram, S. Butenko, and I. V. Hicks, Clique relaxations in social network analysis: The maximum k-plex problem, *Operations Research* 59 (2011), pp. 133–142.

4 R. van Bevern, H. Moser, and R. Niedermeier, Approximation and tidying–a problem kernel for s-plex cluster vertex deletion, *Algorithmica* 62 (2012), pp. 930–950.

5 M.-S. Chang, L.-J. Hung, and P.-C. Su, Exact and fixed-parameter algorithms for problems related to 2-plex, *Proceedings of the 2011 International Computer Science and Engineering Conference (ICSEC 2011)*, pp. 203–208, 2011.

6 M.-S. Chang, L.-H. Chen, L.-J. Hung, Y.-Z. Liu, P. Rossmanith, and S. Sikdar, An $O^*(1.4658^n)$-time exact algorithm for the maximum bounded-degree-1 set problem, *Proceedings of the 31st Workshop on Combinatorial Mathematics and Computation Theory*, pp. 9–18, 2014.

7 M.-S. Chang, L.-H. Chen, L.-J. Hung, P. Rossmanith, and P.-C. Su, Fixed-parameter algorithms for vertex cover P_3, *Discrete Optimization*, 19 (2016), 12–22.

8 L. Cowen, R. Cowen, and D. Woodall, Defective colorings of graphs in surfaces: partitions into subgraphs of bounded valence, *Journal of Graph Theory*, 10 (1986), 187–195.

9 R. Cowen, Some connections between set theory and computer science, in G. Gottlob, A. Leitsch, and D. Mundici (eds.), *Proceedings of the third Kurt Godel colloquium on Computational Logic and proof theory*, Lecture Notes in Computer Science, pp. 14–22, 1993.

10 L. Cowen, W. Goddard, and C. E. Jesurum, Defective coloring revisited, *Journal of Graph Theory*, 24 (1997), pp. 205–219.

11 M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, *Parameterized algorithms*, Springer, 2015.

12 R. G. Downey and M. R. Fellows, *Fundamentals of Parameterized Complexity*, Springer, 2013.

13 F. Fomin and D. Kratsch, *Exact Exponential Algorithms*, Springer, 2010.

14 M. Frick, A survey of (m, k)-colorings, in J. Gimbel, J. W. Kennedy, and L. V. Quintas (eds.), *Quo Vadis, Graph Theory?*, volume 55 of Annals of Discrete Mathematics, pp. 45–58. Elsevier Science Publisher, New York, 1993.

15 W. Goddard, Acyclic colorings of planar graphs, *Discrete Mathematics*, 91 (1991), 91–94.

16 F. Harary and K. Jones, Conditional colorability II: Bipartite variations, *Congressus Numerantium*, 50 (1985), 205–218.

17 T. Jensen and B. Toft, *Graph Coloring Problems*, John Wiley and Sons, New York (1995).

18 F. T. Leighton, A graph coloring algorithm for large scheduling problems, *Journal of Research of the Notional Bureau of Standards*, Vol. 84, No.6, November- December, 1979.

19 J. M. Lewis and M. Yannakakis, The node-deletion problem for hereditary properties is NP-complete, *Journal of Computer and System Sciences*, 20 (1980), 219–230.

20 V. Lotfi, A graph coloring algorithm for large scale scheduling problems, *Computers & Operations Research* 13 (1986), 27–32.

21 L. Lovász, On decomposition of graphs, *Studia Scientiarum Mathematicarum Hungarica* 1 (1966), pp. 237–238.

22 C. Papadimitriou, On selecting a satisfying truth assignment, In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computing (FOCS 1991)*, pp. 163–169, 1991.

23 K. Poh, On the linear vertex-arboricity of a planar graph, *Journal of Graph Theory*, 14 (1990), 73–75.

24 S. B. Seidman and B. L. Foster, A graph-theoretic generalization of the clique concept, *The Journal of Mathematical Sociology* 6 (1978), pp. 139–154.

25 D. Woodall, Improper colourings of graphs, in R. Nelson and R. J. Wilson (eds.), *Graph Colourings*, Longman Scientific and Technical (1990).

Smart Contract Execution – the (+−)-Biased Ballot Problem*

Lin Chen[1], Lei Xu[2], Zhimin Gao[3], Nolan Shah[4], Yang Lu[5], and Weidong Shi[6]

1 **Department of Computer Science, University of Houston, Houston, USA**
chenlin198662@gmail.com
2 **Department of Computer Science, University of Houston, Houston, USA**
xuleimath@gmail.com
3 **Department of Computer Science, University of Houston, Houston, USA**
mtion@hotmail.com
4 **Department of Computer Science, University of Houston, Houston, USA**
nolanshah212@gmail.com
5 **Department of Computer Science, University of Houston, Houston, USA**
ylu17@uh.edu
6 **Department of Computer Science, University of Houston, Houston, USA**
wshi3@uh.edu

Abstract

Transaction system build on top of blockchain, especially smart contract, is becoming an important part of world economy. However, there is a lack of formal study on the behavior of users in these systems, which leaves the correctness and security of such system without a solid foundation. Unlike mining, in which the reward for mining a block is fixed, different execution results of a smart contract may lead to significantly different payoffs of users, which gives more incentives for some user to follow a branch that contains a wrong result, even if the branch is shorter. It is thus important to understand the exact probability that a branch is being selected by the system. We formulate this problem as the (+−)-Biased Ballot Problem as follows: there are n voters one by one voting for either of the two candidates A and B. The probability of a user voting for A or B depends on whether the difference between the current votes of A and B is positive or negative. Our model takes into account the behavior of three different kinds of users when a branch occurs in the system – users having preference over a certain branch based on the history of their transactions, and users being indifferent and simply follow the longest chain. We study two important probabilities that are closely related with a blockchain based system – the probability that A wins at last, and the probability that A receives d votes first. We show how to recursively calculate the two probabilities for any fixed n and d, and also discuss their asymptotic values when n and d are sufficiently large.

1998 ACM Subject Classification G.3 Statistical computing, C.2.1 Distributed networks, F.1.2 Probabilistic computation

Keywords and phrases Blockchain, Probability, Random Walk, Smart Contract

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.21

1 Introduction

Blockchain technology provides a decentralized way for bookkeeping and has been proved to be a powerful tool in various areas, especially fintech applications [3]. Bitcoin [16], as the first blockchain based cryptocurrency system, is now accepted by more than 100,000

[*] A full version of the paper is available at [6], http://i2c.cs.uh.edu/tiki-download_wiki_attachment.php?attId=73&download=y.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 21; pp. 21:1–21:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

merchants worldwide [10], and the value of each bitcoin is more than $1,000. Besides purely cryptocurrency transactions, blockchain is also used to build decentralized smart contract platform, where the execution of contracts is enforced by all nodes participating the blockchain system [5]. This greatly extends the range of application of blockchain technology and could revolutionize world business [19].

Briefly speaking, a blockchain is a chain of blocks with each block containing some information (e.g., the transactions, the execution result of a program and so on) and the hash of the previous block. Users of the system keep appending blocks to the blockchain. Smart contracts can be built upon a blockchain system. A smart contract is an event-driven program which can be viewed as a contract in a decentralized system. In smart contract supported blockchain systems (e.g., Ethereum), a user runs the contract locally, puts the execution result of the smart contract in a block and then tries to append the block to the blockchain. Further details of a blockchain system can be found in Section 1.1.

Ideally, a blockchain always remains as a chain. However, intentionally or not, a user may append a new block after some other block instead of the one at the end of the current blockchain, yielding a branch (or "fork"). There are several different criteria for eliminating branches. One common criterion is the *longest-chain rule* [16], which ensures that the branch that first receives d blocks afterwards will be selected as the "legal" branch, that is, blocks in all the other branches will be neglected by the system.

Users of the system that are financially driven may behave strategically to maximize their own profit. In a blockchain system where a user is paid by a fixed amount of coins whenever he/she successfully appends a block (e.g., Bitcoin), it is natural that he/she will append a block to a branch that is most likely to be selected by the system eventually, which is essentially the current longest branch [16]. Things become much more complicated when smart contracts are involved, since the execution result of the contract may involve a huge gain or loss among users. Suppose there two branches A and B where different execution results of some smart contract are stored. A user who is involved in the smart contract can have strong pereference over these two branches. Indeed, let r_A (r_B) be the reward that the user can get if eventually A (B) is selected by the system. If $r_A > 100r_B$, then even if A is currently the shorter branch and will be selected by the system with the probability of 0.01, the user may still append blocks to A, trying to maximize his/her expected reward. Such a phenominon has been observed by Chen et al. [7]. To better understand and predict the behavior of users in a blockchain system supporting smart contracts like Ethereum, it is thus important to characterize the exact probability that certain branch gets selected, which is the goal of this paper.

1.1 The Blockchain Based Transaction System

A blockchain is a public ledger that records transactions between users [16]. Typically, it is a chain of blocks with each block containing some information (e.g., the transactions, the execution result of a program and so on) and the hash of the previous block. Consequently, the precedence order between blocks are fixed and it is possible to trace back from a recent block to the very first block in the system. Users of the system append blocks to the blockchain through a process called mining. They are called miners[1].

As blockchain is a decentralized system, one of the critical requirements is that all users have to reach consensus on the sequence and content of blocks. There are mainly two types of techniques involved in this process: block construction and chain selection. For block construction, the most common approach is proof-of-work [24], which is widely used in many blockchain based transaction systems, e.g., Bitcoin [16] and Ethereum [25]. Proof-of-work

[1] Throughout this paper, we are only concerned with such users, thus users and miners are used interchangeably.

requires the node to solve a hard problem to build a valid block, and the success probability is determined by physical resources (e.g., CPU and storage) owned by the node [15]. Because the amount of physical resources is relatively fixed, the probability for each node to produce a valid block is stable. It is worth mentioning that besides proof-of-work, there are also other protocols for block construction, e.g., proof-of-stake [4] and proof-of-elapsed-time [8]. However, as proof-of-work much more widely used than other protocols, we focus on proof-of-work throughout this paper and consequently, we treat the probability of producing a valid block as a fixed value. If more than one branches are generated, participating nodes have to decide which branch to follow to add new blocks. Common chain selection criteria include longest-chain rule [16] and GHOST (greedy heaviest-observed sub-tree) rule [14]. Byzantine fault tolerant protocol is also proposed to eliminate disagreements on the chain [9, 11]. Throughout this paper, we focus on the longest-chain rule.

A fixed reward[2] is paid to a user who successfully adds a block, thus giving incentive to mining. However, when a miner adds a block, it is not guaranteed that he/she always appends this block at the bottom of the chain. He/she may choose any previous block to append the new block or sometimes two or more users simultaneously add blocks after the same block yielding a branch. Eventually only one branch will be chosen as the "legal" branch and all others will be discarded. The longest chain rule is a common branch selection method used in Bitcoin and other blockchains. This rule chooses the branch that first accumulates d blocks for some constant d. Note that when a branch is discarded, miners in that branch do not receive any award, thus incentivizing users to follow the longest chain rule.

The strategic behavior of users may cause serious problems when a blockchain system includes smart contracts. A smart contract is an event-driven program which can be viewed as a contract in a decentralized system. In smart contract supported blockchain systems (e.g., Ethereum [25]), a user runs the contract locally, then publishes the execution result of the smart contract in a block. The user is rewarded for mining and contract execution. However, since a smart contract defines the payoff among users involved in the contract, different execution results may lead to significantly different payoffs of users involved in the contract. If a miner is directly or indirectly involved, he/she may want to branch the chain by adding a block containing a (right or wrong) result favorable to them. In this case, miners benefiting from the result will work on different branches in their favor, while indifferent users will simply work on the (temporarily) longer branch. Eventually, one branch will be selected by the system.

This poses the question, what is the probability that a certain branch is selected? This could be cast as a ballot problem: we view both branches as two candidates A and B. Every block added to the system is viewed as a vote and voting for A or B represents which branch the block is added to. As indifferent users choose the longer branch, the probability that a voter votes for A or B is subject to variation on the current number of votes A and B get. We give the formal definition of the ballot problem in the following subsection.

1.2 Problem Statement

We propose the following model to study the behavior of users in a blockchain based smart contract system.

(+−)-Biased Ballot Problem. Suppose there are two candidates A and B. There are n voters one by one making their votes. Each voter either votes for A or B. The probability that the voter votes for A or B depends on the total number of votes received by A and B at the time he votes. Specifically, let n_A^i and n_B^i be the number of votes received by A and B when voter i votes, respectively. Voter i votes for A and B with probability p_+ and

[2] The reward may be adjusted after a significantly long time. In the short term, it is treated as fixed.

$q_+ = 1 - p_+$, if $n_A^i - n_B^i > 0$; with probability p_- and $q_- = 1 - p_-$, if $n_A^i - n_B^i < 0$; with probability p_0 and $q_0 = 1 - p_0$, if $n_A^i - n_B^i = 0$. We are interested in the following two questions in this paper. First, what is the probability that A receives more votes than B at last? Second, if there are infinite voters (indeed, $n \geq 2d$ suffices), what is the probability that A first receives d votes.

An equivalent formulation of the problem under the framework of random walk is the following:

1-Dimensional (+−)-Biased Random Walk. Suppose there is a ball located at the x-axis. Let L_i be the location of the ball after i-steps. Originally, $L_0 = 0$.

$$\mathrm{P}(L_{i+1} - L_i = 1) = \begin{cases} p_+, & \text{if } L_i > 0 \\ p_-, & \text{if } L_i < 0 \\ p_0, & \text{if } L_i = 0 \end{cases}$$

and $\mathrm{P}(L_{i+1} - L_i = -1) = 1 - \mathrm{P}(L_{i+1} - L_i = 1)$.

Note that $L_{i+1} - L_i = 1$ implies that voter $i+1$ votes for A, and $L_{i+1} - L_i = -1$ implies that he votes for B. Therefore, the probability that A receives more votes than B at last is exactly $\mathrm{P}(L_n > 0)$. We call it the ending probability. The event that A receives d votes first is denoted as H_d. We call $\mathrm{P}(H_d)$ as the hitting probability.

We discuss the random walk that starts at the origin in Section 3. In general, the random walk need not start at the origin, i.e., L_0 could be an arbitrary integer. We provide the result as Theorem 12. The reader may refer to the full version [6] of this paper for details.

▶ Remark. In our model, the two candidates A and B represent the two chains a blockchain branches into. Users' preferences over the two chains, based on the history of their transactions, are indicated by the parameters $p_+, q_+, p_0, q_0, p_-, q_-$. Suppose the probability of users favoring chain A is p, users favoring chain B is q, and users being indifferent is λ. Further, we assume that users being indifferent will always add a block to the longer chain, and when A and B have the same length, they add a block randomly. In this case, parameters will take the following values.

$$\begin{aligned} p_+ &= p + \lambda, & p_0 &= p + \lambda/2, & p_- &= p \\ q_+ &= q, & q_0 &= q + \lambda/2, & q_- &= q + \lambda \end{aligned}$$

The ending and hitting probabilities will indicate the chance that eventually A or B becomes the chain that is accepted by the system. Weighing such probabilities against their potential gain or loss (due to the history of their transactions recorded on A and B), users may decide on whether to follow the chain they prefer, or to follow the other chain if the probability that the chain they prefer has too low probability of being accepted by the system.

1.3 Related Work on Random Walk and the Ballot Problem

We give a brief overview of random walk and the ballot problem.

Bertrand's Ballot Problem. In an election where candidate A receives p votes and candidate B receives q votes with $p > q$, what is the probability that A will be strictly ahead of B throughout the count?

The ballot problem is a classical problem in combinatorics whose study dates back to 1878 by Whitworth [13]. The answer of the problem is $\frac{p-q}{p+q}$, which is known as the Bertrand's ballot theorem and could be proved via various approaches, e.g., by a recursion relation [13] or by Andre's reflection method [17].

An equivalent problem is also studied in the context of random walk [1]. Consider the *1-dimensional symmetric simple random walk* on $\mathbb{Z}$, that is, let the random walk start at the origin $L_0 = 0$ and $L_{i+1} = L_i \pm 1$, each with probability $1/2$ and independent of other steps. What is the probability that $L_i > 0$ for all $1 \leq i \leq p+q$ conditioned on $L_{p+q} = p-q$? This probability is given precisely by the Bertrand's ballot theorem. The number of all such random walks is studied in [2]. Various generalizations of the ballot problem in the context of random walk has been studied by Takacs [20, 21, 22].

Random walk, as a general subject, has been studied extensively in the literature. We refer the readers to [23] as a nice survey on geometric random walk and [1] as a nice survey on various ballot problem-related results when the ballot problem is viewed as a random walk.

In recent years, research on the ballot problem and its extensions has found applications in the study of the blockchain based transaction systems. In his seminal paper, Nakamoto [16] introduces the Bitcoin system and uses the result from the ballot problem to study the security of the system. Indeed, the security problem in Bitcoin could be cast as the following modified ballot problem: suppose each voter votes independently with probability p to A and probability $q = 1-p$ to B, what is the probability that A is always strictly ahead of B? Such a model is also adopted in a series of subsequent studies [18, 12].

1.4 Our Contribution

We study the strategic behavior of users in a blockchain based transaction system. Instead of adopting the classical model that assumes attackers and honest users, in our model we assume there are three groups of users when the blockchain branches into chain A and chain B, with two groups favoring chain A and chain B respectively, and the third group being indifferent and favoring the longer chain. We formulate our model as the $(+-)$-Biased Ballot Problem where the two candidates A and B represent the two chains. In our model, we do not necessarily require that chain A is always ahead of chain B. Indeed, we care about the probability that A exceeds B after n blocks are generated, which we call the ending probability, and the probability that A is extended by d blocks at first, which we call the hitting probability. That means, we also take into account of the possibility that chain A is temporarily behind chain B but takes over later on. However, once A is behind B, the probability that the next block is added to A will be adjusted. In the extreme case when $p_- = 0$, i.e., when A is behind no blocks will be added to A, our model reduces to the classical model in which we only consider the probability that A is always ahead of B.

We show how to calculate the ending and hitting probability for the $(+-)$-Biased Ballot Problem and study their asymptotic values when n and d are sufficiently large. Applying our results to blockchain, we show that, if the third group (i.e., users that are indifferent) has much larger probability of generating the next block, then the probability that chain A wins eventually is roughly $0.5 + \theta\Delta$, where Δ is the difference in the probability of generating a block between users favoring A and users favoring B, and $\theta \in [3/4, 3/2]$. We further consider the ballot problem starting at the situation that A is already ahead of B by s votes. Let μ_s be the probability that A wins eventually, then for sufficiently large n the probability chain B can win is no more than $(1-\mu_0)(q_+/p_+)^k$ when $p_+ > q_+$. In the classical ballot problem, the probability that B can catch up by k blocks is $(q_+/p_+)^k$, as is shown by Nakamoto [16] in the Bitcoin system. In our model, however, this probability is further reduced by a factor of $1-\mu_0$.

2 Preliminaries

Bailey's number [2]. Consider a sequence of numbers $x_1, x_2, \cdots, x_{n+r}$ where $x_i \in \{-1, 1\}$. In total, the number of 1's and -1's in the sequence is n and r, respectively. Furthermore, $\sum_{i=1}^{j} x_i \geq 0$ holds for any $1 \leq j \leq n+r$. The number of all such sequences is denoted as

$\sigma(n, r)$ in this paper. It is shown by Bailey that

$$\sigma(n, r) = \frac{n+1-r}{n+1} \cdot \binom{n+r}{r}.$$

Similar as the binomial coefficient, the followings are true for the Bailey's number [2]:

$$\sigma(n, r) = \sigma(n-1, r) + \sigma(n, r-1), \quad 2 \leq r \leq n-1 \tag{1}$$

$$\sigma(n, n-1) = \sigma(n, n). \tag{2}$$

Specifically, if $r = n$, $\sigma(n, r)$ becomes the same as the n-th Catalan number. We will make use of the following generating function for the Catalan number:

$$\sum_{n=0}^{\infty} \sigma(n, n) x^n = \frac{2}{1 + \sqrt{1-4x}}. \tag{3}$$

3 Solution for the (+−)-Biased Ballot Problem

In this section, we show how to calculate the ending and hitting probability for the (+−)-Biased Ballot Problem for any fixed n and d. We also discuss their asymptotic values when n and d are sufficiently large. We focus on the random walk version of the problem. We consider the case that the random walk starts at the origin, i.e., $L_0 = 0$. The reader may refer to to the full version [6] oft this paper for the random walk that starts at an arbitrary location.

Let Λ_k be the event that the random walk starts at $L_0 = 0$, and returns to 0 for the first time after $2k$ steps. Furthermore, we define

$$\Lambda_k^+ = \Lambda_k \cap \{L_i > 0, \forall 1 \leq i \leq 2k\}, \quad \Lambda_k^- = \Lambda_k \cap \{L_i < 0, \forall 1 \leq i \leq 2k\}.$$

Consequently, $\Lambda_k = \Lambda_k^+ \cup \Lambda_k^-$.

Consider the event $L_n > 0$. There are two possibilities, either the random walk starts at 0, goes right and never returns to 0 within n steps, or it returns to 0 for the first time after $n' < n$ steps, implying that after n' steps it re-starts at 0, and ends at some positive location after $n - n'$ steps. Notice that n' must be even. Let B_n^+ be the event that it goes right and never returns to 0 within n steps, we have the following recursive calculation:

$$\mathrm{P}(L_n > 0) = \mathrm{P}(B_n^+) + \sum_{k=1}^{\lfloor n/2 \rfloor} \mathrm{P}(\Lambda_k) \cdot \mathrm{P}(L_{n-2k} > 0) \tag{4}$$

Here $\mathrm{P}(\Lambda_k) \cdot \mathrm{P}(L_{n-2k} > 0)$ is the probability of the event that the random walk starts at 0, goes right, returns to 0 for the first time after $2k$ steps, and then re-start at 0, ends at some positive location after $n - 2k$ additional steps.

We can apply similar arguments for the hitting probability. Let $H_d = \cup_{j=1}^{d} \{L_{2d-j} = j\}$ denote the event that the random walk goes right for d steps and goes left for less than d steps. There are two possibilities, either the random walk starts at 0, goes right and never returns to 0, or it returns to 0 for the first time after $2k$ steps, and the event H_{d-k} happens afterwards. Let D_d^+ denote the event that the random walk never goes back to 0, then we have the following.

$$\mathrm{P}(H_d) = \mathrm{P}(D_d^+) + \sum_{k=1}^{d-1} \mathrm{P}(\Lambda_k) \cdot \mathrm{P}(H_{d-k}) \tag{5}$$

The recursive formulas (4) and (5) has a very similar structure. In the following we show how to calculate $\mathrm{P}(B_n^+)$, $\mathrm{P}(D_n^+)$ and $\mathrm{P}(\Lambda_k)$, whereas for every fixed n and d we can always calculate $\mathrm{P}(L_n > 0)$ and $\mathrm{P}(H_d)$ recursively.

▶ **Lemma 1.**

$$\mathrm{P}(B_n^+) = p_0 \cdot \sum_{0 \le r \le (n-1)/2} \sigma(n-1-r, r) p_+^{n-1-r} q_+^r,$$

where $\sigma(n-1-r, r) = \binom{n-1}{r} \cdot \frac{n-2r}{n-r}$.

Proof. The first step of the random walk should go right, which happens with probability p_0. We denote by $x_i = L_{i+1} - L_i \in \{-1, 1\}$. Consider the case that from the second step to the n-th step, there are in total r steps that go left and $n-1-r$ steps go right. To make sure that the random walk never goes back to 0, we have

$$\sum_{j=1}^{h} x_j \ge 0, \quad \forall 1 \le h \le n-1. \tag{6}$$

Note that when $h = n-1$, we have $n-1-r-r \ge 0$, whereas $r \le (n-1)/2$.

According to the definition of the Bailey's number, the number of integral solutions for Inequality (6) is $\sigma(n-1-r, r)$, and the probability that each solution happens is $p_+^{n-1-r} q_+^r$. Hence, the lemma is proved. ◀

For D_d^+ the argument is exactly the same.

▶ **Lemma 2.**

$$\mathrm{P}(D_d^+) = p_0 \cdot \sum_{0 \le r \le d-1} \sigma(d-1, r) p_+^{d-1} q_+^r,$$

where $\phi(d, r) = \binom{d+r}{r} \cdot \frac{d+1-r}{d+1}$.

Proof. The first step of the random walk should go right, which happens with probability p_0. Again let $x_i = L_{i+1} - L_i \in \{-1, 1\}$. Suppose starting from the second step, there are in total r steps that go left. Notice that in total there are $d-1$ steps go right (excluding the first step) where $0 \le r \le d-1$, then we have

$$\sum_{j=1}^{h} x_j \ge 0, \quad \forall 1 \le h \le d+r-1.$$

The number of integral solutions for the above is $\sigma(d-1, r)$, and the probability that each solution happens is $p_+^{d-1} q_+^r$. Hence, the lemma is proved. ◀

Now we consider Λ_k.

▶ **Lemma 3.**

$$\mathrm{P}(\Lambda_k^+) = p_0 \cdot \sigma(k-1, k-1) p_+^{k-1} q_+^k,$$

where $\sigma(k-1, k-1) = \binom{2k-2}{k-1}/k$.

Proof. Consider Λ_k^+. Starting at 0, the first step of the random walk must go right, and the last step must go left (from 1 to 0). From step 2 to step $2k-1$, we consider $x_i = L_{i+1} - L_i \in \{-1, 1\}$, then it is easy to see that

$$\sum_{j=1}^{h} x_j \ge 0, \quad \forall 1 \le h \le 2k-2.$$

Therefore, the number of all possible solutions is given by the Bailey's number $\sigma(k-1, k-1)$. Each solution occurs with probability $p_+^{k-1} q_+^{k-1}$. Further multiplying the probability of the first and last step $p_0 q_+$, the lemma is proved. ◀

Using the same argument, we have

▶ **Lemma 4.**

$$\mathrm{P}(\Lambda_k^-) = q_0 \cdot \sigma(k-1, k-1) p_-^k q_-^{k-1},$$

where $\sigma(k-1, k-1) = \binom{2k-2}{k-1}/k$*.*

With the above lemmas, for any fixed n and d we can always recursively calculate $\mathrm{P}(L_n > 0)$ and $\mathrm{P}(H_d)$. In the following we discuss their values when n and d go to infinity, which provide an estimation of the two probabilities when n and d are sufficiently large.

▶ **Lemma 5.** *If* $p_+ > q_+$*,*

$$\lim_{n\to\infty} \mathrm{P}(B_n^+) = p_0 \cdot \frac{p_+ - q_+}{p_+}.$$

Otherwise, $\lim_{n\to\infty} \mathrm{P}(B_n^+) = 0$*.*

To show the above lemma, we need the following.

▶ **Lemma 6** ([13], pp.272)**.** *Consider a random walk starting at* $L_0 = 0$*,* $\mathrm{P}(L_{i+1} - L_i = 1) = p$*,* $\mathrm{P}(L_{i+1} - L_i = -1) = q$ *where* $p + q = 1$*. If* $p > q$*, then*

$$\lim_{n\to\infty} \mathrm{P}(L_i \geq 0, \forall 1 \leq i \leq n) = \frac{p-q}{p}.$$

If $p < q$*, the above limit is* 0*.*

Proof of Lemma 5. This infinite summation could be calculated directly by plugging in Lemma 1 and using generating functions. An easier way, however, is to apply Lemma 6. Note that

$$B_n^+ = \{L_0 = 0, L_1 = 1, L_2 \geq 1, L_3 \geq 1, \cdots, L_n \geq 1\}.$$

From the second step to the n-th step, the event B_n^+ could be viewed as starting at 1 and always remaining at the rightside of 1 (including 1 itself). Note that in this case the probability of going left is always q_+ and going right is always p_+, hence, if $p_+ > q_+$

$$\lim_{n\to\infty} \mathrm{P}(B_n^+) = p_0 \cdot \frac{p_+ - q_+}{p_+}.$$

Otherwise, $\lim_{n\to\infty} \mathrm{P}(B_n^+) = 0$. ◀

▶ **Lemma 7.** *If* $p_+ > q_+$*,*

$$\lim_{d\to\infty} \mathrm{P}(D_d^+) = p_0 \cdot \frac{p_+ - q_+}{p_+^2}.$$

Otherwise, $\lim_{d\to\infty} \mathrm{P}(D_d^+) = 0$*.*

The proof is a bit involved, the reader is referred to the full version [6] of this paper for details.

Next we consider Λ_k.

▶ **Lemma 8.**

$$\sum_{k=1}^{\infty} \mathrm{P}(\Lambda_k^+) = \frac{2p_0 q_+}{1 + \sqrt{1 - 4p_+ q_+}}.$$

Proof. Using the generating function (3), we know

$$\sum_{k=1}^{\infty} \mathrm{P}(\Lambda_k^+) = \sum_{k=1}^{\infty} p_0 \sigma(k-1, k-1) p_+^{k-1} q_+^k = p_0 q_+ \sum_{k=0}^{\infty} \sigma(k,k) p_+^k q_+^k = \frac{2p_0 q_+}{1+\sqrt{1-4p_+q_+}}.$$

◀

Similarly, we have

▶ **Lemma 9.**

$$\sum_{k=1}^{\infty} \mathrm{P}(\Lambda_k^-) = \frac{2q_0 p_-}{1+\sqrt{1-4p_-q_-}}.$$

▶ **Theorem 10.** *If* $p_+ \leq q_+$, $\lim_{n\to\infty} \mathrm{P}(L_n > 0) = 0$. *Otherwise,*

$$\lim_{n\to\infty} \mathrm{P}(L_n > 0) = \frac{p_0 \cdot \frac{p_+ - q_+}{p_+}}{1 - \frac{2p_0q_+}{1+\sqrt{1-4p_+q_+}} - \frac{2q_0p_-}{1+\sqrt{1-4p_-q_-}}}$$

Proof. The limit of $\mathrm{P}(L_n > 0)$ could be calculated in the following way. Given that

$$\sum_{k=1}^{\infty} \mathrm{P}(\Lambda_k) = \sum_{k=1}^{\infty} (\mathrm{P}(\Lambda_k^+) + \mathrm{P}(\Lambda_k^-)) = \frac{2p_0q_+}{1+\sqrt{1-4p_+q_+}} + \frac{2q_0p_-}{1+\sqrt{1-4p_-q_-}},$$

For N being sufficiently large, we know

$$0 \leq \sum_{k=N}^{\infty} \mathrm{P}(\Lambda_k) \leq \epsilon_N,$$

whereas for $n > 2N$ we have

$$\mathrm{P}(B_n^+) + \sum_{k=1}^{N} \mathrm{P}(\Lambda_k)\mathrm{P}(L_{n-2k} > 0) \leq \mathrm{P}(L_n > 0) \leq \mathrm{P}(B_n^+) + \sum_{k=1}^{N} \mathrm{P}(\Lambda_k)\mathrm{P}(L_{n-2k} > 0) + \epsilon_N.$$

Let n goes to infinity and let $\mu_0 = \lim_{n\to\infty} \mathrm{P}(L_n > 0)$, $\nu_0 = \lim_{n\to\infty} \mathrm{P}(B_n^+)$, the following holds for any fixed integer N,

$$\nu_0 + \sum_{k=1}^{N} \mathrm{P}(\Lambda_k)\mu_0 \leq \mu_0 \leq \nu_0 + \sum_{k=1}^{N} \mathrm{P}(\Lambda_k)\mu_0 + \epsilon_N.$$

Let N goes to infinity, we know ϵ_N goes to 0, hence

$$\mu_0 = \nu_0 + \sum_{k=1}^{\infty} \mathrm{P}(\Lambda_k)\mu_0.$$

Plug in ν_0, if $p_+ \leq q_+$, $\lim_{n\to\infty} \mathrm{P}(L_n > 0) = 0$, otherwise,

$$\mu_0 = \lim_{n\to\infty} \mathrm{P}(L_n > 0) = \frac{p_0 \cdot \frac{p_+ - q_+}{p_+}}{1 - \frac{2p_0q_+}{1+\sqrt{1-4p_+q_+}} - \frac{2q_0p_-}{1+\sqrt{1-4p_-q_-}}}$$

◀

Using the same argument, we have the following theorem.

▶ **Theorem 11.**

$$\lim_{d\to\infty} \mathrm{P}(H_d) = \frac{p_0 \cdot \frac{p_+ - q_+}{p_+^2}}{1 - \frac{2p_0q_+}{1+\sqrt{1-4p_+q_+}} - \frac{2q_0p_-}{1+\sqrt{1-4p_-q_-}}}.$$

So far we have discussed the random walk starting at the origin. Indeed, if the random walk starts at some point $s > 0$ (the case $s < 0$ could be handled in the same way), then its ending probability μ_s satisfies the following (See the full version [6] of this paper for details).

▶ **Theorem 12.** *For $s \geq 1$, if $p_+ > q_+$*

$$\mu_s = 1 - (1-\mu_0)(\frac{q_+}{p_+})^s.$$

Otherwise, $\mu_s = 0$.

3.1 Application to Blockchain

In this subsection we apply our results to the blockchain system. Specifically, we estimate the value of ending probability when the parameters $p_+, q_+, p_0, q_0, p_-, q_-$ are taking certain fixed values. Note that these parameters are taken as fixed constants.

We first consider Theorem 12. Suppose A represents the main chain and B represents some private chain of attackers. Users favoring the main chain A are the honest users in the traditional studies of blockchain, users favoring chain B are the attackers, and users being indifferent (the third group of users) are actually the majority of users in the system – they do not really have a preference but will just follow the longer chain. Theorem 12 implies that if the main chain on which the honest users keep working on is already k blocks ahead, then the attackers can win with a probability no more than $(1-\mu_0)(\frac{q_+}{p_+})^s$ as long as $p_+ > q_+$, that is, as long as the honest users together with the third group of users can generate the next block with the probability of more than 50%.

We now give a more detailed analysis. Suppose the probability of users favoring chain A is p, users favoring chain B is q, and users being indifferent is λ. We assume that the third group of users, i.e., users being indifferent will always add a block to the longer chain, and when A and B have the same length, they add a block randomly since it makes no difference to them. In this case, we have the following.

$$\begin{aligned} p_+ &= p+\lambda, \qquad & p_0 &= p+\lambda/2, \qquad & p_- &= p \\ q_+ &= q, \qquad & q_0 &= q+\lambda/2, \qquad & q_- &= q+\lambda \end{aligned}$$

If users favoring A are so powerful that $p \geq q+\lambda$, then μ_0 could be simplified as

$$\mu_0 = \lim_{n\to\infty} \mathrm{P}(L_n > 0) = \frac{(p+\frac{1}{2}\lambda)\cdot\frac{p-q+\lambda}{p+\lambda}}{1 - \frac{2(p+\frac{1}{2}\lambda)q}{1+p+\lambda-q} - \frac{2(q+\frac{1}{2}\lambda)p}{1+p-q-\lambda}}$$

Simple calculations show that it becomes 1, that means A can always win. If $q \geq p+\lambda$, however, then the probability is 0.

In the following we assume that $\lambda > |p-q|$, whereas $p_+ > q_+$ and $q_- > p_-$. Under this condition, we can simplify the formula in Theorem 10 as follows,

$$\mu_0 = \frac{p_0\cdot\frac{p_+-q_+}{p_+}}{1-\frac{p_0q_+}{p_+}-\frac{p_-q_0}{q_-}} = \frac{\frac{(p+\lambda/2)(p-q+\lambda)}{p+\lambda}}{1-\frac{q(p+\lambda/2)}{p+\lambda}-\frac{p(q+\lambda/2)}{q+\lambda}}$$

We can rewrite μ_0 as

$$\mu_0 = \frac{p+\frac{1}{2}\lambda-\frac{1}{2}q-\frac{\frac{1}{2}pq}{p+\lambda}}{1-\frac{1}{2}p-\frac{1}{2}q-\frac{\frac{1}{2}pq}{p+\lambda}-\frac{\frac{1}{2}pq}{q+\lambda}}$$

If p and q are tiny compared with λ, say, $p, q \leq 0.1$, then we can ignore $pq/(p+\lambda)$ and $pq/(q+\lambda)$ and derive the following

$$\mu_0 \approx \frac{1}{2} + \frac{\frac{3}{2}p - \frac{3}{2}q}{1+\lambda}.$$

That means, if A is more powerful in generating blocks than B but is still not as powerful as the third party, then his probability of winning the game is larger than half by approximately $\frac{3}{2(1+\lambda)}(p-q) \in [\frac{3}{4}(p-q), \frac{3}{2}(p-q)]$.

4 Conclusion

We study the behavior of users in a blockchain system supporting smart contracts. As different execution results of a smart contract can lead to significantly different payoffs among users, it becomes critical to know the exact probability that a certain branch is selected by the system. We propose a generalized model called the (+−)-Biased Ballot Problem. In our model, we classify users into three groups when the blockchain branches into two chains A and B: the group of user favoring chain A, the group of users favoring chain B and the third group of users that are indifferent and will simply follow the longer chain. Instead of requiring one chain, say, chain A, to be always ahead of the other chain, we allow chain A to be temporarily behind chain B by considering the probability that chain A exceeds B after n blocks are generated (which we call the ending probability), and the probability that chain A exceeds B and when it is extended by d blocks (which we call the hitting probability). We provide recursive equations which allow us to compute the ending and hitting probabilities for any fixed n and d, and discuss the asymptotic values of the ending and hitting probabilities when n and d are sufficiently large.

References

1 L Addario-Berry and BA Reed. Ballot theorems, old and new. In *Horizons of combinatorics*, pages 9–35. Springer, 2008.

2 DF Bailey. Counting arrangements of 1's and-1's. *Mathematics Magazine*, 69(2):128–131, 1996.

3 Shagun Bali and Terry Roche. Blockchain technology: Pushing the envelope in fintech. In *Industry report TABB Forum*, 2015.

4 Vitalik Buterin. What proof of stake is and why it matters. *Bitcoin Magazine, August*, 26, 2013.

5 Vitalik Buterin. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

6 Lin Chen, Lei Xu, Zhimin Gao, Nolan Shah, Yang Lu, and Weidong Shi. Smart contract execution – the (+−)-biased ballot problem. `http://i2c.cs.uh.edu/tiki-download_wiki_attachment.php?attId=71&download=y`.

7 Lin Chen, Lei Xu, Zhimin Gao, Nolan Shah, Yang Lu, and Weidong Shi. Decentralized execution of smart contracts: Agent model perspective and its implications. In *1st Workshop on Trusted Smart Contracts*, 2017.

8 Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *19th Annual International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2017.

9 Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

10 Anthony Cuthbertson. Bitcoin now accepted by 100,000 merchants worldwide, 2015.

11 Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

12 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

13 William Feller. *An introduction to probability theory and its applications: volume I*, volume 3. John Wiley & Sons New York, 1968.

14 Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

15 Debin Liu and L Jean Camp. Proof of work can work. In *WEIS*, 2006.

16 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

17 Marc Renault. Lost (and found) in translation: Andre's actual method and its application to the generalized ballot problem. *The American Mathematical Monthly*, 115(4):358–363, 2008.

18 Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *arXiv preprint arXiv:1507.06183*, 2015.

19 Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

20 Lajos Takács. A generalization of the ballot problem and its application in the theory of queues. *Journal of the American Statistical Association*, 57(298):327–337, 1962.

21 Lajos Takács. The distribution of the majority in a ballot. *journal for probability theory and related fields*, 2(2):118–121, 1963.

22 Lajos Takács. Fluctuations in the ratio of scores in counting a ballot. *Journal of Applied Probability*, 1(02):393–396, 1964.

23 Santosh Vempala. Geometric random walks: a survey. *Combinatorial and computational geometry*, 52(573-612):2, 2005.

24 Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

25 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

Study of a Combinatorial Game in Graphs Through Linear Programming[*][†]

Nathann Cohen[1], Fionn Mc Inerney[2], Nicolas Nisse[3], and Stéphane Pérennes[4]

1 **CNRS, Univ Paris Sud, LRI, Orsay, France**
nathann.cohen@gmail.com
2 **Université Côte d'Azur, Inria, CNRS, I3S, France**
fionn.mc-inerney@inria.fr
3 **Université Côte d'Azur, Inria, CNRS, I3S, France**
nicolas.nisse@inria.fr
4 **Université Côte d'Azur, Inria, CNRS, I3S, France**
stephane.perennes@cnrs.fr

Abstract

In the *Spy Game* played on a graph G, a single spy travels the vertices of G at speed s, while multiple slow guards strive to have, at all times, one of them within distance d of that spy. In order to determine the smallest number of guards necessary for this task, we analyze the game through a Linear Programming formulation and the *fractional strategies* it yields for the guards. We then show the equivalence of fractional and integral strategies in trees. This allows us to design a polynomial-time algorithm for computing an optimal strategy in this class of graphs. Using duality in Linear Programming, we also provide non-trivial bounds on the fractional guard-number of grids and torus. We believe that the approach using fractional relaxation and Linear Programming is promising to obtain new results in the field of combinatorial games.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization

Keywords and phrases Turn-by-turn games in graphs, Graph algorithms, Linear Programming

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.22

1 Introduction

Turn-by-turn combinatorial games in graphs involve two players placing their pawns on vertices of a graph and moving them along its edges in order to achieve some task. For instance, in *Cops and Robber* games, Player 1 has a team of *cops* that must collaborate to capture a *robber* moved by the second player [5, 21, 22]. In the *surveillance game*, Player 1 has no pawn but is allowed to cover vertices at each of its turns, while the goal of Player 2 is to move its surfer to an uncovered vertex [12]. Another example is the *Eternal Domination Problem* in which Player 2 has no pawn but is allowed to *attack* any vertex at each of its turns, while the goal of Player 1 is to always be able to move at least one of its cops to the attacked vertex [16]. Most of these games have been studied because they model

[*] This work has been partially supported by ANR project Stint under reference ANR-13-BS02-0007, ANR program "Investments for the Future" under reference ANR-11-LABX-0031-01, the associated Inria team AlDyNet.

[†] Due to lack of space, several proofs have been omitted and can be found in [9], https://hal.archives-ouvertes.fr/hal-01462890.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 22; pp. 22:1–22:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

natural problems involving mobile agents cooperating to perform some task (e.g., network security, robot motion planning). Surprisingly, these games can also be used to provide a novel understanding of problems arising in telecommunication networks: for instance, the surveillance game was introduced for modelling resources prefetching [12].

In all these games, the goal is to minimize the amount of resources (e.g., the number of cops) ensuring the victory of one of the players. These combinatorial problems are generally "hard": Cops and Robber games are EXPTIME-complete [18] and W[2]-hard [13] and the Surveillance game is NP-hard and even PSPACE-complete in directed graphs [12]. Moreover, many longstanding (probably difficult) open questions have not been solved yet for these games. For instance, the celebrated Meyniel's conjecture states that $O(\sqrt{n})$ cops are sufficient to capture a robber in any n-node graph [24], Schröder asks whether $g+3$ cops are sufficient in any graph with genus at most g [23], the status of the complexity of Eternal Domination is still unknown, etc. A classical approach to tackle these open problems has been to study variants of these games in which one of the players often has some restrictions. For instance, the robber may be faster than the cops [2, 13], the cops may capture at some distance [4], the surveyed area may be forced to be connected [14], etc. Another approach may be to restrict the games to particular graph classes such as trees [19], grids [17, 20], planar graphs [1], etc.

Recently, some of the authors of the present paper proposed a new framework that considers a fractional variant of these games (roughly where pawns may be split into arbitrarily small entities) and uses Linear Programming to obtain new bounds and algorithms [8, 15]. While this approach seems not to be successful to handle Cops and Robber games, it has been fruitful in designing approximation algorithms for other combinatorial games. Precisely, it allowed to design polynomial-time approximation algorithms for various (NP-hard) variants of the surveillance game [15]. In this paper, we present a new successful application of this approach. In particular, we consider the *Spy-game* [8, 10] and show that it can be solved in polynomial-time in trees using this approach. We emphasize that, as far as we know, it is the first exact algorithm for such combinatorial games using a Linear Programming approach and that we were not able to solve it without this technique. We hope that our results will encourage people to use this framework to study combinatorial games and we believe it will enable progress toward solutions of the difficult open problems.

Spy-game. The *Spy-game* has been defined as it is closely related to the Cops and fast robber game and it generalizes the Eternal Domination Problem [8, 10]. The Spy-game is a turn-by-turn 2-Player game with perfect information. The first player has a *spy* which is first placed at some vertex of a graph G. The second player has $k \in \mathbb{N}$ *guards* that are then placed at some vertices of G. Turn-by-turn, the spy may move along at most $s \geq 1$ edges ($s \in \mathbb{N}^*$ is the *speed* of the spy), and then each guard may move along one edge. Any number of guards and the spy may occupy the same vertex. The goal of the game is to minimize the number of guards, called *guard-number* and denoted by $gn_{s,d}(G)$, ensuring that, at every step after the guards' turn, the spy is at distance at most $d \in \mathbb{N}$ from at least one guard (we say that the spy is *controlled* at distance d). Note that, when $d = 0$ and s is large (at least the diameter of G), the Spy-game is equivalent to the Eternal Domination Problem. The guard-numbers of paths and cycles, and corresponding optimal guards' strategies, have been characterized in [8]. To tackle the more difficult case of trees, we consider the fractional variant of the Spy-game, in which the rules are unchanged for the spy but the guards can be split into arbitrarily small entities.

Fractional Spy-game. Formally, the fractional Spy-game proceeds as follows in a graph $G = (V, E)$. Let $s \geq 2$, $d \geq 0$ be two integers and let $k \in \mathbb{R}$ such that $k > 0$. First, the spy is placed at a vertex. Then, each vertex v receives some amount $g_v \in \mathbb{R}^+$ (a non negative real) of guards such that the total amount of guards is $\sum_{v \in V} g_v = k$. Then, turn-by-turn, the spy may first move at distance at most s from its current position. Then, the "fractional" guards move following a flow constrained as follows. For any $v \in V$, and for any $u \in N[v]$[1], there is a flow $f(v, u) \in \mathbb{R}^+$ of guards going from v to $u \in N[v]$, such that $\sum_{u \in N[v]} f(v, u) = g_v$, i.e., the amount of guards leaving v and staying at v is exactly what was at v. Finally, for any vertex $v \in V$, the amount of guards occupying v at the end of the turn is $g'_v = \sum_{u \in N[v]} f(u, v)$. We now need to rephrase the fact that the guards control the spy at distance d at every turn. This is the case if, after every guards' turn, $\sum_{w \in N_d[x]} g'_w \geq 1$, where x is the vertex occupied by the spy. Let $fgn_{s,d}(G)$ denote the minimum total amount of fractional guards needed to always control at distance d a spy with speed s in a graph G. Note that, by definition, since the fractional game is a relaxation of the "integral" Spy-game: for any graph G and any $s \geq 2$, $d \geq 0$, $fgn_{s,d}(G) \leq gn_{s,d}(G)$. The fractional variant of the Spy-game has been used to show the first non-trivial lower bound on the guard-number of grids [8]. In this paper, we will give the first exact algorithm using the framework of fractional combinatorial games.

1.1 Our results

We study the Spy-game in the classes of trees and grids. We prove that the guard-number of any tree can be computed in polynomial-time and give a non-trivial upper bound on the fractional guard-number of grids. More precisely, for every $s \geq 2$ and $d \geq 0$:

- We design a Linear Program that computes $fgn_{s,d}(T)$ and a corresponding strategy in polynomial-time for any tree T. Then, we show that any fractional strategy (winning for the guards) using k guards in a tree can be turned into a winning (integral) strategy using $\lfloor k \rfloor$ guards. The key argument is that we can restrict the study to what we call *Spy-positional* strategies. Altogether, this shows that, in any tree T, $fgn_{s,d}(T) = gn_{s,d}(T)$, and that $gn_{s,d}(T)$ and a corresponding winning strategy can be computed in polynomial-time.
- Then, we show that there exists a constant $0 < \alpha \leq \log(3/2)$ such that $fgn_{s,d}(G_{n \times n}) = O(n^{2-\alpha})$. Note that the best known upper bound for $gn_{s,d}(G_{n \times n})$ is $O(n^2)$. A similar bound holds for the $n \times n$ torus.

We believe that the methods using Linear Programming used in this paper are a promising way to better understand other combinatorial games in graphs.

1.2 Related Work

Spy-game. The Spy-game has been defined in [8, 10]. It has been shown that, for every $d \geq 0$ and $s \geq 2$, computing $gn_{s,d}(G)$ is NP-hard in a subclass of chordal graphs (precisely, graphs obtained from a clique and some paths, where one end of each path is connected to some vertices of the clique) [8, 10]. The guard-number of paths is also characterized and almost tight lower and upper bounds are given in the case of cycles. More precisely, $gn_{s,d}(P) = \left\lceil \frac{n}{2d+2+\left\lfloor \frac{2d}{s-1} \right\rfloor} \right\rceil$ for any n-node path P. Moreover, the strategy consists of partitioning the path into $gn_{s,d}(P)$ subpaths with one guard assigned to each one [8, 10]. We show that such a strategy (assigning disjoint subtrees to each guard) is not necessarily

[1] For any graph G, any integer ℓ and $v \in V(G)$, let $N_\ell[v]$ be the set of vertices at distance at most ℓ from v in G and let $N[v] = N_1[v]$.

optimal in trees (see Section 4). Finally, it was proven using a fractional relaxation of the game that there exists $\beta > 0$, such that $\Omega(n^{1+\beta})$ guards are necessary to win in an $n \times n$-grid $G_{n\times n}$ [8]. Precisely, it was proven that $fgn_{s,d}(G_{n\times n}) = \Omega(n^{1+\beta})$ and the result follows from the fact that $fgn_{s,d}(G) \leq gn_{s,d}(G)$ for any graph G. Note that no direct (without using the fractional variant) proof is known. Note also that the best known upper bound in grids is the trivial one: $O(n^2)$. The difficulty of finding the exact value of the guard-number of grids can be related to the difficulty of finding the exact number of cops required to capture a fast robber in a grid [2, 13]. We believe that obtaining results or tools in one case would lead to further progress in the other case.

Eternal Domination. The Spy game generalizes the *Eternal Domination Problem* [16]. In the latter game, a team of mobile agents (cops) occupy some vertices of a graph. At every turn, the second player *attacks* some vertex v and then each of the cops is allowed to move to one of its neighbors or may stay idle such that at least one cop occupies v (note that in the original variant, only one agent was allowed to move at each turn [7]). In other words, the agents must always occupy a dominating set D, such that for any vertex $v \notin D$, the agents can move to another dominating set containing v. The minimum number of agents ensuring to win the game in a graph G is denoted by $\gamma^m(G)$. It is easy to see that the Eternal Domination Problem is equivalent to the Spy game when the spy is arbitrarily fast and $d = 0$, i.e., $\gamma^m(G) = gn_{s,0}(G)$ for any s which is at least the diameter of the graph. Therefore, our results apply to the Eternal Domination Problem.

Eternal Domination has been investigated in several graph classes. In grids, only a few cases are known: for instance, tight bounds are known in $m \times n$ grids for $n \leq 4$ [3, 11] and the case $n = 5$ is considered in [25]. The best known general upper bound in grids is $\left\lceil \frac{nm}{5} \right\rceil + O(n + m)$ [20]. Note that the minimum size of a dominating set in any grid has only been characterized recently [17]. In the class of trees T, $\gamma^m(T)$ can be computed in polynomial-time [19]. The key property in this simple recursive algorithm is that an optimal strategy consists of partitioning a tree into vertex-disjoint stars, each star being assigned to at most 2 cops. As already mentioned, such a method does not extend for the Spy-game.

2 Representation of winning strategies and Spy-positional strategies

In this paper, all graphs are simple (without loops nor multi-edges), connected, and undirected. For any vertex $v \in V$ in a graph $G = (V, E)$, let $N(v)$ denote the set of neighbors of v and $N[v] = N(v) \cup \{v\}$. Moreover, for any integer $s \geq 0$ and vertex $v \in V$, let $N_s[v]$ be the set of vertices at distance at most s from v.

A strategy for the guards is a function describing the moves of the guards at every step. A strategy is *winning* if it allows the guards to perpetually control the spy. It is easy to show that there is always an optimal winning strategy (using the minimum number of guards) which is *positional*, i.e. such that the next move is only determined by the current position of both the spy and the guards, and not by the history of the game[2]. In other words, there is always an optimal winning strategy which is a function that takes the current positions of the spy and of the guards and returns the new positions of the guards (and so, their moves).

[2] That can be easily shown by considering the configurations' graph of the game.

Representation of (fractional) guards' strategies. Let $G = (V, E)$ be an n-node graph, $s \geq 2$ and $d \geq 0$ be two integers. Let $V = \{v_1, \cdots, v_n\}$. A winning strategy σ using $k \in \mathbb{R}^+$ guards is defined as a set $\sigma = \{\mathcal{C}_v\}_{v \in V}$ of sets of configurations. That is, for any $v \in V$ (a possible position for the spy), $\mathcal{C}_v$ is a non-empty set of functions, called *configurations*, that represent the possible positions of the guards when the spy is at v. More precisely, any $\omega \in \mathcal{C}_v$ is a function $\omega : V \to \mathbb{R}^+$, where $\omega(u) \in \mathbb{R}^+$ represents the amount of guards at vertex $u \in V$ when the spy occupies v, that must satisfy $\sum_{u \in V} \omega(u) = k$ and $\sum_{u \in N_d[v]} \omega(u) \geq 1$. Finally, for any $v \in V$, any $\omega \in \mathcal{C}_v$, and any $v' \in N_s[v]$, there must exist $\omega' \in \mathcal{C}_{v'}$ such that the guards can go from ω to ω' in one step. That is, for any possible move of the spy (from v to v'), there must exist a valid flow from ω to ω' (the guards must be able to reach a configuration controlling the spy in v'). A strategy is *integral* if $k \in \mathbb{N}^+$, every of its configurations is a function $V \to \mathbb{N}$, and every move is an integral flow. The *size* of a strategy is the number of different configurations necessary to describe the strategy, i.e., the size of σ is $\sum_{v \in V} |\mathcal{C}_v|$. Note that, a same position for the spy may correspond to different positions of the guards. Therefore, the size of an integral strategy using k guards in an n-node graph is $n^{O(k)}$. Moreover, the size of a fractional strategy is *a priori* unbounded.

Spy-positional strategies. In this paper, we will also consider more constrained strategies. A winning strategy is said to be *Spy-positional* if it depends only on the position of the spy. That is, in a spy-positional strategy $\sigma = \{\mathcal{C}_v\}_{v \in V}$, the positions of the guards are only determined by the position of the spy. In particular, every time the spy occupies some vertex v, the set of vertices occupied by the guards is defined by a unique function $\sigma_v : V(G) \to \mathbb{N}$ such that, for every $u \in V$, $\sigma_v(u)$ is the number of guards occupying u when the spy is occupying v. That is, $\mathcal{C}_v = \{\sigma_v\}$ and $|\mathcal{C}_v| = 1$ for every $v \in V$. An important consequence for our purpose is that any (fractional or integral) spy-positional strategy has size $O(n)$.

Let us remark that, in a spy-positional strategy, it is not required that the same guards occupy the same vertices when the spy is at some vertex. That is, assume that, at some step, the spy occupies some vertex v, some Guard A occupies a vertex a and a guard B occupies a vertex b. It may happen that, after some steps, the spy goes back to v and now Guard A is at b and Guard B is at a (however, the set of vertices occupied by the guards is the same).

Second, there does not always exist an optimal strategy (using the minimum number of guards) that is spy-positional. As an example, consider the cycle C_5 with 5 vertices $\{a, b, c, d, e\}$. It is easy to show that $gn_{2,1}(C_5) = 1$ but that every spy-positional strategy needs 2 guards. One of our main results is to show that, in trees, there always exists an optimal strategy which is spy-positional.

Let $fgn^*_{s,d}(G)$ be the minimum total amount of fractional guards needed to always control at distance d a spy with speed s in a graph G, when the guards are constrained to play spy-positional strategies. By definition, for any graph G and any $s \geq 2$, $d \geq 0$,

$$fgn_{s,d}(G) \leq \min\{fgn^*_{s,d}(G), gn_{s,d}(G)\}.$$

3 Spy-positional fractional strategies in general graphs

This section is devoted to present a polynomial-time algorithm that computes optimal spy-positional fractional strategies in general graphs. Here, optimal means using the minimum total amount of guards with the extra constraint that guards are restricted to play spy-positional strategies. In other words, we prove that, for any graph G, $s \geq 2$, and $d \geq 0$, $fgn^*_{s,d}(G)$ and a corresponding strategy can be computed in polynomial time.

We prove this result by describing a Linear Program with polynomial size that computes such strategies. In Section 4, we will show that in any tree T, $gn_{s,d}(T) = fgn^*_{s,d}(T)$. More precisely, we will show that in trees, the Linear Program below can be used to compute optimal (integral) strategies in polynomial time.

We describe a Linear Program for computing an optimal fractional spy-positional strategy.

Variables. Let $G = (V, E)$ be a connected n-node graph. Recall that a spy-positional strategy is defined by, for each position of the spy, the amount of guards that must occupy each vertex. Therefore, for any two vertices $u, v \in V$, let $\sigma_v(u) \in \mathbb{R}^+$ be the non negative real variable representing the amount of guards occupying vertex u when the spy is at v.

Moreover, for any $x \in V$, $y \in N_s[x]$ and for any $u \in V$ and $v \in N[u]$, let $f_{x,y,u,v} \in \mathbb{R}^+$ be the non negative real variable representing the amount of guards going from vertex u to $v \in N[u]$ when the spy goes from x to $y \in N_s[x]$. Finally, a variable k will represent the total amount of guards. Overall, there are $O((|E|+1)n^2) = O(n^4)$ real variables.

These variables fully describe a strategy, since σ encodes a distribution of cops for every position of the spy and f describes a feasible transition between two successive distributions.

Objective function. We aim at minimizing the total amount of guards.

$$\texttt{Minimize } k. \tag{1}$$

Constraints. The first family of constraints states that, for every position $v \in V$ of the spy, the total amount of guards is at most k.

$$\forall v \in V, \quad \sum_{u \in V} \sigma_v(u) \leq k. \tag{2}$$

The second family of constraints states that, for every position $v \in V$ of the spy, the amount of guards at distance at most d from the spy is at least 1, i.e., the guards always control the spy at distance d.

$$\forall v \in V, \quad \sum_{u \in N_d[v]} \sigma_v(u) \geq 1. \tag{3}$$

The third family of constraints states that, for any move of the spy (from x to $y \in N_s[x]$), the corresponding moves of the guards ensure that the amount of guards leaving a vertex $v \in V$ plus what remains at v equals the amount of guards that was at v before the move.

$$\forall x \in V\,,\;\; y \in N_s[x]\,,\;\; v \in V\,,\quad \sum_{w \in N[v]} f_{x,y,v,w} = \sigma_x(v). \tag{4}$$

The fourth family of constraints states that, for any move of the spy (from x to $y \in N_s[x]$), the corresponding moves of the guards ensure that the amount of guards that are at a vertex $w \in V$ after the moves equals the amount of guards arriving in w plus what remains at w.

$$\forall x \in V\,,\;\; y \in N_s[x]\,,\;\; w \in V\,,\quad \sum_{v \in N[w]} f_{x,y,v,w} = \sigma_y(w). \tag{5}$$

There are $O(n^4)$ constraints and the above Linear Program has polynomial size and clearly computes an optimal spy-positional fractional strategy. Hence:

▶ **Theorem 1.** *For any connected graph G, and any two integers $s \geq 2$ and $d \geq 0$, the above Linear Program computes $fgn^*_{s,d}(G)$ and a corresponding spy-positional strategy in polynomial time.*

4 Spy game is Polynomial in Trees

This section is devoted to the study of the Spy-game in trees (Theorem 4). Before going into the details, we would like to emphasize one difficulty when dealing with guards' strategies.

A natural idea would be to partition the tree into smaller subtrees (with bounded diameter) with a constant number of guards assigned to each of them. That is, each guard would be assigned (possibly with other guards) a subtree S and would move only when the spy is in S (in particular, the guard would only occupy some vertices of S). As already mentioned, there exist such strategies that are optimal in paths [8, 10] or in trees when $d = 0$ and s is large (Eternal Domination) [19]. We cannot expect such strategies for the Spy-game (for any $s \geq 2$ and $d > 0$) in trees. Precisely, we can define a family of trees with unbounded guard-number such that, for each of these trees, there is a strategy of the spy that forces every guard to occupy every non-leaf vertex infinitely often (see [9]). Hence, optimal guards' strategies seem difficult to be described in trees.

To overcome this difficulty, we use the power of Linear Programming. Precisely, we prove that, in any tree T and for any $s \geq 2$, $d \geq 0$, $gn_{s,d}(T) = fgn^*_{s,d}(T)$. Therefore, using the Linear Program of Section 3, it follows that computing $gn_{s,d}(T)$ can be done in polynomial time in trees. The proof is twofold. First, we prove that $gn_{s,d}(T) = fgn_{s,d}(T)$ for any $s \geq 2$ and $d \geq 0$ (i.e., the integrality gap is null in trees), and then that $fgn_{s,d}(T) = fgn^*_{s,d}(T)$.

▶ **Theorem 2.** *For any tree T and for any $s \geq 2$, $d \geq 0$, $gn_{s,d}(T) = fgn_{s,d}(T)$. More precisely, any fractional winning strategy using a total amount of $k \in \mathbb{R}^+$ guards can be transformed into an integral winning strategy using $\lfloor k \rfloor$ guards. Moreover, such a transformation can be done in polynomial time in the size of the fractional strategy.*

Proof. Let $\sigma = \{\mathcal{C}_v\}_{v \in V}$ be any fractional winning strategy using a total amount of $k \in \mathbb{R}^+$ guards to control a spy with speed $s \geq 2$, at distance $d \geq 0$, and in an n-node tree $T = (V, E)$.

We build a winning integral strategy σ^r using $\lfloor k \rfloor$ guards by "rounding" all configurations of σ. For any configuration ω of σ, we will define an integral configuration ω^r (which we call a *rounding* of ω) using $\lfloor k \rfloor$ guards (Claim 1), such that if the spy is controlled in ω then it is also controlled in ω^r (Claim 2). Moreover, for any two configurations ω_1 and ω_2 such that there is a feasible flow from ω_1 to ω_2, we show that there is feasible integral flow from ω_1^r to ω_2^r (Claim 3). Altogether, this shows that σ^r is a winning integral strategy using $\lfloor k \rfloor$ guards, which proves the theorem. The omitted proofs of the claims below can be found in [9].

From now on, let us consider T to be rooted at some vertex $r \in V$.

Notations. For any $u \in V$, let T_u be the subtree of T rooted in u (i.e., the subtree that consists of u and all its descendants). For any configuration $\omega : V \to \mathbb{R}^+$, let $\omega(T_u) = \sum_{v \in V(T_u)} \omega(v)$ and let $\omega(T) = \omega(T_r)$. By definition, $\omega(T_u) \geq \omega(u)$ for every $u \in V$. Finally, let $cont(T, \omega) = \{u \in V : \sum_{v \in N_d[u]} \omega(v) \geq 1\}$ (i.e., $cont(T, \omega)$ is the set of vertices u such that the spy on u is controlled at distance d by the guards in the configuration ω).

Let us define the *rounded* configuration $\omega^r : V \mapsto \mathbb{N}$ as, for every $u \in V$,

$$\omega^r(u) = \left\lfloor \omega(u) + \sum_{v \text{ child of } u} (\omega(T_v) - \lfloor \omega(T_v) \rfloor) \right\rfloor$$

Intuitively, the fractional part of guards that are in each of the subtrees rooted in the children of u is "pushed" to u. Then u "keeps" only the integral part of the sum of what it had plus what it received from its children.

We first prove that rounding a configuration using k guards provides an integral configuration using $\lfloor k \rfloor$ guards. A simple induction on n shows that:

▶ **Claim 1.** *For any configuration $\omega : V(T) \to \mathbb{R}^+$, $\omega^r(T) = \lfloor \omega(T) \rfloor$*

Then, Claim 2 proves that every position of the spy that is controlled by the guards in a configuration ω is also controlled by the guards in the configuration ω^r. We prove that, for every $v \in V$ and $\ell \in \mathbb{N}$, $\sum_{x \in N_\ell[v]} \omega^r(x) \geq \lfloor \sum_{x \in N_\ell[v]} \omega(x) \rfloor$. Hence, if $u \in cont(T, \omega)$, i.e., $1 \leq \sum_{x \in N_d[u]} \omega(x)$, then $1 \leq \lfloor \sum_{x \in N_d[u]} \omega(x) \rfloor \leq \sum_{x \in N_d[u]} \omega^r(x)$ and so $u \in cont(T, \omega^r)$.

▶ **Claim 2.** *For any configuration $\omega : V(T) \to \mathbb{R}^+$, $cont(T, \omega) \subseteq cont(T, \omega^r)$*

Claim 3 shows that the moves that were valid in σ still hold in the "rounded" strategy. Its proof can be found in [9].

▶ **Claim 3.** *Let $\omega_1, \omega_2 : V(T) \mapsto \mathbb{R}^+$ be two configurations such that the guards can go from ω_1 to ω_2 in one step (there is feasible flow from ω_1 to ω_2). Then, the guards can go from ω_1^r to ω_2^r in one step (there is feasible integral flow from ω_1^r to ω_2^r).*

This concludes the proof of Theorem 2. ◀

The second step in this section is to show that there is always an optimal fractional strategy which is spy-positional. For this purpose, we prove the following theorem.

▶ **Theorem 3.** *For any tree T and for any $s \geq 2$, $d \geq 0$, $fgn^*_{s,d}(T) = fgn_{s,d}(T)$. More precisely, any fractional winning strategy using a total amount of $k \in \mathbb{R}^+$ guards can be transformed into a spy-positional winning strategy using k guards.*

Proof. Let $\sigma = \{\mathcal{C}_v\}_{v \in V}$ be any fractional winning strategy using a total amount of $k \in \mathbb{R}^+$ guards to control a spy with speed $s \geq 2$, at distance $d \geq 0$, and in an n-node tree $T = (V, E)$. Recall that, for any vertex $v \in V$, $\mathcal{C}_v$ is the set of possible configurations $\omega : V \to \mathbb{R}^+$ for the guards when the spy is at v.

The proof consists in defining a spy-positional strategy σ^{min} that is a winning strategy using k guards. For any $v \in V$, we will define the function $\omega_v^{min} : V \to \mathbb{R}^+$ to be the (unique) configuration of σ^{min} when the spy is at v, i.e., $\sigma^{min} = \{\omega_v^{min}\}_{v \in V}$. We first prove that σ^{min} is a strategy using k guards (Claims 4-5), then that the spy at $v \in V$ is controlled at distance d by the guards in the configuration ω_v^{min} (Claim 6). Finally, we prove that, for any move of the spy from v to $v' \in V$, the guards can move from ω_v^{min} to $\omega_{v'}^{min}$ (Claim 7).

From now on, T is rooted in an arbitrary vertex $r \in V$.

Notations. For any weight function $\omega : V \to \mathbb{R}^+$, let $\omega^+ : V \to \mathbb{R}^+$ be the *cumulative* function of ω, defined by, for every $u \in V$, $\omega^+(u) = \sum_{v \in V(T_u)} \omega(v) = \omega(T_u)$. Let $v \in V$ and $\mathcal{C}_v = \{\omega_1, \cdots, \omega_h\} \in \sigma$ be the set of configurations of the guards, when the spy is in v. Let $\alpha_v : V \to \mathbb{R}^+$ be such that, for every $u \in V$, $\alpha_v(u) = \min_{1 \leq i \leq h} \omega_i^+(u)$. Now, ω_v^{min} is defined as the (unique) function such that α_v is its cumulative function, i.e., $\alpha_v = (\omega_v^{min})^+$. Formally, for every $u \in V$: $\omega_v^{min}(u) = \alpha_v(u) - \sum_{x \text{ child of } u} \alpha_v(x)$.

Claim 4 proves that, for every $v \in V$, $\omega_v^{min} : V \to \mathbb{R}^+$ is a configuration.

▶ **Claim 4.** *For every $u \in V$, $\omega_v^{min}(u) \geq 0$.*

Proof of Claim 4. Let $1 \leq i \leq h$ be an integer such that $\alpha_v(u) = \min_{1 \leq j \leq h} \omega_j^+(u) = \omega_i^+(u)$. By definition of α_v, for every $x \in Children(u)$, $\alpha_v(x) = \min_{1 \leq j \leq h} \omega_j^+(x) \geq \omega_i^+(x)$. Hence, $\omega_v^{min}(u) \geq \omega_i^+(u) - \sum_{x \in Children(u)} \omega_i^+(x) = \omega_i(u) \geq 0$. ◀

Claim 5 proves that, for every $v \in V$, the configuration ω_v^{min} uses k guards.

▶ **Claim 5.** *For every $v \in V$, $\sum_{u \in V} \omega_v^{min}(u) = k$.*

Proof of Claim 5. For every $1 \leq i \leq h$, $\omega_i^+(r) = k$. Hence, $\alpha_v(r) = \min_{1 \leq i \leq h} \omega_i^+(r) = k$. $\sum_{u \in V} \omega_v^{min}(u) = (\omega_v^{min})^+(r) = \alpha_v(r) = k$ (since α_v is the cumulative function of ω_v^{min}). ◀

Claim 6 proves that the guards in the configuration ω_v^{min} control a spy located at v. Finally, Claim 7 shows that the moves that were valid in σ still hold for σ^{min} (see [9]).

▶ **Claim 6.** *For every $v \in V$, $\sum_{u \in N_d[v]} \omega_v^{min}(u) \geq 1$.*

▶ **Claim 7.** *For every $v \in V$ and $v' \in N_s[v]$, there is a feasible flow from ω_v^{min} to $\omega_{v'}^{min}$.*

This concludes the proof of Theorem 3. ◀

▶ **Theorem 4.** *Let $s \geq 2$ and $d \geq 0$ be two integers. There is a polynomial-time algorithm that computes an integral winning strategy using $gn_{s,d}(T)$ guards to control a spy with speed s at distance d in any tree T.*

Proof. By Theorem 3, there exists an optimal (fractional) winning strategy that is spy-positional. By Theorem 1, such a strategy can be computed in polynomial time. By Theorem 2, an optimal integral winning strategy can be computed in polynomial time from any optimal fractional winning strategy. ◀

5 Fractional Spy-game in Grid and Torus

In this section, we provide some progress toward the understanding of the Spy-game in grids. Precisely, we provide the first fractional strategy using a sub-linear (in the number of vertices) number of guards. It is clear that, for any $n \times n$ grid $G_{n \times n}$, $gn_{s,d}(G_{n \times n}) = O(n^2)$, and it is known that $fgn_{s,d}(G_{n \times n})$ is super-linear in n [8]. However, the exact order of magnitude of $gn_{s,d}(G_{n \times n})$ (and of $fgn_{s,d}(G_{n \times n})$) is not known. We prove that $fgn_{s,d}(G_{n \times n})$ is sub-quadratic in n.

Let $n, m \geq 2$ be two integers. We consider the $n \times m$ toroidal grid $TG_{n \times m} = (V, E)$, i.e., the graph with vertices $v_{i,j} = (i, j)$ and edges $\{(i,j), (i+1 \mod n, j)\}$ and $\{(i,j), (i, j+1 \mod m)\}$, for all $0 \leq i < n$ and $0 \leq j < m$. The $n \times m$ grid $G_{n \times m}$ is obtained from $TG_{n \times m}$ by removing the edges $\{\{(i, m-1), (i, 0)\}; \{(n-1, j), (0, j)\} \mid \forall 0 \leq i < n, 0 \leq j < m\}$.

First, we show that the number of fractional (resp., integral) guards required in the grid and in the torus have the same order of magnitude. Informally, the proof consists in considering a strategy $\mathcal{S}$ in a grid (resp., in a torus) and in applying in the torus (resp., in the grid) four symmetric strategies, each one mimicking $\mathcal{S}$ (see [9]).

▶ **Lemma 5.** *For every $n, m \geq 2$, $s \geq 2$, $d \geq 0$, and for every $f \in \{gn_{s,d}, fgn_{s,d}, fgn^*_{s,d}\}$:*

$$f(TG_{n \times m})/4 \leq f(G_{n \times m}) \quad \leq 4 \cdot f(TG_{n \times m}).$$

The remaining part of this section is devoted to prove the following theorem.

▶ **Theorem 6.** *There exists $0 < \alpha \leq \log(3/2) \approx 0.58$ such that, for every $s \geq 2$, $d \geq 0$,*

$$fgn^*_{s,d}(TG_{n \times n}) = O(n^{2-\alpha}).$$

To prove Theorem 6, we make use of the Linear Program (LP) of Section 3. Recall that, in a spy-positional strategy, the positions of the guards (configuration) only depend on the position of the spy. In any vertex-transitive graph (so in $TG_{n \times n}$), there is actually a unique configuration to be considered (where the spy is occupying the vertex $(0,0)$). Therefore, the LP of Section 3 can be reformulated as follows.

We are looking for a function $\omega : \{0, \cdots, n-1\}^2 \to \mathbb{R}^+$ such that $\omega(i,j)$ is the amount of guards occupying the vertex (i,j) when the spy is occupying the vertex $(0,0)$. This function must be defined such as to minimize the number of guards, i.e., $\sum_{0 \leq i,j < n} \omega(i,j)$ must be minimum, subject to the following constraints. The spy must be controlled, i.e., $\sum_{(i,j) \in N_d[(0,0)]} \omega(i,j) \geq 1$. Moreover, for any move of the spy from $(0,0)$ to $(x,y) \in N_s[(0,0)]$, there must be a feasible flow from the configuration $(\omega(i,j))_{(i,j) \in V(TG_{n\times n})}$ to $(\omega(i-x, j-y))_{(i,j) \in V(TG_{n\times n})}$. Before going further, let us simplify the latter constraint. Indeed, instead of considering every possible move of the spy in $N_s[(0,0)]$, we only consider the *extremal moves* from $(0,0)$ to one of the vertices in $\{(0,s), (s,0), (-s,0), (0,-s)\}$, i.e., we weaken the spy by allowing it to move only "horizontally" or "vertically" at full speed. We prove (see [9]) that it does not change the order of magnitude of an optimal solution.

The above LP, restricted to vertex-transitive graphs, is more efficient than the one presented in Section 3 since there is only one configuration to be considered and less flow constraints (and so, much less variables and constraints). In particular, it gives interesting experimental results as presented in the conclusion. In what follows, we present and analyze a function using a sub-quadratic (in n) number of guards that satisfies the above LP.

Precisely, let $0 < \alpha < 1$ and let $d(v)$ (resp., $d(i,j)$) denote the distance between vertex v (resp., (i,j)) and vertex $(0,0)$ in $TG_{n\times n}$.

▶ **Definition 7** (Strategies ω_α)**.** Let us consider the spy-positional strategy ω_α of the form $\omega_\alpha(i,j) = \frac{B}{(d(i,j)+1)^\alpha}$ for every $(i,j) \in V(TG_{n\times n})$ and for some constant B defined later.

Note that ω_α is *symmetric*, i.e., $\omega_\alpha(i,j) = \omega_\alpha(n-i,j) = \omega_\alpha(i, n-j \mod n) = \omega_\alpha(n-i, n-j \mod n)$. Therefore, by symmetry, we only need to check that there is a feasible flow from the configuration $(\omega_\alpha(i,j))_{(i,j) \in V(TG_{n\times n})}$ to the one $(\omega_\alpha(i-s,j))_{(i,j) \in V(TG_{n\times n})}$, i.e., when the spy goes from $(0,0)$ to $(s,0)$.

Equivalently, the flow constraints can be defined as a flow problem in a *transportation bipartite auxiliary network* H defined as follows (i.e., the constraints are satisfied if and only if there is feasible flow in H). Let $H = (V_1 \cup V_2, E(H))$ be the graph such that V_1 and V_2 are two copies of $V(TG_{n\times n})$. There is an arc from $u \in V_1$ to $v \in V_2$ if $\{u,v\} \in E(TG_{n\times n})$. Each vertex $(i,j) \in V_1$ has a supply $\omega_\alpha(i,j)$ and every vertex $(i',j') \in V_2$ has a demand $\omega_\alpha(i-s,j)$. By Hall's Theorem [6], there is a feasible flow in H if and only if, for every $A \subseteq V_1$, the total supply in $N[A]$ is at least the demand in $A \subseteq V_2$, i.e., at least $\sum_{(i,j) \in A} \omega_\alpha(i-s,j)$.

To summarize, the flow constraints can be stated as:

$$\forall A \subseteq V(TG_{n\times n}), \sum_{(i,j) \in N[A]} \omega_\alpha(i,j) \geq \sum_{(i,j) \in A} \omega_\alpha(i-s,j). \tag{6}$$

We aim at deciding the range of α such that the function ω_α satisfies constraint 6. For this purpose, we first aim at finding a set $\mathcal{H}_s \subseteq V(TG_{n\times n})$ such that $\kappa_\alpha(\mathcal{H}_s) = \sum_{(i,j) \in N[\mathcal{H}_s]} \omega_\alpha(i,j) - \sum_{(i,j) \in \mathcal{H}_s} \omega_\alpha(i-s,j)$ is minimum. For such a set $\mathcal{H}_s$, if $\kappa_\alpha(\mathcal{H}_s) \geq 0$, it implies that ω_α satisfies constraint 6.

Let $\mathcal{H}_s$ be the set of vertices $(i,j) \in V(TG_{n\times n})$ defined by:

$$\mathcal{H}_{\mathbf{s}} = \{(i,j) \mid s/2 \leq i \leq (n+s)/2 \mod n,\ 0 \leq j < n\}.$$

The proof of the following technical lemma is available in [9].

▶ **Lemma 8.** *Let $\alpha > 0$ and $s \leq n/2$. For every $A \subseteq V(TG_{n\times n})$, $\kappa_\alpha(A) \geq \kappa_\alpha(\mathcal{H}_s)$.*

Finally, we are ready to present a winning strategy in the $n \times n$ torus which proves Th. 6.

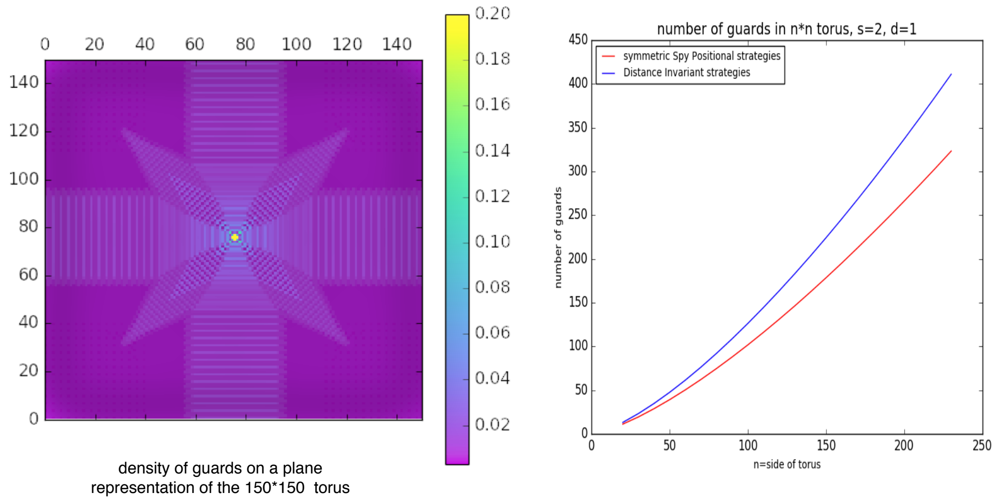


Figure 1 Experimental results, $s = 2$ and $d = 1$. (Left) Density of guards on a plane representation of the 150*150 torus in an optimal symmetrical Spy-positional configuration. (Right) Minimum number of guards for symmetrical (red) and distance-invariant (blue) Spy-Positional strategies.

▶ **Lemma 9.** *Let $n, s \geq 2$, $s \leq n/2$, $d \geq 0$ and $0 < \alpha \leq \log(3/2)$. There exists a constant $B > 0$ (independent of n) such that the function $\omega_\alpha : V(TG_{n\times n}) \to \mathbb{R}^+$ where $\omega_\alpha(v) = \frac{B}{(d(v)+1)^\alpha}$ for every $v \in V(TG_{n\times n})$ is a spy-positional winning fractional strategy that uses $O(n^{2-\alpha})$ guards to control a spy with speed s at distance d in $TG_{n\times n}$.*

Sketch of Proof. See [9] for full proof. To verify that ω_α is a winning strategy, we need to prove that it satisfies constraints 3 and 6. Let B_d be the set of vertices at distance at most d from $(0, 0)$ and let $B = 1/\sum_{v\in B_d} \frac{1}{(d(v)+1)^\alpha}$. Constraint 3 is satisfied by the choice of B. Some computations allow us to show that $\kappa_\alpha(\mathcal{H}_s) \geq 0$ if $0 < \alpha \leq \log(3/2)$ and therefore, by Lemma 8, Constraint 6 is satisfied. Finally, a simple summation shows that the strategy uses $\sum_{v\in V(TG_{n\times n})} \omega_\alpha(v) = O(n^{2-\alpha})$ guards. ◀

6 Conclusion

Concerning the Spy-game, the main open question is to determine the exact value of $gn_{s,d}(G_{n\times n})$ in any $n \times n$ grid $G_{n\times n}$ (or torus). A first step towards such a result would be to prove that $gn_{s,d}(G_{n\times n}) = O(gn_{s',d'}(G_{n\times n}))$ for any $s, s' \geq 2$ and $d, d' \geq 0$. To get more intuition on optimal strategies for guards, we used Cplex to solve the LP described in Section 3 with additional constraints of symmetry. The left drawing in Fig. 1 represents the density of guards in the torus of side 150 (where the central vertex is the position of the spy) for $s = 2$ and $d = 1$. It shows that optimal symmetric Spy-positional (SSP) strategies may be much more intricate than the strategy ω_α we studied. For instance, it is not monotone when the distance to the spy's position increases. On the right, we plotted the number of guards used by optimal SSP (in red) which is much less than $n^{2-\log(3/2)}$ for

$n \leq 250$ (it is difficult to extrapolate further intuition from such small values of n)[3]. Even the optimal distance-invariant strategies (i.e., the density of guards is only a function of the distance to the spy's position) computed using the LP (plotted in blue) use much less guards than $n^{2-\log(3/2)}$ (we did not plot the function $n^{2-\log(3/2)}$ for more readability, indeed, $50^{2-\log(3/2)} > 500$ and $250^{2-\log(3/2)} > 6600$). In trees, it would be interesting to design a combinatorial algorithm (i.e., not relying on the solution of a Linear Program) that computes optimal strategies for controlling a spy with speed s at distance d.

More importantly, using the fractional framework to obtain new results in two-player combinatorial games in graphs seems promising.

References

1 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8:1–12, 1984.

2 P. Balister, S. Binski, B. Bollobás, and B. P. Narayanan. Catching a fast robber on the grid. *CoRR*, abs/1609.01002, 2016. URL: http://arxiv.org/abs/1609.01002.

3 I. Beaton, S. Finbow, and J.A. MacDonald. Eternal domination numbers of 4 × n grid graphs. *J. Comb. Math. Comb. Comput.*, 85:33–48, 2013.

4 A. Bonato, E. Chiniforooshan, and P. Pralat. Cops and robbers from a distance. *Theor. Comput. Sci.*, 411(43):3834–3844, 2010.

5 A. Bonato and R. J. Nowakowski. *The game of Cops and Robbers on Graphs.* American Math. Soc., 2011.

6 J. A. Bondy and U. S. R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics.* Springer, 2008.

7 A. Burger, E. J. Cockayne, W. R. Gründlingh, C. M. Mynhardt, J. H. van Vuuren, and W. Winterbach. Infinite order domination in graphs. *J. Comb. Math. Comb. Comput.*, 50:179–194, 2004.

8 N. Cohen, M. Hilaire, N. A. Martins, N. Nisse, and S. Pérennes. Spy-game on graphs. In *8th International Conference on Fun with Algorithms, FUN 2016*, pages 10:1–10:16, 2016.

9 N. Cohen, F. Mc Inerney, N. Nisse, and S. Pérennes. Study of a combinatorial game in graphs through linear programming. Technical report, INRIA, 2017. URL: https://hal.archives-ouvertes.fr/hal-01462890.

10 N. Cohen, N. A. Martins, F. Mc Inerney, N. Nisse, S. Pérennes, and R. Sampaio. Spy-game on graphs: complexity and simple topologies. Technical report, INRIA, 2017. URL: https://hal.archives-ouvertes.fr/hal-01463297.

11 A. Z. Delaney and M. E. Messinger. Closing the gap: Eternal domination on 3 × n grids. *to appear in Contributions to Discrete Mathematics*, 2015.

12 F. V. Fomin, F. Giroire, A. Jean-Marie, D. Mazauric, and N. Nisse. To satisfy impatient web surfers is hard. In *6th Int. Conf. on Fun with Algorithms (FUN)*, volume 7288 of *LNCS*, pages 166–176, 2012.

13 F. V. Fomin, P. A. Golovach, J. Kratochvíl, N. Nisse, and K. Suchan. Pursuing a fast robber on a graph. *Theor. Comput. Sci.*, 411(7-9):1167–1181, 2010.

14 F. Giroire, D. Mazauric, N. Nisse, S. Pérennes, and R. P. Soares. Connected surveillance game. In *20th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Lecture Notes in Computer Science. Springer, 2013.

15 F. Giroire, N. Nisse, S. Pérennes, and R. P. Soares. Fractional combinatorial games. Technical report, INRIA, 2013. RR8371. URL: http://hal.inria.fr/hal-00865345.

[3] Solving the LP for $n \geq 150$ takes more than one hour on a basic laptop

16 W. Goddard, S. M. Hedetniemi, and S. T. Hedetniemi. Eternal security in graphs. *J. Comb. Math. Comb. Comput.*, 52:160–180, 2005.

17 D. Gonçalves, A. Pinlou, M. Rao, and S. Thomassé. The domination number of grids. *SIAM J. Discrete Math.*, 25(3):1443–1453, 2011.

18 W. B. Kinnersley. Cops and robbers is exptime-complete. *JCTB*, 111:201–220, 2015.

19 W. F. Klostermeyer and G. MacGillivray. Eternal dominating sets in graphs. *J. Comb. Math. Comb. Comput.*, 68, 2009.

20 I. Lamprou, R. Martin, and S. Schewe. Perpetually dominating large grids. *CoRR*, abs/1611.08204, 2016. URL: `http://arxiv.org/abs/1611.08204`.

21 R. J. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Maths*, 43:235–239, 1983.

22 A. Quilliot. *Problèmes de jeux, de point fixe, de connectivité et de représentation sur des graphes, des ensembles ordonnés et des hypergraphes.* Doctorat d'état, Univ. Paris 4, 1983.

23 B. S. W. Schröder. The copnumber of a graph is bounded by $\lfloor \frac{3}{2} genus(g) \rfloor + 3$. *Categorical perspectives (Kent, OH, 1998), Trends in Mathematics*, pages 243–263, 2001.

24 A. Scott and B. Sudakov. A bound for the cops and robbers problem. *SIAM J. Discrete Math.*, 25(3):1438–1442, 2011.

25 C. M. van Bommel and M. F. van Bommel. Eternal domination numbers of $5 \times n$ grid graphs. *J. Comb. Math. Comb. Comput.*, 97:83–102, 2016.

On Maximal Cliques with Connectivity Constraints in Directed Graphs[*]

Alessio Conte[1], Mamadou Moustapha Kanté[†2], Takeaki Uno[3], and Kunihiro Wasa[4]

1 **Università di Pisa, Pisa, Italy**
`conte@di.unipi.it`
2 **Université Clermont Auvergne, LIMOS, CNRS, Aubière, France**
`mamadou.kante@uca.fr`
3 **National Institute of Informatics, Tokyo, Japan**
`uno@nii.ac.jp`
4 **National Institute of Informatics, Tokyo, Japan**
`wasa@nii.ac.jp`

Abstract

Finding communities in the form of cohesive subgraphs is a fundamental problem in network analysis. In domains that model networks as undirected graphs, communities are generally associated with dense subgraphs, and many community models have been proposed. Maximal cliques are arguably the most widely studied among such models, with early works dating back to the '60s, and a continuous stream of research up to the present. In domains that model networks as *directed* graphs, several approaches for community detection have been proposed, but there seems to be no clear model of cohesive subgraph, *i.e.*, of what a community should look like. We extend the fundamental model of clique to directed graphs, adding the natural constraint of strong connectivity within the clique. We characterize the problem by giving a tight bound for the number of such cliques in a graph, and highlighting useful structural properties. We then exploit these properties to produce the first algorithm with polynomial delay for enumerating maximal strongly connected cliques.

1998 ACM Subject Classification G.2.2 Graph Theory, Graph algorithms

Keywords and phrases Enumeration algorithms, Bounded delay, Directed graphs, Community structure, Network analytics

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.23

1 Introduction

The problem of community detection in graphs has been extensively studied. In undirected graphs, dense subgraphs are often used to detect communities, with applications in areas such as social network analysis [25, 28], biology [16], and more [13].

Several definitions of dense subgraph have been proposed to model communities [22, 28]. The earliest, and perhaps the most widely studied is that of the maximal clique: interest in the problem of finding maximal cliques started several decades ago [1, 5, 21] and effort to produce efficient algorithms can still be seen in recent works [8, 10, 12].

[*] This work was supported by JST CREST, Grant Number JPMJCR1401, Japan.
[†] M.M. Kanté is supported by French Agency for Research under the GraphEN project (ANR-15-CE-0009).

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 23; pp. 23:1–23:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As for directed graphs, there seems to be consensus in literature [18, 19, 23] on the fact that ignoring edge directions and applying community detection techniques for undirected graphs is not satisfactory. Several ad-hoc techniques for clustering and community discovery have been proposed, mirroring the goals of algorithms for undirected graphs.

Awerbuch *et al.* [3] proposed a bounded-error scheme for aggregating vertices in directed graphs as a hierarchical structure. Leicht *et al.* [19] adapted the concept of *modularity* to account for edge directions, with the aim of extracting more meaningful clusters. The LinkRank algorithm [18] aimed at partitioning a directed graph into communities using random walks and the PageRank algorithm. More approaches can be found in [13].

Subgraph-based community models in undirected networks are thoroughly studied in community detection (and network analysis in general), thus it would be natural to imagine that similar models were object of study in the directed area. Surprisingly, this seems to be a road less traveled.[1] Charikar *et al.* [6] considered communities in directed graphs as sets of vertices whose induced subgraphs have many edges, regardless of connectivity. The well-known work by Kleinberg *et al.* [17] defined a community in the web graph with respect to a topic as a special *bipartite clique* $K_{i,j}$, in which each of the i vertices has edges *towards* each of the other j vertices, which represent authority pages on the topic. To the best of our knowledge, there are no other community models for directed graphs that are widely accepted and rigorous. This motivates our interest in combining the basic maximal clique model with connectivity in directed graphs, that is strong connectivity. We call this model a *strongly connected clique* (SCQ for short), and investigate both its properties and the problem of efficiently finding all maximal SCQs.

Generic enumeration techniques for maximal subgraphs have been proposed for *strongly accessible* properties [2], *i.e.*, such that every non-maximal subgraph A which verifies the property is included in a subgraph B of size exactly $|A| + 1$ that also verifies the property. Cohen *et al.* [7] proposed an algorithmic framework for enumerating maximal subgraphs with respect to subsets of strongly accessible properties, namely *hereditary* and *connected-hereditary* graph properties. SCQs, however, fit in neither of these classes.

Finding maximal subgraphs satisfying a non *accessible* property is a challenging task, as their structure is unsystematic, and their enumeration requires new techniques and theoretical insight. In this work, we show that SCQs have a peculiar but rigorous structure, which fits under a relaxed, more general notion of accessibility. We then exploit this structure to design SCQ-ENUM, an efficient algorithm that enumerates SCQs with *delay* bounded by $O(\min(\omega(G)d^2\Delta^2, m^2))$, where $\omega(G)$, d, Δ and m are respectively the largest size of an SCQ, degeneracy, maximum degree and number of edges of the input graph, and the delay is the maximum time elapsed between two consecutive outputs. The value of SCQ-ENUM is two-fold: on one hand, it constitutes a first step towards the characterization, and potentially towards general enumeration techniques, for a wider range of problems that are not accessible. On the other hand, SCQ-ENUM is also an efficient practical tool for discovering community structures in directed networks. Finally, we complete the analysis of the model by giving a tight bound for the number of maximal SCQs in an n-vertex graph.

[1] It should be mentioned that strongly connected components have been object of thorough study, however these may be very large, sparse, and thus may not be significant indicators of community structures.

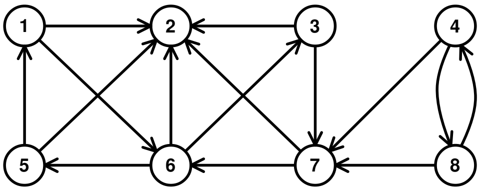


Figure 1 A graph with 4 maximal SCQs ({1,5,6},{2},{3,6,7},{4,8}) whose underlying undirected graph has 3 maximal cliques ({1,2,5,6},{2,3,6,7},{4,7,8}).

2 Preliminaries

We refer to [11] for our graph terminology, and all (directed) graphs considered in this paper are without multi-edges and loops (but may contain edges of opposite direction). The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$. A (directed) edge (or arc) from x to y is denoted by (x, y) in which x is the *tail* and y the *head*; we will say that the edge is *from* x and *towards* y. When $E(G)$ is symmetric, *i.e.*, $(x, y) \in E(G)$ if and only if $(y, x) \in E(G)$, we call G *undirected* and denote each edge (x, y) of G by xy (equivalently yx). For a graph G, we denote by $\mathsf{u}(G)$, called *underlying (undirected) graph* of G, the undirected graph with vertex set $V(G)$ and edge set $\{xy \mid (x, y) \in E(G) \text{ or } (y, x) \in E(G)\}$. We use n and m to denote the number of vertices and edges, respectively, in any graph.

For a vertex x in a graph G, $N_G(x)$ denotes its set of neighbours, which includes both *in-neighbours*, *i.e.*, $\{y \in V(G) \mid (y, x) \in E(G)\}$, and *out-neighbours*, *i.e.*, $\{y \in V(G) \mid (x, y) \in E(G)\}$. $|N_G(x)|$ denotes the *degree* of x, and $\Delta(G)$ the maximum degree of a vertex in G. Any vertex with degree $|V(G)| - 1$ is called a *universal* vertex. The subgraph of G induced by $X \subseteq V(G)$ is the graph $G[X] = (X, E(G) \cap (X \times X))$. When the graph G is clear from the context we will drop the subscripts from the notations $N_G(x)$ and similar ones, and also write V (or similar notations E, Δ, ...) instead of $V(G)$ (or $E(G)$, $\Delta(G)$, ...).

The power-set of the set V is denoted by 2^V. For two sets of vertices A and B we denote by $A \setminus B$ the set $\{x \in A \mid x \notin B\}$. Given a total ordering on the vertices in V, represented by increasing labels $v_1, \ldots, v_n$, the associated *lexicographic ordering* on 2^V, denoted by $\leq$, is such that $A \leq B$ if A contains the smallest element not in common, *i.e.*, $\min((A \cup B) \setminus (B \cap A)) \in A$.

A *clique* of an undirected graph G is a subset C of $V(G)$ that induces a complete graph, and a *maximal clique* is a clique C of G such that $C \cup \{x\}$ is not a clique for all $x \in V(G) \setminus C$. Let $G = (V, E)$ be a (directed) graph. A *strongly connected clique* (or SCQ for short) is a set $C \subseteq V(G)$ such that $G[C]$ is strongly connected, and $\mathsf{u}(G)[C]$ is a clique of $\mathsf{u}(G)$. We recall that a directed graph (or subgraph) is strongly connected if and only if for each bipartition (V_1, V_2) of V there is an edge from V_2 to V_1 (and symmetrically from V_1 to V_2) [9]. We assume that a single vertex is an SCQ; we denote the maximum size of an SCQ in G by $\omega(G)$, and the maximum size of a clique in $\mathsf{u}(G)$ by $\omega(\mathsf{u}(G))$. It is worth noticing that if G is undirected, SCQs and cliques coincide. An SCQ C is maximal if there is no SCQ C' such that $C \subset C'$. Given $C \subseteq V$, the set $X \subseteq V \setminus C$ is *addible* to C if $C \cup X$ is an SCQ, and $Y \subseteq C$ is *removable* from C if $C \setminus Y$ is an SCQ. Furthermore, we say that a vertex x is a *sink w.r.t.* C if there is no $(x, y) \in E(G)$ with $y \in C$, and a *source w.r.t.* C if there is no $(y, x) \in E(G)$ with $y \in C$. A graph with its maximal SCQs and cliques is shown in Figure 1.

A graph G is *d-degenerate* if each induced subgraph of G has a vertex of degree at most d. The *degeneracy* of a graph G is the minimum d such that G is d-degenerate. A *degeneracy ordering* of a graph G of degeneracy d is a sequence $v_1, v_2, \ldots, v_n$ of its vertex set such that the degree of each v_i in $G[\{v_i, \dots, v_n\}]$ is at most d; we call $N(v_i) \cap \{v_{i+1}, \dots, v_n\}$ the *forward neighbours* of v_i. We assume that any graph is given with a degeneracy ordering[2].

3 Problem Characterization

Maximal SCQs are a challenging problem as they do not satisfy the *strong accessibility* property. However, we provide some related properties that will be the key of our enumeration algorithm.

3.1 Relaxed Accessibility of Strongly Connected Cliques

We recall that a *set system* $(V, \mathcal{E} \subseteq 2^V)$ is *(weakly) accessible* if for each $X \in \mathcal{E}$, there is $x \in X$ such that $X \setminus \{x\} \in \mathcal{E}$, and it is *strongly accessible* if in addition for each $X, Y \in \mathcal{E}$ with $Y \subset X$, there is $x \in X \setminus Y$ such that $Y \cup \{x\} \in \mathcal{E}$. In both cases it is assumed that $\emptyset \in \mathcal{E}$. The following two lemmas prove a relaxed notion of weak and strong accessibility.

▶ **Lemma 1.** *Let C be a non-empty* SCQ *of G. There exists $Z \subseteq C$ removable from C and such that $|Z| \leq 2$.*

Proof. As a single vertex and the empty set are SCQs, if $|C| = 3$ any $Z \subset C$ with $|Z| = 2$ is removable, and if $|C| = 1$ or $|C| = 2$ any vertex in C is removable. Suppose then that $|C| \geq 4$ and let us prove that it has a removable vertex.

Let y be an arbitrary vertex of C and suppose that $C' = C \setminus y$ is not strongly connected. Then, there exists a bipartition (X, Y) of C' such that $E(G[C]) \cap (Y \times X) = \emptyset$. As $C = C' \cup \{y\}$ is strongly connected, we must have $w \in X$ and $w' \in Y$ such that $(y, w), (w', y) \in E(G[C])$, and y can reach every vertex in X, and also every vertex from Y can reach y. Since, $|C| \geq 4$ and thus $|C'| \geq 3$, either $|X| \geq 2$ or $|Y| \geq 2$. Assume $|X| \geq 2$: let $z \in X$ be a leaf of a traversal of $X \cup \{y\}$ starting from y (recall that y can reach all vertices in X). As z is a leaf, if we remove it, y can still reach all vertices in $X \setminus \{z\}$. Furthermore, each vertex in $X \setminus \{z\}$ has an edge towards *every* vertex in Y, as C is an SCQ, and every vertex in Y can reach y. Thus $\{y\} \cup (X \setminus \{z\}) \cup Y = C \setminus \{z\}$ is an SCQ, *i.e.*, $Z = \{z\}$ is a removable set in C with $|Z| \leq 2$. If $|X| = 1$, then $|Y| \geq 2$, and the proof is symmetrical by choosing z as a leaf vertex in a traversal of $G[Y \cup \{y\}]$ with the edges reversed, starting from y. ◀

▶ **Lemma 2.** *Given two* SCQs *C and D such that $D \subset C$, there exists $X \subseteq C \setminus D$ addible to D with $|X| \leq 2$.*

Proof. If $|C \setminus D| \leq 2$ the lemma is trivially true, so assume $|C \setminus D| \geq 3$. Any vertex in $C \setminus D$ with edges both towards and from vertices in D is addible to D, so assume that no such vertex exists. Hence, all the vertices in $C \setminus D$ are either sinks or sources *w.r.t.* D, that we denote by K and R, respectively. Any set $\{k, r\}$ with $k \in K$, $r \in R$ and $(k, r) \in E(G[C])$ is an addible set to D. Assume then that no such set does exist: all the vertices in K cannot reach R or D, and all the vertices in R cannot be reached from K or D. This is a contradiction as $C = D \cup K \cup R$ is strongly connected, thus an addible set $\{k, r\}$ exists. ◀

And as any non-maximal SCQ is contained in a larger one, we obtain the following.

▶ **Corollary 3.** *An* SCQ *C is maximal in G if and only if there is no $X \subseteq V(G) \setminus C$ addible to C with $|X| \leq 2$.*

[2] Such an ordering can be computed in linear time by iteratively removing the smallest degree vertex.

Thanks to Lemmas 1 and 2, we can say that the property of being an SCQ belongs to a relaxed class of accessibility, since (i) for each $X \in \mathcal{E}$, there is $Z \subseteq X$ such that $X \setminus Z \in \mathcal{E}$, and (ii) for each $X, Y \in \mathcal{E}$ with $Y \subseteq X$ there is $Z \in X \setminus Y$ such that $Y \cup Z \in \mathcal{E}$, where the size of Z is at most 2. This is a generalization of the definitions of strong and weakly accessible classes, which are obtained from the above by simply setting $|Z| = 1$.

3.2 Maximal Undirected and Strongly Connected Cliques

On top of the accessibility of the problem, we are interested in studying the relationship between the SCQs in G and the cliques in the underlying undirected graph $\mathsf{u}(G)$. Lemma 4 highlights the first basic, but important, relationship.

▶ **Lemma 4.** *Given a directed (not necessarily strongly connected) clique D, the strongly connected components of D are the maximal* SCQ*s in D.*

Proof. Any SCQ C is contained in a strongly connected component of a directed clique by definition, as C is strongly connected and $\mathsf{u}(G[C])$ is a clique. Furthermore, an SCQ may not contain vertices from different strongly connected components, as it would not be strongly connected. Thus, a strongly connected component of D is a maximal SCQ in D. ◀

▶ **Corollary 5.** *A directed clique D contains at most $|D|$ maximal* SCQ*s, which are disjoint.*

3.3 Bounding the Number of Maximal SCQs

For a graph G, let us denote by $gc(G)$ and $gc(\mathsf{u}(G))$ the number of maximal SCQs in G and maximal cliques in $\mathsf{u}(G)$ respectively, and for n, let us denote by $g(n)$ the maximum number of maximal SCQs in an n-vertex graph. From Corollary 5, any maximal SCQ in a directed graph G is contained in a maximal clique of $\mathsf{u}(G)$, *i.e.*, $gc(G) \leq \omega(\mathsf{u}(G)) \cdot gc(\mathsf{u}(G))$. But the number of SCQs in a graph can be much smaller than the number of cliques of its underlying undirected graph: For instance, an n-vertex DAG has exactly n maximal SCQs of size 1 while the number of maximal cliques of its underlying undirected graph can be arbitrarily large. As the maximum number of maximal cliques in an undirected n-vertex graph is $3^{\frac{n}{3}}$ [21], we can immediately conclude that $3^{\frac{n}{3}} \leq g(n) \leq n \times 3^{\frac{n}{3}}$. We can adapt the proof from [21] to show that $g(n)$ is indeed $3^{\frac{n}{3}}$.

Let G be an n-vertex graph, and x, y two vertices of G, and let $G(x; y)$ be defined similarly to [21], as the graph obtained by removing all edges incident to x, and replacing them so that the neighbourhood of x is identical to that of y, *i.e.*, $(x, v) \in E(G(x; y))$ iff $(y, v) \in E(G)$ and $(v, x) \in E(G(x; y))$ iff $(v, y) \in E(G)$. Let $\chi(x)$ be the number of maximal SCQs containing x, let $\alpha(x)$ be the number of new maximal SCQs created by removing x (*i.e.*, subsets of SCQs containing x which become now maximal), and $\beta(x)$ the number of SCQs which are not maximal anymore after removing x[3]. It is straightforward to see that if x and y are not adjacent, the number of SCQs in $G(x; y)$ is given by $gc(G) + \chi(y) - \chi(x) + \alpha(x)$. Indeed all SCQs containing x have been removed, and replaced by $\alpha(x)$ new maximal cliques; furthermore, for each of the $\chi(y)$ maximal SCQs containing y in G, we now have a new one containing x instead of y. If x and y are adjacent, any maximal SCQ containing y in $G(x; y)$ will be simply incremented with x; as a result, the number of maximal SCQs in $G(x; y)$ will

[3] Note that in the undirected case $\alpha(x)$ is bounded by $\chi(x)$, but in the directed case $\alpha(x)$ may be larger than $\chi(x)$ by up to an n factor.

be only $gc(G) - \chi(x) + \alpha(x)$. We are now ready to characterize the graph with the highest number of maximal SCQs.

▶ **Lemma 6.** *Let G be a graph on $n > 4$ vertices with $gc(G) = g(n)$. There exists G^*, an n-vertex graph such that $gc(G^*) = g(n)$ and $\mathsf{u}(G^*)$ is a* complete multipartite graph *with no universal vertices.*

Proof. If $\mathsf{u}(G)$ is a clique, it has at most n maximal SCQs by Lemma 4, so we can replace G with any DAG and still have n maximal SCQs. Thus we can assume that $\mathsf{u}(G)$ is not a clique, and has at least 2 non-universal vertices (*i.e.*, not connected to every other vertex).

For two non-adjacent vertices x and y, we know that $G(x; y)$ and $G(y; x)$ cannot have more SCQs than G. As $\alpha(x) \geq 0$, we have $\chi(x) = \chi(y)$ for any pair of non-adjacent vertices. This implies $\alpha(x) = 0$ for every non-universal vertex x. Thus, if x and y are non-adjacent, $gc(G(x; y)) = gc(G) = g(n)$. From G, we can obtain the graph G^* as follows: for each vertex x, and for each vertex y non-adjacent to x, iteratively replace G with $G(y; x)$.

Observe from the discussion above that $gc(G^*) = gc(G) = g(n)$. Also, $\mathsf{u}(G^*)$ is a complete multipartite graph. Indeed, as each pair of non-adjacent vertices has the same neighbours, we can partition the graph into independent sets such that two vertices in two different independent sets are adjacent. Again, if $\mathsf{u}(G^*)$ is a clique (this may be the case for $n = 4$), we replace G^* with any DAG without compromising the number of maximal SCQs and thus obtaining at least 2 non-universal vertices.

Assume G^* has a universal vertex v. Removing v can decrease the number of maximal SCQs by at most 1. In fact, any SCQ C that is non maximal after removing v, is included in a larger SCQ C', to which v can be added as it is in the same strongly connected component as C (which is a subset of C') and it is connected to all vertices of C'. Thus the only maximal SCQ that can may be lost is the one made by only v, if it is a maximal SCQ.

Let A be the set of universal vertices of G^*. If $|A| > 1$ we can simply remove the edges between two vertices of A and replace each edge (a, b), for $a \in A$ and $b \in V(G^*) \setminus A$, by (b, a). If $|A| = 1$, *i.e.*, $A = \{v\}$ for some v, let us take any independent set I in G^* of size at least 2 (which exists because v is the only universal vertex), and then remove all the edges between v and vertices in I, and replace any edge (v, b), for $b \in V(G^*) \setminus I \cup \{v\}$, by (b, v). As a sink is a maximal SCQ, we can conclude from the paragraph above that the number of SCQs of the obtained graph is still equal to $gc(G) = g(n)$. Since, the underlying undirected graph of this obtained graph is a complete multipartite graph with no universal vertices, we are done. ◀

Thanks to Lemma 6, we can link the number of maximal SCQs in G^* to the number of maximal cliques in $\mathsf{u}(G^*)$. This relation will enable us to give a tight bound for $g(n)$.

▶ **Lemma 7.** *Let $G = (V, E)$ be a graph such that $\mathsf{u}(G)$ is a complete multipartite graph with no universal vertices. Then $gc(G) \leq gc(\mathsf{u}(G))$.*

Proof. Let $\mathcal{S} = \{S_1 \dots S_k\}$ be the set of maximal independent sets of $\mathsf{u}(G)$, which forms a k-partition of V. Let $s(v)$ be the size of the unique maximal independent set S_i containing v; as G and $\mathsf{u}(G)$ have no universal vertices, $s(v) \geq 2$. By definition each maximal SCQ in G is a subset of some maximal clique of $\mathsf{u}(G)$, and recall that each maximal clique of $\mathsf{u}(G)$ (a complete multipartite graph) is obtained by selecting exactly one vertex from each of its maximal independent sets.

Let the occurrence $mc(C)$ of an SCQ C be the number of maximal cliques of $\mathsf{u}(G)$ that contain C, and let the *weight* $w(C)$ of C be $\frac{1}{mc(C)}$, or 0 if $C = \emptyset$. For a maximal clique Q of $\mathsf{u}(G)$, let the weight $w(Q)$ of Q be instead $\sum_{X \text{ a maximal SCQ of } G[Q]} w(X)$. The sum of the weights of all maximal cliques in $\mathsf{u}(G)$ will be *at least* equal to the number of maximal SCQs in

G: any maximal SCQ C will be considered $mc(C)$ times, each time adding $w(C) = 1/mc(C)$, for a total contribution of 1. This sum may be larger, as it can include subsets of maximal cliques which are not maximal SCQs in G, but cannot be smaller.

Let C be a maximal SCQ and let $\mathcal{T} \subseteq \mathcal{S}$ be the maximal independent sets that do not contain any vertex of C. Then, the maximal cliques that contain C are all the ones obtained by adding a single vertex from each independent set in $\mathcal{T}$, thus $mc(C) = \prod_{S_i \in \mathcal{T}} |S_i|$. This means that adding a vertex v to C reduces $mc(C)$ and increases $w(C)$ by a factor $s(v)$.

Let us now consider the highest possible weight of a maximal clique Q in $\mathsf{u}(G)$. Note that, by Corollary 5, the maximal SCQs within Q are at most $|Q|$ and do not overlap. If Q contains a single maximal SCQ X, we have $|X| = |Q|$, $mc(X) = 1$ and $w(X) = 1$, thus $w(Q) = 1$. Otherwise, let X and Y be two maximal SCQs in Q, X being the one with highest weight. Note that $w(X) + w(Y) \leq 2w(X)$. Assume that we could remove a vertex v from Y and add it to X, obtaining X' and Y': we have $w(X') = s(v) \cdot w(X)$, and as $s(v) \geq 2$, $w(X')+w(Y') = w(X)\cdot s(v)+w(Y') \geq 2w(X) \geq w(X)+w(Y)$. This hypothetical operation can increase the total weight of Q but not decrease it, *i.e.*, for any distribution of maximal SCQs in Q, a different that has the size of the largest SCQ increased by one, and that of another one reduced by one, has greater or equal weight. We can repeat this hypothetical step, iteratively enlarging X until we will finally consider a distribution with a single maximal SCQ X of size $|X| = |Q|$. As $w(Q)$ in this case is at least as large as that obtained with any other distribution of maximal SCQs in Q, and as shown above $w(Q) = 1$ in this case, we have $w(Q)$ is always at most 1. Therefore, the number of maximal SCQs in G, *i.e.*, the sum of all weights of the maximal cliques in $\mathsf{u}(G)$ cannot be larger than $gc(\mathsf{u}(G))$. ◀

It is known that the undirected graph with the highest number of maximal cliques is the Moon-Moser graph [21], which is a complete multipartite graph in which as many maximal independent sets as possible have size 3, while the remaining ones may only have size 2.[4] Clearly, such a graph does not have universal vertices, and thus is compatible with the definition of G^*. We can thus say that the underlying graph $\mathsf{u}(G)$ of the graph G^* with the highest number of maximal SCQs will be a Moon-Moser graph. Furthermore, by Lemma 7 we get the upper bound $g(n) = gc(G^*) \leq gc(\mathsf{u}(G^*))$.

It is now easy to prove that this is a tight bound: when G is symmetric (*i.e.*, $(x, y) \in E(G)$ iff $(y, x) \in E(G)$) then the connectivity in G is the same as in $\mathsf{u}(G)$ and each maximal clique in $\mathsf{u}(G)$ will be a maximal SCQ in G. Thus we have $g(n) = gc(G^*) = gc(\mathsf{u}(G^*))$. By combining this lower bound with Lemma 7 and the Moon-Moser bound [21], we can conclude the following (the case $2 \leq n \leq 4$, not covered by Lemma 6, is omitted for space reasons, but can be trivially verified).

▶ **Theorem 8.** *For every integer $n > 1$,*

$$g(n) = \begin{cases} 3^{\frac{n}{3}} & \text{if } n \equiv 0 \pmod{3}, \\ \frac{4}{3} \cdot 3^{\lfloor \frac{n}{3} \rfloor} & \text{if } n \equiv 1 \pmod{3} \\ 2 \cdot 3^{\lfloor \frac{n}{3} \rfloor} & \text{if } n \equiv 2 \pmod{3}. \end{cases}$$

Finally, the same result can be proven for *oriented* graphs, that are directed graphs where each edge may only have one direction, as long as n is not 5 or 6 (this is omitted for space reasons but also involves suitable orientations of Moon-Moser graphs).

[4] Equivalently, the remaining ones may have size 4. However it is not necessary to consider this case.

4 Listing Maximal SCQs

While the number of maximal SCQs in a graph G is at most n times the number of maximal cliques of its underlying (undirected) graph $\mathsf{u}(G)$, and each maximal SCQ of G is contained in some maximal clique of $\mathsf{u}(G)$, one *cannot* efficiently use output-polynomial algorithms for listing maximal cliques in undirected graphs in order to list the maximal SCQs of a graph. For example, any orientation of a Moon-Moser n-vertex graph into a DAG has exactly n maximal SCQs, while its underlying graph has $\Theta(3^{\frac{n}{3}})$ maximal cliques. This strategy would hence take exponential running time to find just a linear number of maximal SCQs.

In this section we design an algorithm that enumerates all maximal SCQs of a graph $G = (V, E)$ with polynomial delay.

Intuitively, given a maximal SCQ (called sometimes a solution) S, our algorithm uses the vertices in $V \setminus S$ to find other solutions similar to S; we refer to this process as *visiting* S. By visiting these newly found solutions the algorithm eventually finds all solutions in G.

4.1 Algorithm Description

The algorithm, which we call SCQ-ENUM, is described in Algorithm 1. SCQ-ENUM uses a RESULT set, which will store all solutions found so far. The primitive *contains*(S, RESULT) is a subroutine that returns true if $S \in \text{RESULT}$, *i.e.*, S has already been found and does not need to be visited again, and the primitive *add*(S, RESULT) adds S to the RESULT set. Finally, SCQ-ENUM exploits the function COMPLETE(X, A), which will iteratively add the lexicographically minimum addible vertex or pair of vertices from A to a SCQ X, until X is maximal *w.r.t.* A, and return it. For brevity, COMPLETE(X) represents COMPLETE(X, V). Thanks to the accessibility proven in Lemma 2 and Corollary 3, COMPLETE(X) will surely return a maximal SCQ. We recall that we assume the graph given with the degeneracy ordering, and we consider that ordering and its associated lexicographic one in the algorithm (see Section 2). The primitive MIN-LEX$(\mathcal{T})$ finds the minimum in the collection $\mathcal{T} \subseteq 2^V$.

SCQ-ENUM is in the same spirit as the one for listing the maximal cliques in an undirected graph [10]. It does a DFS traversal of the graph of solutions where (S, S') is an edge if S' can be obtained from S by adding a new vertex (or a pair of vertices), removing its (their) non-neighbours and finally completing the obtained set into a maximal SCQ. Let us describe the algorithm.

SCQ-ENUM consists in calling the function ENUM(S), with S a maximal SCQ. In turn, ENUM(S) will find *all* solutions that have a non-empty overlap with S. The function will consider all vertices $x \in V \setminus S$, and for each of them will try to generate a new maximal SCQ containing x and some vertices of S: by calling $X \leftarrow$ COMPLETE$(\{x\}, I)$ the algorithm will get the SCQ containing x, maximal *w.r.t.* the induced subgraph $G[S \cup \{x\}]$; note that there is only one such SCQ, as $G[S \cup \{x\}]$ is a clique in $\mathsf{u}(G[S \cup \{x\}])$ (see Lemma 4). Then X is extended with COMPLETE(X) so that it is maximal *w.r.t.* G, *i.e.*, a solution. Then, in the second for loop, the same process is repeated for pairs of vertices rather than single vertices. Every time a solution S' is found, we recur in ENUM(S'), which will visit S', adding it to the RESULT set and finding more solutions starting from S'. If S' is already in the RESULT set, however, it means it was already visited and all the relative solutions have been found, thus we can ignore it and backtrack.

Algorithm 1: SCQ-ENUM

```
Input   : A graph G=(V,E)
Output  : The set RESULT containing all maximal SCQs in $G$
Global  : RESULT set, initially empty
for $v \in V$ do
    ENUM(COMPLETE($\{v\}$))
Function ENUM($S$)
    if $contains(S, \text{RESULT})$ then return
    $add(S, \text{RESULT})$
    foreach $x \in V \setminus S$ do
        $I \leftarrow (S \cap N(x) \cup \{x\})$
        $X \leftarrow$ COMPLETE($\{x\}, I$)
        if $X = \{x\}$ then continue
        ENUM(COMPLETE($X$))
    foreach $\{y, z\} \subseteq V \setminus S$ do
        $I \leftarrow (S \cap N(y) \cap N(z) \cup \{y, z\})$
        $X \leftarrow$ COMPLETE($\{y, z\}, I$)
        if $X = \{y, z\}$ then continue
        ENUM(COMPLETE($X$))
```

4.2 Correctness

We will hereby prove the correctness of SCQ-ENUM. The principle of finding maximal solutions from other solutions is used by many enumeration algorithms, but this has so far been applied exclusively to properties with *strong accessibility* [2], such as hereditary [10, 14, 26], or connected-hereditary [4, 7].

Thanks to the results obtained in Section 3, however, we will be able to prove the correctness of our technique, despite SCQs not being strongly (or even weakly) accessible, similar to [7]. In the following, given two SCQs S and T, let $S \cap_{scq} T$ be the largest SCQ in $S \cap T$; we recall that this may be a single vertex, which is indeed an SCQ.

Proving that no solution is found twice by SCQ-ENUM is trivial, as duplication is removed by the RESULT set, and since every output is a maximal SCQ since it is the result of a COMPLETE call, we only need to prove that every maximal SCQ is found:

▶ **Theorem 9.** SCQ-ENUM *finds all and only maximal* SCQ*s exactly once.*

Proof. Let T be any solution not yet found by the algorithm. Let S be the solution found by SCQ-ENUM which maximizes $|S \cap_{scq} T|$. Note that $|S \cap_{scq} T| \geq 1$: for any $v \in T$, the algorithm will visit $C =$ COMPLETE($\{v\}$), a maximal SCQ containing v, so $|C \cap_{scq} T| \geq 1$. Now let $Z = S \cap_{scq} T$. We have $Z \neq T$, otherwise T would not be maximal, and by Lemma 2 there exists $Y \subseteq T \setminus Z$ with $1 \leq |Y| \leq 2$ s.t. $Z \cup Y$ is an SCQ. Note that Y is not contained in S, as otherwise $Z \cup Y$ would be a larger SCQ in $S \cap_{scq} T$. Three cases are possible: (i) $Y = \{x\}$, then $x \in V \setminus S$ and x is considered in the first *for* loop. (ii) $Y = \{y, z\} \subseteq V \setminus S$, then $\{y, z\}$ is considered in the second *for* loop. (iii) $|Y| = \{y, z\}$, with $y \in V \setminus S$ and $z \in S$, then y is considered in the first *for* loop and we will have $z \in S \cap N(y) \cup \{y\}$.

In all these cases, the SCQ X (maximal in I) that is found, will contain $Z \cup Y$ by Lemma 4. When we execute COMPLETE(X), we will either find T, or a maximal SCQ S' that contains

```
Algorithm 2: COMPLETE($X, A$)
  Input   : $X$, an SCQ, and $A \subseteq V$, a set of vertices
  Output  : $X' \supseteq X$, an SCQ maximal with respect to $A$
Function COMPLETE($X, A$)
    while $EXT \leftarrow$ MIN-EXTENSION($X, A$) $\neq$ null do
        $X \leftarrow X \cup EXT$;
    return $X$;
Function MIN-EXTENSION($X, A$)
    $ADD \leftarrow \{Y \subseteq A \setminus X : 1 \leq |Y| \leq 2$ and $X \cup Y$ is an SCQ$\}$;
    return MIN-LEX($ADD$)
```

$Z \cup Y$. As $|Z \cup Y| > |Z|$, we have $|S' \cap_{scq} T| > |S \cap_{scq} T|$. By induction, when visiting S' we will either find T, or S'' such that $|S'' \cap_{scq} T| > |S' \cap_{scq} T|$, until eventually, in at most $|T|$ such steps, SCQ-ENUM will find T. ◀

5 Complexity Analysis

We now analyze the complexity of SCQ-ENUM, *i.e.*, Algorithm 1, showing that it lists maximal SCQs with delay $O(\min(\omega(G)d^2\Delta^2, m^2))$. Recall that m, n, Δ, d and $\omega(G)$ are respectively the number of edges, number of vertices, maximum vertex degree, degeneracy and maximum size of an SCQ in G, and that the vertices are $v_1, \ldots, v_n$ are given in a *degeneracy ordering.* Firstly, we bound the complexity of the function COMPLETE:

▶ **Lemma 10.** COMPLETE(X, A) *(Algorithm 2) can be executed in time* $O(\min(d\Delta, m))$.

Proof. Consider the vertices in A adjacent to all vertices of X, *i.e.* $\bigcap_{x \in X}(N(x)) \cap A$, and partition them in three sets, each stored in increasing lexicographical order: SINK contains all the *sinks w.r.t.* X; SOURCE the *sources w.r.t.* X, and BOTH all vertices that have at least one edge *from* and one *towards* some vertex in X (*i.e.*, neither sinks nor sources *w.r.t.* to X). The computing time is the sum of the degrees of vertices in X, *i.e.* $\min(\omega(G)\Delta, m)$ and the total size $|\text{SINK}| + |\text{SOURCE}| + |\text{BOTH}|$ of the sets is bounded by Δ.

As each step adds either one or two vertices to X, and $|X|$ is bounded by $\omega(G)$, we will have at most $\omega(G)$ steps. Whenever we add a vertex a to X, we can update the SINK, SOURCE, BOTH sets by only looking at $N(a)$: any non neighbour of a is excluded from these sets, any vertex in SINK with an edge towards a, or vertex in SOURCE with an edge from a is moved in BOTH. This takes $O(|N(a)|)$ time, thus $O(\min(\omega(G)\Delta, m))$ time for all updates.

If the BOTH set is not empty, we can find the (lexicographically) smallest x in $O(1)$ time. Then, we need to find the smallest pair $a \in \text{SINK}, b \in \text{SOURCE}$ s.t. there is an edge from a to b, if it exists. We do this by scanning vertices in $\text{SINK} \cup \text{SOURCE}$ in order, and for each scanning its *forward* neighbours, still in order; we stop at the first pair that verifies the property. We never need to consider the same pair twice, as the condition (edge from a to b) will stay false (although the vertices might be later moved to BOTH and still enter X), thus the total cost will be $O(\min(d\Delta, m))$ for all steps. Finally, the smallest among x and $\{a, b\}$ corresponds to MIN-EXTENSION(X, A); we add it to X and update the SINK, SOURCE, BOTH sets. The total cost is given by $O(\min(\omega(G)\Delta, m) + \min(d\Delta, m)) = O(\min(d\Delta, m))$ ◀

Finally, we are ready to give the time complexity of SCQ-ENUM:

▶ **Lemma 11.** SCQ-ENUM *(Algorithm 1) has* $O(\min(\omega(G)d^2\Delta^2, m^2))$ *time delay.*

Proof. Let us first focus on the amortized cost per solution, *i.e.*, the cost of an execution of ENUM(S) without considering children recursive calls (which lead to other solutions). To compute *contains*(S, RESULT) and *add*(S, RESULT) we store RESULT as a *trie*, whose depth will be $O(\omega(G))$, and degree will be bounded by Δ as SCQs are made of adjacent vertices.[5] Checking the existence and adding a solution to this trie takes time $O(\min(\omega(G)\log(\Delta), m))$.

In the *for* loops we only need to consider x and $\{y, z\}$ s.t. $I \neq \{x\}$ and $I \neq \{y, z\}$, thus only vertices with a neighbour in S: for x we have $|S|\Delta \leq \min(\omega(G)\Delta, n)$ choices; as for $\{y, z\}$ we have $\min(\omega(G)\Delta, n)$ choices for y, and for each y up to d choices for z (we only need to consider each pair once, *e.g.* when $y < z$, so we only scan the *forward* neighbours of y), for a total of $\min(\omega(G)d\Delta, m)$ choices. The cost of each loop is given by $O(\Delta)$ for computing I, and $O(\min(d\Delta, m))$ to compute X and COMPLETE(X). If the recursive call ENUM(COMPLETE(X)) will generate a new solution, the cost will be attributed to the child solution; otherwise the recursive call will only perform the *contains*(S, RESULT) call.

The total cost of an iteration of ENUM(S) is thus the cost of the *contains*/*add* procedures, plus the number of execution of the loops times the cost of a loop iteration, *i.e.*, $O(\min(\omega(G)\log(\Delta), m) + \min(\omega(G)d\Delta, m) \cdot (\min(\omega(G)\log(\Delta), m) + \min(d\Delta, m)))$. Thus, the cost per solution is bounded by both $O(\omega(G)d^2\Delta^2)$ and $O(m^2)$.

Finally, as each recursive call outputs a solution, we can exploit the alternative output method by Uno [27], as done in [20, 10]: we output a solution at the beginning of a recursive call when the depth of the recursion tree is even, and at the end when it is odd; this way the delay of the algorithm will be equal to the amortized cost per solution. ◀

Calling α the number of solutions, this gives us a total time of $\alpha \cdot O(\min(\omega(G)d^2\Delta^2, m^2))$. The space complexity is dominated by the size of the RESULT set, that is $O(\alpha \cdot \omega(G))$ as it will contain α solutions of size bounded by $\omega(G) \leq d + 1 \leq n$. While α is potentially exponential, we remark that SCQ-ENUM can still be efficiently applied to analyze real world networks: recalling Corollary 5 and the discussion in Section 3.3, we have that α will only be up to a factor $\omega(\mathsf{u}(G)) \leq n$ larger than the number of maximal (undirected) cliques in $\mathsf{u}(G)$. It is generally agreed upon that real-world networks are sparse [12, 15], and as such contain an extremely small number of maximal cliques compared to the theoretical maximum [24]. Furthermore, the number of maximal cliques is actually polynomial when the degeneracy (or arboricity) is bounded [12], which is the case in many sparse networks.

6 Conclusions and Future Work

In this work we proposed a model for communities in directed graphs, that of maximal strongly connected cliques. We analyzed this model, giving tight bounds on the number of such cliques in an n-vertex graph and proving some accessibility properties. We exploited these properties to produce SCQ-ENUM, an algorithm that lists maximal strongly connected cliques with polynomial delay, *i.e.*, $O(\min(\omega(G)d^2\Delta^2, m^2))$, that can be a valid tool for analyzing the community structures of directed real-world networks.

Future work is focused in two directions: the first is using the proposed algorithm to discover new community structures in directed networks, while the second is to further investigate the generalized definition of accessibility given by the existence of an addible (or removable) set of elements of bounded size to each non-maximal solution.

[5] The root of the trie has degree up to n; we store this as a vector of size n, accessible in $O(1)$ time.

References

1 Eralp Abdurrahim Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM J Comput*, 2(1):1–6, 1973.

2 Hiroki Arimura and Takeaki Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1088–1099. SIAM, 2009.

3 Baruch Awerbuch and Yuval Shavitt. Topology aggregation for directed graphs. *IEEE/ACM Transactions On Networking*, 9(1):82–90, 2001.

4 Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. Efficient enumeration of maximal k-plexes. In *SIGMOD*, pages 431–444, New York, NY, USA, 2015. ACM.

5 Coenraad Bron and Joep Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

6 Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95. Springer, 2000.

7 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147 – 1159, 2008.

8 Carlo Comin and Romeo Rizzi. An improved upper bound on maximal clique listing via rectangular fast matrix multiplication. *CoRR*, abs/1506.01082, 2015.

9 Alessio Conte, Roberto Grossi, Andrea Marino, Romeo Rizzi, and Luca Versari. Directing road networks by listing strong orientations. *IWOCA*, pages 83–95, 2016.

10 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In *ICALP*, 2016.

11 Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.

12 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013.

13 Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

14 Komei Fukuda. Note on new complexity classes ENP, EP and CEP, 1996. Accessed: 02-2016. URL: `https://www.inf.ethz.ch/personal/fukudak/old/ENP_home/ENP_note.html`.

15 Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *WG*, pages 159–167. Springer, 2006.

16 Björn H. Junker and Falk Schreiber. *Analysis of biological networks*, volume 2. John Wiley & Sons, 2011.

17 Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: Measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer, 1999.

18 Darong Lai, Hongtao Lu, and Christine Nardini. Finding communities in directed networks by pagerank random walk induced network embedding. *Physica A: Statistical Mechanics and its Applications*, 389(12):2443–2454, 2010.

19 Elizabeth A. Leicht and Mark E.J. Newman. Community structure in directed networks. *Physical review letters*, 100(11):118703, 2008.

20 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *Algorithm Theory-SWAT 2004*, pages 260–272. Springer, 2004.

21 John W. Moon and Leo Moser. On cliques in graphs. *Isr J Math*, 3(1):23–28, 1965.

22 Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. Clique relaxation models in social network analysis. *Handbook of Optimization in Complex Networks*, pages 143–162, 2012.

23 Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *P Natl Acad Sci USA*, 105(4):1118–1123, 2008.

24 Matthew C. Schmidt, Nagiza F. Samatova, Kevin Thomas, and Byung-Hoon Park. A scalable, parallel algorithm for maximal clique enumeration. *J Parallel Distr Com*, 69(4):417–428, 2009.

25 John Scott. *Social network analysis.* Sage, 2012.

26 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J Comput*, 6(3):505–517, 1977.

27 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. *NII Technical Report NII-2003-004E, Tokyo, Japan*, 4, 2003.

28 Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

Square-Contact Representations of Partial 2-Trees and Triconnected Simply-Nested Graphs*†

Giordano Da Lozzo[1], **William E. Devanny**[2], **David Eppstein**[3], and **Timothy Johnson**[4]

1 Computer Science Department, University of California, Irvine, USA
`gdalozzo@uci.edu`

2 Computer Science Department, University of California, Irvine, USA
`wdevanny@uci.edu`

3 Computer Science Department, University of California, Irvine, USA
`eppstein@uci.edu`

4 Computer Science Department, University of California, Irvine, USA
`tujohnso@uci.edu`

Abstract

A *square-contact representation* of a planar graph $G = (V, E)$ maps vertices in V to interior-disjoint axis-aligned squares in the plane and edges in E to adjacencies between the sides of the corresponding squares. In this paper, we study *proper* square-contact representations of planar graphs, in which any two squares are either disjoint or share infinitely many points.

We characterize the partial 2-trees and the triconnected cycle-trees allowing for such representations. For partial 2-trees our characterization uses a simple forbidden subgraph whose structure forces a separating triangle in any embedding. For the triconnected cycle-trees, a subclass of the triconnected simply-nested graphs, we use a new structural decomposition for the graphs in this family, which may be of independent interest. Finally, we study square-contact representations of general triconnected simply-nested graphs with respect to their outerplanarity index.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Square-Contact Representations, Partial 2-Trees, Simply-Nested Graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.24

1 Introduction

Contact representations of graphs, in which the vertices of a graph are represented by non-overlapping or non-crossing geometric objects of a specific type, and edges are represented by tangencies or other contacts between these objects, form an important line of research in graph drawing and geometric graph theory. For instance, the Koebe–Andreev–Thurston circle packing theorem states that every planar graph is a contact graph of circles [13]. Other types of contact representations that have been studied include contacts of unit circles [2, 9], line segments [10], circular arcs [1], triangles [8], L-shaped polylines [3], and cubes [7].

* Supported in part by the National Science Foundation under Grants CCF-1228639, CCF-1618301, and CCF-1616248. This article also reports on work supported by the U.S. Defense Advanced Research Projects Agency (DARPA) under agreement no. AFRL FA8750-15-2-0092. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

† A full version of the paper is available at [5], `https://arxiv.org/abs/1710.00426`.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 24; pp. 24:1–24:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Schramm's monster packing theorem [11] implies that every planar graph can be represented by the tangencies of translated and scaled copies of any smooth convex body in the plane. However, it is more difficult to use this theorem for non-smooth shapes, such as polygons: when k bodies can meet at a point, the monster theorem may pack them in a degenerate way in which separating k-cycles, and their interiors, shrink to a single point.

In this paper we study one of the simplest cases of contact representations that cannot be adequately handled using the monster theorem: contact systems of axis-parallel squares. We distinguish between *proper* and *improper* contacts: a proper contact representation disallows squares that meet only at their corners, while an *improper* or *weak* contact representation allows corner-corner contacts of squares. These weak contacts may represent edges of the graph, but they are also allowed between squares that should be non-adjacent. The weak contact representations by squares were shown by Schramm [12] to include all of the proper induced subgraphs of maximal planar graphs that have no separating 3-cycles or 4-cycles. However, a characterization of the graphs having proper contact representations by squares remains elusive.

There is a simple necessary condition for the existence of a proper contact representation by squares. No three properly-touching squares can surround a nonzero-area region of the plane. Therefore, if every embedding of a planar graph G with four or more vertices has a separating triangle or a triangle as the outer face, then G cannot have a proper contact representation. Our main results show that this necessary condition is also sufficient for two notable families of planar graphs: partial 2-trees (including series-parallel graphs) and triconnected cycle-trees (including the Halin graphs). However, we show that this necessary condition is not sufficient for the existence of weak and proper square-contact representations of 3-outerplanar and 2-outerplanar triconnected simply-nested graphs.

Due to space limits, full versions of omitted or sketched proofs are provided in [5].

2 Preliminaries

For standard graph theory concepts and definitions related to planar graphs, their embeddings, and connectivity we refer the reader, e.g., to [6] and to [5].

The graphs considered in this paper are planar, finite, simple, and connected. We denote the vertex set V and the edge set E of a graph $G = (V, E)$ by $V(G)$ and $E(G)$, respectively. Let H and G be two graphs. We say that G is H-free if G does not contain a subgraph isomorphic to H. The complete k-partite graph $K_{|V_1|,\dots,|V_k|}$ is the graph $(V = \bigcup_{i=1}^{k} V_i, E = \bigcup_{i<j} V_i \times V_j)$.

Series-parallel graphs and partial 2-trees. A *two-terminal series-parallel* graph G with source s and target t can be recursively defined as follows:

- **(i)** Edge st is a two-terminal series-parallel graph. Let $G_1, \dots, G_k$ be two-terminal series-parallel graphs and let s_i and t_i be the source and the target of G_i, respectively, with $1 \leq i \leq k$.
- **(ii)** The *series composition* of $G_1, \dots, G_k$ obtained by identifying s_i with t_{i+1}, for $i = 1, \dots, k-1$, is a two-terminal series-parallel graph with source s_k and target t_1; and
- **(iii)** the *parallel composition* of $G_1, \dots, G_k$ obtained by identifying s_i with s_1 and t_i with t_1, for $i = 2, \dots, k$, is a two-terminal series-parallel graph with source s_1 and target t_1.

A *series-parallel graph* is either a single edge or a two-terminal series-parallel graph with the addition of an edge, called *reference edge* joining s and t. Clearly, series-parallel graphs are 2-connected. A series-parallel graph G with reference edge e is naturally associated with a rooted tree T, called the *SPQ*-tree of G. Each internal node of T, with the exception of the one associated with e, corresponds to a two-terminal series-parallel graph. Nodes of T are of three types: S-, P-, and Q-*nodes*. Further, tree T is rooted to the Q-node corresponding to e.

Let μ be a node of T with terminals s and t and children $\mu_1, \dots, \mu_k$, if any. Node μ has an associated multigraph, called the *skeleton* of μ and denoted by $skel_\mu$, containing a *virtual edge* $e_i = s_i t_i$, for each child μ_i of μ. Skeleton $skel_\mu$ shows how the children of μ, represented by "virtual edges", are arranged into μ. The skeleton $skel_\mu$ of μ is:
(i) edge st, if μ is a leaf Q-node,
(ii) the multi-edge obtained by identifying the source s_i and the target t_i of each virtual edge e_i, for $i = 1, \dots, k$, with a new source s and and new target t, respectively, or
(iii) the path $e_1, \dots, e_k$, where virtual edge e_i and e_{i+1} share vertex $s_i = t_{i+1}$, with $1 \leq i < k$.
If μ is an S-node, then we denote by $\ell(\mu)$ the length of $skel_\mu$, i.e., $\ell(\mu) = k$.

For each virtual edge e_i of $skel_\mu$, recursively replace e_i with the skeleton $skel_{\mu_i}$ of its corresponding child μ_i. The two-terminal series-parallel subgraph of G that is obtained in this way is the *pertinent graph* of μ and is denoted by G_μ. We have that G_μ is:
(i) edge st, if μ is a Q-node,
(ii) the series composition of the two-terminal series-parallel graphs $G_{\mu_1}, \dots, G_{\mu_k}$, if μ is an S-node, and
(iii) the parallel composition of the two-terminal series-parallel graphs $G_{\mu_1}, \dots, G_{\mu_k}$, if μ is a P-node.
We denote by G^-_μ the subgraph of G_μ obtained by removing from it terminals s and t together with their incident edges.

A 2-*tree* is a graph that can be obtained from an edge by repeatedly adding a new vertex connected to two adjacent vertices. Every 2-tree is planar and 2-connected. A *partial* 2-*tree* is a subgraph of a 2-tree. Equivalently, *partial* 2-*tree* can be defined as the K_4-minor-free graphs. In particular, the series-parallel graphs are exactly the 2-connected partial 2-trees.

Simply-nested graphs. Let G be an embedded planar graph and let $G_1, \dots, G_k$ be the sequence of embedded planar graphs such that $G_1 = G$, graph G_{i+1} is obtained from G_i be removing all the vertices incident to the outer face of G_i together with their incident edges, and G_k is outerplanar. We say that the embedding of G is k-*outerplanar.* A graph is k-*outerplanar* if it admits a k-outerplanar embedding. The set V_i of vertices incident to the outer face of G_i is the i-*th level* of G. A k-outerplanar graph is *simply-nested* [4] if, for $i = 1, \dots, k-1$, graphs $G[V_i]$ are chordless cycles and $G[V_k]$ is either a cycle or a tree.

We define *cycle-trees* and *cycle-cycles* the 2-outerplanar simply-nested graphs whose internal level is a tree and a cycle, respectively. The 2-outerplanar 3-connected simply-nested graphs have a nice geometric interpretation. Similarly to the Halin graphs, which are the graphs of polyhedra containing a face that share an edge with all other faces, 3-connected cycle-trees are the graphs of polyhedra containing a face touched by all other faces. Analogously, the 3-connected cycle-cycle graphs with no chords on the inner cycle are the graphs of polyhedra in which there exist two disjoint faces that are both touched by all other faces.

Square-contact representations. Let $G = (V, E)$ be a planar graph. A *square-contact representation* Γ of G maps each vertex $v \in V$ to an axis-aligned square $S_\Gamma(v)$ in the plane, such that, for any two vertices $u, v \in V$, squares $S_\Gamma(u)$ and $S_\Gamma(v)$ are interior-disjoint, and the sides of $S_\Gamma(u)$ and $S_\Gamma(v)$ touch if and only if $uv \in E$. A square-contact representation of G is *proper* if any two touching squares share infinitely many points, i.e., they cannot share only a corner point, and *non-proper*, otherwise. When the square-contact representation is clear from the context, we may choose to drop the Γ subscript and just use $S(v)$ to refer to the square for vertex v. In the remainder of the paper, we only consider proper square-contact representations and refer to such representations simply as square-contact representations.

Geometric transformations. Let G be planar graph and let Γ be a square-contact representation of G. Also, let p be any point in Γ. We define the $\nearrow$-, $\nwarrow$-, $\swarrow$-, and $\searrow$-*quadrant* of p in Γ as the first, second, third, and fourth quadrant around p, respectively. Suppose that the half-lines delimiting the $\swarrow$-quadrant of p in Γ do not intersect the interior of any square in Γ. Also, let Γ' be the part of Γ lying in the $\swarrow$-quadrant of p. Then, a $\overset{\swarrow}{p}$*-scaling* of Γ *by a factor* $\alpha > 0$ is a square-contact representation Γ^* defined as follows; see, e.g., Fig. 3. Initialize $\Gamma^* = \Gamma$ and remove from Γ^* the drawing of the squares contained in the interior of Γ'. Then, insert into Γ^* a copy Γ'' of Γ' scaled by α such that the upper-right corner of Γ'' coincides with p. Clearly, depending on the scale factor α, drawing Γ^* may or may not be a square-contact representation of G (as adjacencies may be lost or gained). In the following, we refer to the case in which $\alpha > 1$ simply as a $\overset{\swarrow}{p}$*-scaling* of Γ and to the case in which $0 < \alpha < 1$ as a *negative* $\overset{\swarrow}{p}$*-scaling* of Γ. The definitions of $\overset{\circ}{p}$*-scaling* and *negative* $\overset{\circ}{p}$*-scaling*, with $\circ \in \{\nwarrow, \searrow, \nearrow\}$, are analogous. Finally, let v be a vertex of G and let x, y, z, and w be the upper-left, lower-left, lower-right, and upper-left corner points of $S(v)$ in Γ. A $\overset{\nwarrow}{v}$*-scaling*, $\overset{\swarrow}{v}$*-scaling*, $\overset{\searrow}{v}$*-scaling*, $\overset{\nearrow}{v}$*-scaling* of Γ is a $\overset{\nwarrow}{x}$*-scaling*, $\overset{\swarrow}{y}$*-scaling*, $\overset{\searrow}{z}$*-scaling*, $\overset{\nearrow}{w}$*-scaling* of Γ, respectively.

3 Partial 2-Trees

In this section, we study square-contact representations of partial 2-trees and give the following simple characterization for graphs in this family admitting such representations.

▶ **Theorem 1.** *Let G be a partial 2-tree. Then, the following statements are equivalent:*
(i) *G is $K_{1,1,3}$-free,*
(ii) *G admits an embedding without separating triangles, and*
(iii) *G admits a square-contact representation.*

In order to prove Theorem 1, we first show that, without loss of generality, we can restrict our attention to the biconnected partial 2-trees, i.e., the series-parallel graphs.

▶ **Lemma 1.** *Let G be a $K_{1,1,3}$-free partial 2-tree. Then, there exists a $K_{1,1,3}$-free series-parallel graph G^* such that $G \subset G^*$ and G admits a square-contact representation if G^* does.*

Sketch. Let $\beta(H)$ denote the number of blocks, i.e., the maximal biconnected components, of a graph H. Adding to G a new vertex connected to two vertices in $V(G)$ incident to the same cut-vertex of G, belonging to different blocks, and sharing a common face yields a graph G' such that $\beta(G') = \beta(G) - 1$. It is easy to see that G' is $K_{1,1,3}$-free and that G' does not contain K_4 as a minor. Hence, repeating such an augmentation eventually yields a series-parallel graph G^* that is $K_{1,1,3}$-free. Also, by construction, two vertices in $V(G)$ are adjacent in G^* if and only if they are adjacent in G. Therefore, a square-contact representation of G can be derived from a square-contact representation Γ^* of G^*, by removing from Γ^* all the squares corresponding to vertices in $V(G^*) \setminus V(G)$. ◀

As already observed in Section 1, an embedding without separating triangles is necessary for the existence of a square-contact representation, and $K_{1,1,3}$ has no embedding without separating triangles. Thus, $(iii) \Rightarrow (ii) \Rightarrow (i)$ are immediate. To complete the proof of Theorem 1, we show how to construct a square-contact representation of any $K_{1,1,3}$-free series-parallel graph, proving that $(i) \Rightarrow (iii)$. We formalize this result in the next theorem.

▶ **Theorem 2.** *Every $K_{1,1,3}$-free series-parallel graph admits a square-contact representation.*

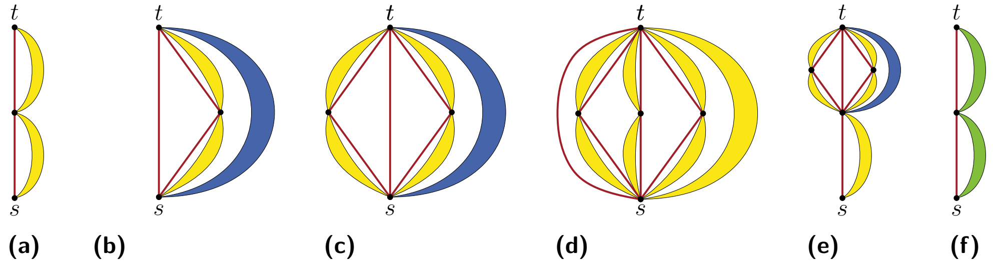


Figure 1 (a) A critical S-node, (b) an almost-bad P-node, (c) a bad P-node, (d) a forbidden P-node, (e) an S-node of `Type B`, and (f) an S-node of `Type C`. Yellow, green, and blue regions represent parallel compositions of any number of S-nodes, at most one critical S-node and any number of non-critical S-nodes, and any number of non-critical S-nodes, respectively.

Let G be a series-parallel graph and let T be the SPQ-tree of G with respect to any reference edge. We start with some definitions; refer to Fig. 1. Let μ be an S-node in T. We say that μ is *critical*, if $skel_\mu = s–x–t$ and the two children of μ both contain an edge between their terminals, i.e., $sx, xt \in E(G_\mu)$, and *non-critical*, otherwise. Let μ be a P-node in T containing an edge between its terminals. We say that μ is *almost bad*, if it has exactly one critical child, *bad*, if it has exactly two critical children, and *forbidden*, if it has more than two critical children. Finally, let μ be a P-node in T. We say that μ is *good*, if it is neither bad, nor almost bad, nor forbidden.

We now assign one of three possible types to each S-node μ in T as follows (for each child μ_i of μ, we denote the two terminals of G_{μ_i} as s_i and t_i).

Type A Node μ is of `Type A`, if either $\ell(\mu) > 2$ or $\ell(\mu) = 2$ and at least one child of μ does not contain an edge between its terminals, i.e., $|\{s_1t_1, s_2t_2\} \cap E(G_\mu)| < 2$.

Type B Node μ is of `Type B`, if $\ell(\mu) = 2$, all its children contain an edge between their terminals, and at least one of them is a bad P-node.

Type C Node μ is of `Type C`, if $\ell(\mu) = 2$, and all its children contain an edge between their terminals, and none of them is a bad P-node.

Observe that S-nodes of `Type B` and of `Type C` are also critical.

Let G be a $K_{1,1,3}$-free series-parallel graph and let T be the SPQ-tree of G with respect to any reference edge. We have the following simple observations regarding the P-nodes in T.

▶ **Observation 1.** *SPQ-tree T contains no forbidden P-node; refer to Fig. 1(d).*

▶ **Observation 2.** *Let μ be a P-node in T with terminals s and t such that $st \in E(G_\mu)$. Then, none of the children of μ is of `Type B` and at most two children of μ are of `Type C`.*

We now consider special square-contact representations for the pertinent graphs of the S-nodes in T. Let Γ_μ be a square-contact representation of G_μ. We say that Γ_μ is either a *rectangular*, *L-shape*, or *pipe drawing* of G_μ, if it satisfies the following conditions; refer to Fig. 2.

Rectangular drawing $S(t)$ lies to the left and above $S(s)$ and the drawing Γ^-_μ of G^-_μ in Γ_μ lies to the right of $S(t)$ and above $S(s)$; also, all the squares of Γ^-_μ whose left side (bottom side) is collinear with the right side of $S(t)$ (with the top side of $S(s)$) are adjacent to $S(t)$ (to $S(s)$).

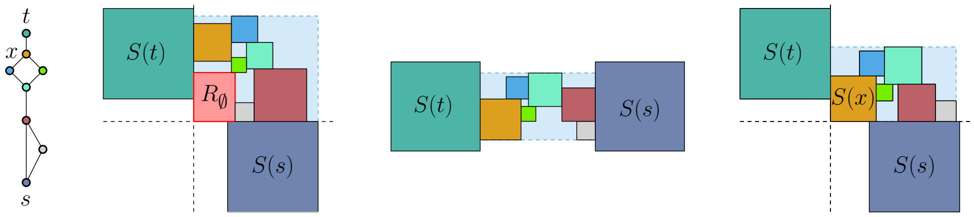


Figure 2 From left to right: pertinent G_μ of an S-node μ with terminals s and t, L-shape and pipe drawings of G_μ, respectively, and a rectangular drawing of an S-node ν with pertinent $G_\nu = G_\mu \cup sx$. The L-shape region and horizontal pipe enclosing G^-_μ and the rectangle enclosing G^-_ν are shaded blue.

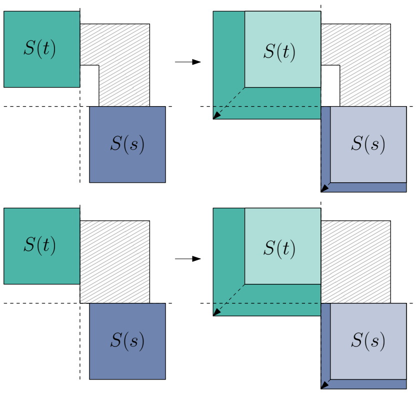


Figure 3 Transforming Γ_τ into Γ_ρ.

L-shape drawing Γ_μ is a rectangular drawing in which there exists a rectangular region (red region $R_\emptyset$ in Fig. 2) inside the bounding box of Γ^-_μ whose interior does not intersect any square in Γ^-_μ and whose lower-left corner lies at the intersection point between the vertical line passing through the right side of $S(t)$ and the horizontal line passing through the top side of $S(s)$.

Pipe drawing $S(t)$ lies to the left of $S(s)$ and the drawing Γ^-_μ of G^-_μ in Γ_μ lies to the right of $S(t)$ and to the left of $S(s)$; also, all the squares of Γ^-_μ whose left side (right side) is collinear with the right side of $S(t)$ (with the left side of $S(s)$) are adjacent to $S(t)$ (to $S(s)$).

In the following, we generally refer to a drawing of an S-node μ in T (of G_μ) which is either an L-shape drawing, a pipe drawing, or a rectangular drawing as a *valid drawing* of μ (of G_μ).

Let Γ^-_μ be the square-contact representation of G^-_μ contained in Γ_μ. Observe that Γ^-_μ lies in the interior of an orthogonal hexagon with an internal angle equal to 270°, i.e., an *L-shape polygon* (or, simply, *L-shape*), if Γ^-_μ is an L-shape drawing. Also, Γ^-_μ lies in the interior of a rectangle whose opposite vertical sides are adjacent to the right side of $S(t)$ and to the left side of $S(s)$, i.e., a *horizontal pipe*, if Γ^-_μ is a pipe drawing. Finally, Γ^-_μ lies in the interior of a rectangle whose left and bottom side are adjacent to the right side of $S(t)$ and to the top side of $S(s)$, respectively, if Γ^-_μ is a rectangular drawing.

Proof of Theorem 2. In order to prove Theorem 2, we proceed as follows. Let G be a $K_{1,1,3}$-free series-parallel graph and let T be the SPQ-tree of G rooted at a Q-node ρ with terminals s and t, whose unique child τ is an S-node. Observe that such a Q-node always exists, since G is simple, and that node τ is either of `Type A` or of `Type C`, since G is $K_{1,1,3}$-tree. We perform a bottom-up traversal in T to construct one or two valid drawings of G_μ, for each S-node $\mu \in T$. Namely, we compute:

- an L-shape drawing, if μ is of `Type A` (Lemma 4),
- a pipe drawing, if μ is of `Type B` (Lemma 5), and
- both a pipe drawing and a rectangular drawing, if μ is of `Type C` (Lemma 6).

Thus, when node τ is considered, we can compute either an L-shape drawing of G_τ, if τ is of `Type A`, or a rectangular drawing of G_τ, if τ is of `Type C`. Further, both such valid drawings Γ_τ of G_τ can be easily turned into a square-contact representation Γ_ρ of $G = G_\tau \cup st$, by performing a $\overset{\swarrow}{t}$-scaling and an $\overset{\swarrow}{s}$-scaling of Γ_τ in such a way that the right side of $S(t)$ and the left side of $S(s)$ touch; refer to Fig. 3. This is possible since both in an L-shape drawing and in a rectangular drawing of G_τ all the squares of G^-_τ whose left side (bottom side) is collinear with the right side of $S(t)$ (with the top side of $S(s)$) are adjacent to $S(t)$ (to $S(s)$).

Let μ be an S-node and let $\mu_1, \dots, \mu_k$ be the children of μ in T. If each child μ_i of μ is a Q-node, then node μ is of `Type A`, if $\ell(\mu) > 2$, and it is of `Type C`, otherwise. It is not difficult to see that, in the former case, G_μ admits an L-shape drawing and that, in the latter case, G_μ admits both a pipe drawing and a rectangular drawing. In the remainder of the section, we consider the case in which μ has both Q-node and P-node children.

We first show how to construct special square-contact representations of G_μ, that we call *canonical drawings*, for any P-node μ in T, assuming that valid drawings have been computed for each S-node child of μ. We distinguish five possible canonical drawings, depending on

1. the number and type of the S-node children of μ and
2. the presence of edge st.

Each canonical drawing has three variants: `vertical` (V), `horizontal` (H), and `diagonal` (D). We name such canonical representations *XY drawings*, where $X \in \{V, H, D\}$ denotes the variant of the representation and $Y = 1$, if $st \in E(G_\mu)$, and $Y = 0$, otherwise. Canonical drawings share the following main property (which, in fact, also holds for valid drawings).

▶ **Property 1.** *Let Γ_μ be a valid drawing or a canonical drawing of G_μ. Then, for each vertex v in $V(G^-_\mu)$, it holds that $vs \in E(G_\mu)$ ($vt \in E(G_\mu)$) if:*

1. *$S(v)$ has a side that is collinear with a side of $S(s)$ (of $S(t)$) in Γ_μ and*
2. *$S(v)$ is separated from $S(s)$ (from $S(t)$) in Γ_μ by the line passing through such a side.*

Property 1 allows us to modify canonical and valid drawings by appropriate $\overset{\circ}{s}$-scaling and $\overset{\circ}{t}$-scaling transformations, with $\circ \in \{\nwarrow, \nearrow, \searrow, \swarrow\}$, preserving adjacencies between vertices in G_μ.

First, consider a P-node μ in T with terminals s and t such that $st \notin E(G_\mu)$ and let $\mu_1, \dots, \mu_k$ be the S-node children of μ. We say that a square-contact representation Γ_μ of G_μ is an *H0 drawing* or a *V0 drawing*, if it satisfies the following conditions (in addition to Property 1); refer to Fig. 4.

H0 drawing $S(t)$ lies to the left of $S(s)$, the bottom side of $S(s)$ lies below the bottom side of $S(t)$, and the drawing of G^-_μ in Γ_μ lies to the right of $S(t)$, below the top side of $S(t)$, above the bottom side of $S(s)$, and to the left of the right side of $S(s)$.

V0 drawing $S(t)$ lies above $S(s)$, the left side of $S(s)$ lies to the right of the left side of $S(t)$, and the drawing of G^-_μ in Γ_μ lies above $S(s)$, to the right of the left side of $S(s)$, below the top side of $S(t)$, and to the left of the right side of $S(s)$.

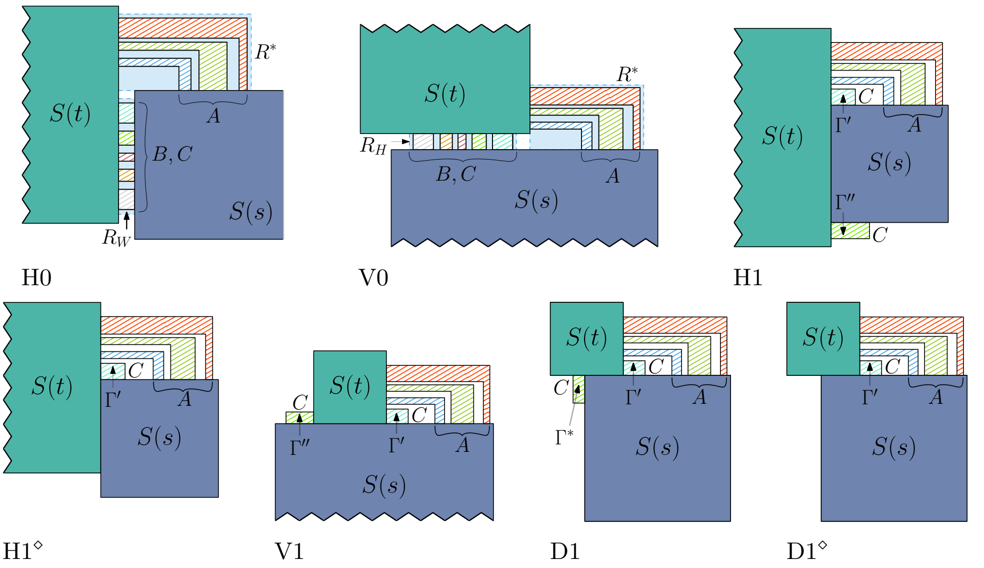


Figure 4 Canonical drawings of a P-node μ. The striped regions correspond to L-shapes, horizontal pipes, and rectangles enclosing the square-contact representations of graphs $G^-_{\mu_i}$, for each S-node child μ_i of μ. Labels A, B, and C indicate the type of each S-node.

Now, consider a P-node μ in T with terminals s and t such that $st \in E(G_\mu)$ and let $\mu_1, \dots, \mu_k$ be the S-node children of μ. We say that a square-contact representation Γ_μ of G_μ is an *H1 drawing*, an *H1$^\diamond$ drawing*, a *V1 drawing*, a *D1 drawing*, or a *D1$^\diamond$ drawing*, if it satisfies the following conditions (in addition to Property 1); refer to Fig. 4.

H1 drawing $S(t)$ lies to the left of $S(s)$, the bottom side of $S(s)$ lies above the bottom side of $S(t)$, and the drawing of G^-_μ in Γ_μ lies to the right of $S(t)$, below the top side of $S(t)$, above the bottom side of $S(t)$, and to the left of the right side of $S(s)$.

H1$^\diamond$ drawing $S(t)$ lies to the left of $S(s)$, the bottom side of $S(s)$ lies below the bottom side of $S(t)$, and the drawing of G^-_μ in Γ_μ lies to the right of $S(t)$, below the top side of $S(t)$, above the top side of $S(s)$, and to the left of the right side of $S(s)$.

V1 drawing $S(t)$ lies above $S(s)$ and the drawing of G^-_μ in Γ_μ lies above $S(s)$, below the top side of $S(t)$, to the right of the left side of $S(s)$, and to the left of the right side of $S(s)$.

D1 drawing $S(t)$ lies above $S(s)$ and the left side of $S(t)$ lies to the left of the left side of $S(s)$, and the drawing of G^-_μ in Γ_μ lies to the right of the left side of $S(t)$, below the top side of $S(t)$, above the bottom side of $S(s)$, and to the left of the right side of $S(s)$.

D1$^\diamond$ drawing Γ_μ is a D1 drawing of G_μ in which the drawing of G^-_μ lies to the right of $S(t)$.

We now present two lemmata for the possible canonical drawings of each P-node μ in T. Recall that, by Observation 1, we can assume that μ is not a forbidden P-node. Let $\mu_1, \dots, \mu_k$ be the S-node children of μ. The general strategy in the proofs of both lemmata consists of

1. computing appropriate valid drawings $\Gamma_{\mu_1}, \dots, \Gamma_{\mu_k}$ for the pertinent graphs $G_{\mu_1}, \dots, G_{\mu_k}$ of $\mu_1, \dots, \mu_k$, respectively,
2. modifying the square-contact representation of $G^-_{\mu_i}$ contained in Γ_{μ_i}, for $i = 1, \dots, k$, by means of affine transformations, so that representations derived from S-nodes of the same type lie in the interior of the same polygon, and finally
3. composing the resulting drawings into a canonical drawing of G_μ. Refer to [5] for details.

We first consider the case in which μ does not contain an edge between its terminals. In this case, by Lemmata 4, 5, and 6, we can assume that Γ_{μ_i} is an L-shape drawing, if μ_i is of `Type A`, and a pipe drawing, if μ_i is of `Type B` or of `Type C`, for $i = 1, \dots, k$.

▶ **Lemma 2.** *Let μ be a P-node in T with terminals s and t such that $st \notin E(G_\mu)$. Then, graph G_μ admits an H0 drawing and a V0 drawing.*

Then, we consider the case in which μ contains an edge between its terminals. Recall that, by Observation 2, node μ has no child of `Type B` and at most two children of `Type C`. In particular, node μ has two children of `Type C`, if it is bad, and one child of `Type C`, if it is almost bad. In this case, by Lemmata 4 and 6, we can assume that Γ_{μ_i} is an L-shape drawing, if μ_i is of `Type A`, and a rectangular drawing, if μ_i is of `Type C`, for $i = 1, \dots, k$.

▶ **Lemma 3.** *Let μ be a P-node in T with terminals s and t such that $st \in E(G_\mu)$. Then, graph G_μ admits*

- *an H1 drawing, a V1 drawing, and a D1 drawing, if μ is bad, or*
- *an H1$^\diamond$ drawing and a D1$^\diamond$ drawing, if μ is good or almost bad.*

We finally turn our attention to the valid drawings of the S-nodes in T. Let μ be an S-node in T and let $\mu_1, \dots, \mu_k$ be the children of μ (where the virtual edge e_i, corresponding to node μ_i, precedes the virtual edge e_{i+1}, corresponding to node μ_{i+1}, from t to s in $skel_\mu$). The next three lemmata immediately imply Theorem 2. To simplify their proofs, we assume that each child of μ is a P-node. In fact, the case in which a child of μ is a Q-node can be treated analogously to that of a P-node containing an edge between its terminals. The general strategy in the proofs of all three lemmata consists of

1. computing appropriate canonical drawings $\Gamma_{\mu_1}, \dots, \Gamma_{\mu_k}$ for the pertinent graphs $G_{\mu_1}, \dots, G_{\mu_k}$ of $\mu_1, \dots, \mu_k$, respectively,
2. modifying these drawings, by means of affine transformations, so that the squares corresponding to terminals shared by different children of μ can be identified without introducing any overlapping between squares corresponding to internal vertices of G_{μ_i} and G_{μ_j}, with $i \neq j$, and finally
3. composing the resulting drawings into a valid drawing of G_μ.

▶ **Lemma 4.** *If μ is an S-node of `Type A`, then G_μ admits an L-shape drawing.*

Proof. We first describe how to select a valid drawing of Γ_{μ_i} of G_{μ_i}, for $i = 1, \dots, k$, based on whether (i) $\ell(\mu) > 2$ or (ii) $\ell(\mu) = 2$. Recall that, if $\ell(\mu) = 2$, then at least one child of μ does not contain an edge between its terminals, say μ_1 (the case in which $s_1t_1 \in E(G_{\mu_1})$ and $s_2t_2 \notin E(G_{\mu_2})$ is analogous).

(i) By Lemma 2 and Lemma 3, we can construct a drawing Γ_{μ_i}, for each μ_i, such that:

1. Γ_{μ_1} is an H0 drawing, if $s_1t_1 \notin E(G_{\mu_1})$, and Γ_{μ_1} is an H1 drawing (H1$^\diamond$ drawing), if μ_1 is bad (if μ_1 is good or almost bad);
2. Γ_{μ_2} is a V0 drawing, if $s_2t_2 \notin E(G_{\mu_2})$, and Γ_{μ_2} is a D1 drawing (D1$^\diamond$ drawing), if μ_2 is bad (if μ_2 is good or almost bad); and
3. Γ_{μ_i} is a V0 drawing, if $s_it_i \notin E(G_{\mu_i})$, and Γ_{μ_i} is a V1 drawing (D1$^\diamond$ drawing), if μ_i is bad (if μ_i is good or almost bad), for every $i > 2$.

(ii) By Lemma 2 and Lemma 3, we can construct an H0 drawing Γ_{μ_1} of G_{μ_1} and a V1 drawing (D1$^\diamond$ drawing) Γ_{μ_2} of G_{μ_2}, if μ_2 is bad (if μ_2 is good or almost bad).

We show how to compose all such drawings into an L-shape drawing Γ_μ of G_μ as follows. Refer to Fig. 5(a) for an example of how to compose drawings Γ_{μ_i}, with $i = 1, \dots, k$, in case (i) and to Fig. 5(b) for an example of how to compose drawings Γ_{μ_1} and Γ_{μ_2} in case (ii). First,

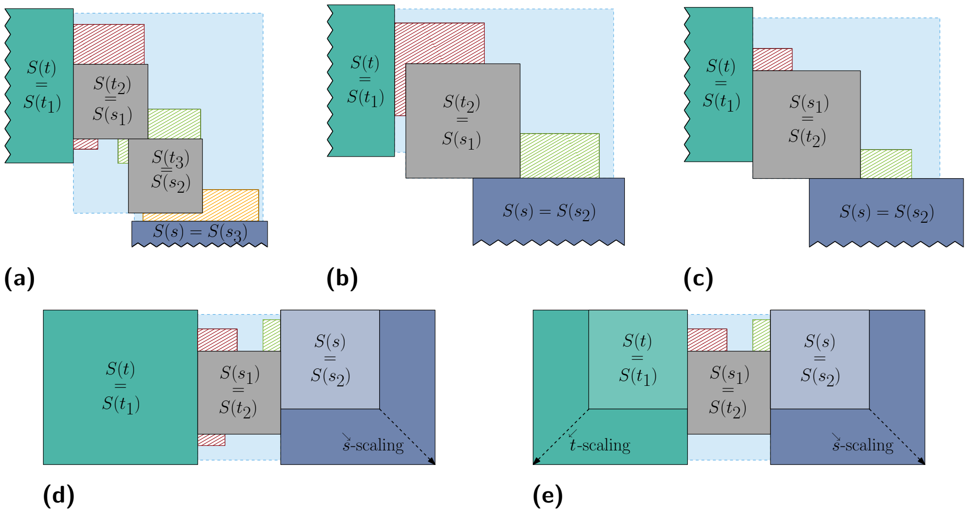


Figure 5 Illustrations for the proofs of Lemmata 4, 5, and 6. Striped polygons of the same color enclose different parts of the drawing of each graph $G^{-}_{\mu_i}$ (contained in the canonical drawing Γ_{μ_i} of G_{μ_i}). (a) An H1 drawing of G_{μ_1}, a D1 drawing of G_{μ_2}, and a D1$^\diamond$ drawing of G_{μ_3} are combined into an L-shape drawing. (b) An H0 drawing of G_{μ_1} and a D1$^\diamond$ drawing of G_{μ_2} are combined into an L-shape drawing. (c) An H1$^\diamond$ drawing of G_{μ_1} and a D1$^\diamond$ drawing of G_{μ_2} are combined into a rectangular drawing. (d) An H1 drawing of G_{μ_1} and a D1$^\diamond$ drawing of G_{μ_2} are combined into a pipe drawing. (e) An H1$^\diamond$ drawing of G_{μ_1} and a D1$^\diamond$ drawing of G_{μ_2} are combined into a pipe drawing.

we scale $S(s_i)$ and $S(t_i)$ in Γ_{μ_i} so that the bounding box of the drawing of each connected component of $G_{\mu_i} - \{s_i, t_i\}$ in Γ_{μ_i}, for $i = 1, \ldots, k$, becomes arbitrarily small with respect to the drawing of $S(s_i)$ and $S(t_i)$. This avoids overlapping between internal vertices of G_{μ_i} and G_{μ_j}, with $i \neq j$, in the next phases of the construction. Then, we scale and translate each drawing Γ_{μ_i} so that $S(t_{i+1}) = S(s_i)$, with $i < k$. It is easy to see that, by the choice of the canonical drawings of each G_{μ_i}, there exists a rectangular region in Γ_μ whose interior does not intersect any square representing a vertex in G^{-}_{μ} and whose lower-left corner lies at the intersection point between the vertical line passing through the right side of $S(t)$ and the horizontal line passing through the top side of $S(s)$ in Γ_μ. ◀

The proof of the next two lemmata also exploits rotations of drawings Γ_{μ_i} and can be carried out in a fashion similar to the proof of Lemma 4. Refer to [5] for details.

▶ **Lemma 5.** *If μ is an S-node of* `Type B`, *then G_μ admits a pipe drawing.*

▶ **Lemma 6.** *If μ is an S-node of* `Type C`, *then G_μ admits a pipe and a rectangular drawing.*

4 Triconnected Simply-Nested Graphs

In this section, we devote our attention to 3-connected simply-nested graphs.

A cycle-tree with a single edge removed from the outer cycle is a *path-tree* (to avoid special cases, we allow the outer cycle of the cycle-tree to be a 2-gon). In path-trees, we refer to vertices in the tree as *tree vertices* and vertices in the external path as *path vertices*. A tree vertex can *see* a path vertex if they share a face in the original cycle-tree. Define an *almost-triconnected path-tree with root ρ, leftmost path vertex ℓ, and rightmost path vertex r* to be a path-tree containing in one of its faces a tree vertex ρ and path vertices ℓ and r such that if the edges $\rho\ell$, ρr, and ℓr were added, the resulting graph would be a 3-connected cycle-tree.

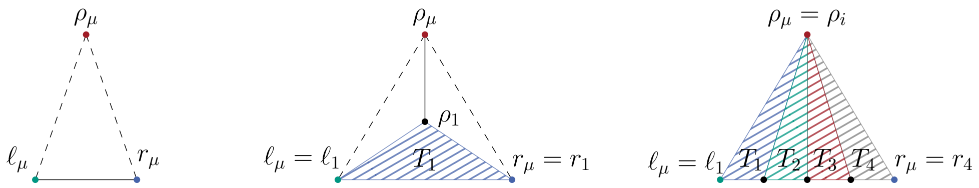


Figure 6 Path-trees associated with a Q-node (left), an S-node (middle), and a P-node (right). Dashed edges may or may not exist. Striped triangles represent smaller path-trees T_i with root ρ_i.

SPQ-decomposition of path-trees. We now describe a recursive decomposition for almost-triconnected path-trees. We call this an SPQ-decomposition, because it bears a striking similarity to the SPQ-decomposition of series-parallel graphs. Let G be a 3-connected cycle-tree, let ℓr be an edge incident to the outer cycle of G, and let ρ be a tree vertex incident to the internal face of G edge ℓr is incident to. Also, let $G' = G - \ell r$ be the almost-triconnected path-tree obtained from G by removing edge ℓr. Graph G' defines a rooted decomposition tree T whose nodes are of three different kinds: *S-*, *P-*, and *Q-nodes*. Each node μ of T is associated with a path-tree G_μ with root ρ_μ, leftmost path vertex ℓ_μ, and rightmost path vertex r_μ obtained—except the Q-nodes—from smaller path-trees T_i with root ρ_i, leftmost path vertex ℓ_i, and rightmost path vertex r_i, for $i = 1, \dots, k$, as follows.

- A *Q-node* μ is associated with a path-tree G_μ with three vertices: one tree vertex ρ_μ and two path vertices ℓ_μ and r_μ. The tree vertex ρ_μ is the root of G_μ, while path vertices ℓ_μ and r_μ are the leftmost and the rightmost path vertex of G_μ, respectively. Edge $\ell_\mu r_\mu$ will always exist, but edges $\rho_\mu \ell_\mu$ and $\rho_\mu r_\mu$ may or may not exist; see Fig. 6(left).
- An *S-node* μ is associated with a path-tree G_μ obtained from path-tree T_1 by adding a new root ρ_μ connected to ρ_1. Also, $\ell_\mu = \ell_1$ and $r_\mu = r_1$ are the leftmost and the rightmost path vertex of G_μ, respectively. Edges $\rho_\mu \ell_\mu$ and $\rho_\mu r_\mu$ may or may not exist; see Fig. 6(midde).
- A *P-node* μ is associated with a path-tree G_μ obtained from path-trees T_i by merging $T_1, T_2, \dots, T_k$ from left to right as follows. First, roots ρ_i are identified into a new root ρ_μ. Then, the rightmost path vertex r_i of T_i and the leftmost path vertex ℓ_{i+1} of T_{i+1} are identified, for $i = 1, \dots, k-1$. Path vertices $\ell_\mu = \ell_1$ and $r_\mu = r_k$ are the leftmost and the rightmost path vertex of G_μ, respectively; see Fig. 6(right).

We have the following lemma.

▶ **Lemma 7.** *Any almost-triconnected path-tree admits an SPQ-decomposition.*

In [5] we show how to construct a square-contact representation of any almost-triconnected path-tree G without separating triangles and whose outer face is not a triangle by inductively maintaining the invariant depicted in Fig. 7 for the S- and P-nodes of an SPQ-decomposition of G. We formalize this result in the next lemma.

▶ **Lemma 8.** *Any almost-triconnected path-tree G without separating triangles and whose outer face is not a triangle admits a square-contact representation.*

To construct a square-contact representation for a 3-connected cycle-tree, it is natural to remove an edge in the outer cycle to obtain a path-tree, use Lemma 8 to construct a square-contact representation, and then attempt to reintroduce a contact for the removed edge. However, because Lemma 8 places the leftmost and rightmost path vertices on the left and right side of the drawing, it is unclear how to add a contact between them. Instead,

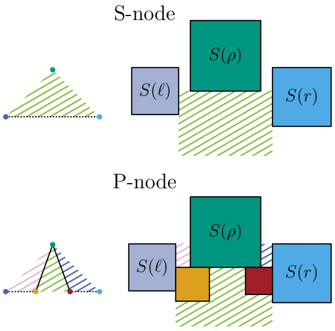


Figure 7 Invariants for S- and P-nodes with more than two path vertices.

we split the cycle-tree into two overlapping almost-triconnected path-trees, obtain their square-contact representations by Lemma 7, and overlay them to form a square-contact representation for the entire cycle-tree.

▶ **Theorem 3.** *Any 3-connected cycle-tree G without separating triangles and whose outer face is not a triangle admits a square-contact representation.*

As Halin graphs are 3-connected cycle-trees without separating triangles and have, except for K_4, a non-triangular outer face, we have the following.

▶ **Corollary 4.** *Any Halin graph $G \not\simeq K_4$ admits a square-contact representation.*

Next, we investigate square-contact representations of 2-outerplanar simply-nested graphs that are not cycle-trees (Theorem 5) and 3-outerplanar simply nested graphs (Theorem 6).

▶ **Theorem 5.** *There exists a 3-connected 2-outerplanar simply-nested graph that does not admit any proper square-contact representation.*

Proof. Consider the two nested quadrilaterals shown in Fig. 8(left). One of its two quadrilateral faces must be the outer one, giving the embedding shown. In any square-contact representation, the inner polygon surrounded by the squares for the four outer vertices must be a rectangle, as it has only four sides. Each of the four inner squares must touch one of the four corners of this rectangle (the corner made by its two outer neighbors). For the four inner squares to touch the four corners of the rectangle and each other, the only possibility is that the rectangle is a square and each inner square fills one quarter of it, as shown in Fig. 8(middle). However, this representation is improper, as diagonally-opposite inner squares meet at their corners. ◀

▶ **Theorem 6.** *There exists a 3-connected 3-outerplanar simply-nested graph that does not admit any square-contact representation.*

Proof. Consider the graph shown in Fig. 8(right). Its quadrilateral face must be the outer one, giving the embedding shown. As in the proof of Theorem 5, the only possible representation for its two outer quadrilaterals has the four outer squares surrounding a central square region, divided into four quarters representing the four middle vertices, as shown in Fig. 8(middle). However, this representation leaves no room for the inner vertex. ◀

We remark that the graph of Theorem 6 is actually 2-outerplanar simply-nested, but not with its quadrilateral face as the outer face.

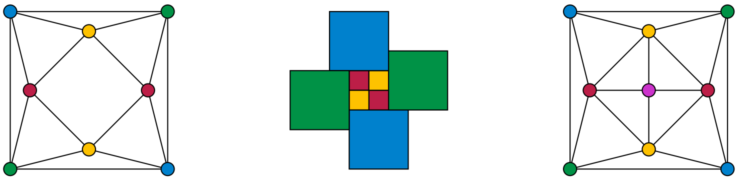


Figure 8 Left: Two nested quadrilaterals form a graph with no proper square-contact representation. Middle: An improper square-contact representation for the same graph. Right: A graph with no square-contact representation, even an improper one.

5 Conclusions

In this paper, we provided simple characterizations for two notable families of planar graphs that admit proper square-contact representations. Moreover, we introduced a new decomposition for an interesting family of polyhedral graphs that generalize the Halin graphs, i.e., the 3-connected cycle-trees. Finally, we showed that the absence of separating triangles and a non-triangular outer face do not guarantee the existence of weak and proper square-contact representations of 3-outerplanar and 2-outerplanar simply-nested graphs, respectively.

Acknowledgements. We thank Jawaherul M. Alam for useful discussions on this subject.

References

1 Md. Jawaherul Alam, David Eppstein, Michael Kaufmann, Stephen G. Kobourov, Sergey Pupyrev, André Schulz, and Torsten Ueckerdt. Contact graphs of circular arcs. In *WADS '15*, volume 9214 of *LNCS*, pages 1–13. Springer, 2015. doi:10.1007/978-3-319-21840-3_1.

2 Clinton Bowen, Stephane Durocher, Maarten Löffler, Anika Rounds, André Schulz, and Csaba D. Tóth. Realization of simply connected polygonal linkages and recognition of unit disk contact trees. In *GD '15*, volume 9411 of *LNCS*, pages 447–459. Springer, 2015. doi:10.1007/978-3-319-27261-0_37.

3 Steven Chaplick, Stephen G. Kobourov, and Torsten Ueckerdt. Equilateral l-contact graphs. In *WG '13*, volume 8165 of *LNCS*, pages 139–151. Springer, 2013. doi:10.1007/978-3-642-45043-3_13.

4 Robert J. Cimikowski. Finding hamiltonian cycles in certain planar graphs. *Inf. Process. Lett.*, 35(5):249–254, 1990. doi:10.1016/0020-0190(90)90053-Z.

5 Giordano Da Lozzo, William Devanny, David Eppstein, and Timothy Johnson. Square-contact representations of partial 2-trees and triconnected simply-nested graphs. Tech. Report arXiv:1710.00426, Cornell University, 2017. URL: http://arxiv.org/abs/1710.00426.

6 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

7 Stefan Felsner and Mathew C. Francis. Contact representations of planar graphs with cubes. In *SoCG '11*, pages 315–320. ACM, 2011. doi:10.1145/1998196.1998250.

8 Daniel Gonçalves, Benjamin Lévêque, and Alexandre Pinlou. Triangle contact representations and duality. *Discrete Comput. Geom.*, 48(1):239–254, 2012. doi:10.1007/s00454-012-9400-1.

9 H. Harborth. Lösung zu Problem 664A. *Elemente der Mathematik*, 29:14–15, 1974.

10 Petr Hliněný. Contact graphs of line segments are NP-complete. *Discrete Math.*, 235(1-3):95–106, 2001. doi:10.1016/S0012-365X(00)00263-6.

11 Oded Schramm. *Combinatorically Prescribed Packings and Applications to Conformal and Quasiconformal Maps.* PhD thesis, Princeton University, 1990.

12 Oded Schramm. Square tilings with prescribed combinatorics. *Israel J. Math.*, 84(1-2):97–118, 1993. doi:10.1007/BF02761693.

13 Kenneth Stephenson. *Introduction to Circle Packing: The theory of discrete analytic functions.* Cambridge University Press (1), 2005.

Faster DBScan and HDBScan in Low-Dimensional Euclidean Spaces[*]

Mark de Berg[†1], Ade Gunawan[2], and Marcel Roeloffzen[‡3]

1 **Department of Computing Science, TU Eindhoven, Eindhoven, The Netherlands**
`mdberg@win.tue.nl`

2 **Department of Computing Science, TU Eindhoven, Eindhoven, The Netherlands**

3 **National institute of informatics, Tokyo and JST ERATO, Kawarabayashi Large Graph Project, Japan**
`marcel@nii.ac.jp`

Abstract

We present a new algorithm for the widely used density-based clustering method DBSCAN. Our algorithm computes the DBSCAN-clustering in $O(n \log n)$ time in $\mathbb{R}^2$, irrespective of the scale parameter ε, but assuming the second parameter MINPTS is set to a fixed constant, as is the case in practice. We also present an $O(n \log n)$ randomized algorithm for HDBSCAN in the plane – HDBSCAN is a hierarchical version of DBSCAN introduced recently – and we show how to compute an approximate version of HDBSCAN in near-linear time in any fixed dimension.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, H.3.3 Information Search and Retrieval

Keywords and phrases Density-based clustering, hierarchical clustering

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.25

1 Introduction

Clustering is one of the most fundamental tasks in data mining. Due to the wide variety of applications where clustering is important, the clustering problem comes in many variants. These variants differ for example in the dimensionality of the data set D and in the underlying metric, but also in the objective of the clustering. Thus a multitude of clustering algorithms has been developed [21], each with their own strengths and weaknesses. We are interested in *density-based clustering*, where clusters are defined by areas in which the density of the data points is high and clusters are separated from each other by areas of low density.

One of the most popular density-based clustering methods is DBSCAN; the paper by Ester *et al.* [12] on DBSCAN has been cited over 8,800 times, and in 2014 DBSCAN received the test-of-time award from KDD, a leading data-mining conference. DBSCAN has two parameters, ε and MINPTS, that together determine when the density around a point $p \in D$ is high enough for p to be part of a cluster as opposed to being noise; see Section 2 for a precise definition of the DBSCAN clustering. Typically MINPTS is a constant – in the original article [12] it is concluded that MINPTS $= 4$ works well – but finding the right value for ε

[*] A full version of the paper is available at [7], `https://arxiv.org/abs/1702.08607`.

[†] MdB is supported by the Netherlands Organization for Scientific Research under grant 024.002.003.

[‡] MR is supported by JST ERATO Grant Number JPMJER1201, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 25; pp. 25:1–25:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is more difficult. The worst-case running time of the original DBSCAN algorithm is $\Theta(n^2)$. It is often stated that the running time is $O(n \log n)$ for Euclidean spaces when a suitable indexing structure such as an R-tree is used to support the DBSCAN algorithm. While this may be true in certain practical cases, it is not true from a theoretical point of view.

Several variants of DBSCAN algorithm have been proposed, often with the goal to speed up the computation. Some (IDBSCAN [5] and FDBSCAN [16]) do so at the expense of computing a slightly different, and not clearly defined, clustering. Others (GriDBSCAN [17]) compute the same clustering as DBSCAN, but without speeding up the worst-case running time.

A fundamental bottleneck of the original DBSCAN algorithm is that it performs a query with each point $p \in D$ to find $N_\varepsilon(p, D)$, the set of points within distance ε of p. The algorithms uses these points to continue expanding the cluster, hence, range counting would not be sufficient. It follows that $\sum_{p \in D} |N_\varepsilon(p, D)|$ is a lower bound on the running time of the DBSCAN algorithm. In the worst case $\sum_{p \in D} |N_\varepsilon(p, D)| = \Theta(n^2)$, so even with a fast indexing structure the worst-case running time of the original DBSCAN algorithm is $\Omega(n^2)$. (Apart from this, the worst-case query time of R-trees and other standard indexing structures is not logarithmic even if we disregard the time to report points.) In most practical instances the DBSCAN algorithm is much faster than quadratic. The reason is that ε is typically small so that the sets $N_\varepsilon(p, D)$ do not contain many points and the range queries can be answered quickly. However, the fact that the algorithm always explicitly reports the sets $N_\varepsilon(p, D)$ makes the running time sensitive to the choice of ε and the density of the point set D. For example, suppose we have a disk-shaped cluster with a Gaussian distribution around the disk center. Then a suitable value of ε will lead to large sets $N_\varepsilon(p, D)$ for points p near the center of the cluster.

Chen *et al.* [9] overcame the quadratic bottleneck of the standard approach, and designed an algorithm[1] with $O(n^{2-\frac{2}{d+2}} \text{ polylog } n)$ worst-case running time. They also present an approximate algorithm. Note that for $d = 2$ the running time of the exact algorithm is $O(n^{1.5} \text{ polylog } n)$. Chen *et al.* remark that their exact algorithm is mainly of theoretical interest. The natural question is then whether or not it is possible to to compute the DBSCAN clustering in subquadratic time in the worst case, irrespective of the value of ε, with a simple and practical algorithm.

Although DBSCAN is used extensively and performs well in many situations, it has its drawbacks. One is that it produces a flat, non-hierarchial clustering which heavily depends on the choice of the scale parameter ε. Ankerst *et al.* [3] therefore introduced OPTICS, which can be seen as a hierarchical version of DBSCAN. Recently Campello *et al.* [8] proposed an improved density-based hierarchical clustering method – similar to OPTICS but cleaner – together with a cluster-stability measure that can be used to automatically extract relevant clusters. The new method, called HDBSCAN, only needs the parameter MinPts, which is much easier to choose than ε. Campello *et al.* used MinPts=4 in all their experiments. While HDBSCAN is very powerful, the algorithm to compute the HDBSCAN hierarchy runs in quadratic time; not only in the worst-case, but actually also in the best-case. There have been only few papers dealing with speeding up HDBSCAN or its predecessor OPTICS. A notable recent exception is POPTICS [20], a parallel algorithm that computes a similar, but not the same, hierarchy as OPTICS. We do not know of any algorithm that computes the HDBSCAN or OPTICS hierarchy in subquadratic time. Thus the second question we study is: is it possible to compute the HDBSCAN hierarchy in subquadratic time.

[1] As described, the algorithm actually computes a variation of the DBSCAN clustering, but it is easily adapted to compute the true DBSCAN clustering.

Our results. We present an $O(n \log n)$ algorithm to compute the DBSCAN clustering for a set D of n points in the plane, irrespective of the setting of the parameter ε used to define the DBSCAN clustering. Here, and in our other results, we assume that the parameter MinPts is a fixed constant. As mentioned this is the case in practice, where one typically uses MinPts $= 4$. We remark that our algorithm is not only fast in theory, but a slightly simplified version is also competitive in practice and much less sensitive to the choice of ε than the original DBSCAN algorithm. In this submission we focus on our theoretical contributions. Experimental results can be found in the full version [7].

We also present a new algorithm for planar HDBSCAN: we show how to compute the HDBSCAN hierarchy in $\mathbb{R}^2$ in $O(n \log n)$ expected time, thus obtaining the first subquadratic algorithm for the problem.

In higher dimensions exact algorithms are much slower and so we consider approximations instead. We extend the concept of an approximate DBSCAN clustering as defined by Chen *et al.* [9] and by Gan and Tao [13] (see below) to the hierarchical version. We thus obtain δ-approximate HDBSCAN, an approximate version of the HDBSCAN hierarchy of Campello *et al.* [8], where the parameter δ specifies the accuracy of the approximation. Intuitively, a δ-approximate HDBSCAN hierarchy has the same clusters as the standard HDBSCAN hierarchy at any level ε, except that clusters at distance $(1-\delta)\cdot\varepsilon$ from each other may be merged, see Section 5 for a precise definition. We show that a δ-approximate HDBSCAN hierarchy in $\mathbb{R}^d$ can be computed in $O((n/\delta^{(d-1)/2}) \log^{d-1} n)$ time.

Further related work. Our paper is the conference paper corresponding to the so far unpublished master's thesis of the second author [15], which contained the results on DBSCAN, extended with results on HDBSCAN. In the meantime, Gan and Tao [13] published a paper in which they extend the work from the master's thesis to $\mathbb{R}^d$, resulting in an algorithm for DBSCAN with a running time of $O(n^{2-\frac{2}{\lceil d/2 \rceil+1}+\gamma})$; we briefly comment on how this is done at the end of Section 3. Gan and Tao also prove that computing the DBSCAN clustering in $\mathbb{R}^d$ for $d \geqslant 3$ is at least as hard as the so-called unit-spherical emptiness problem, which is believed to require $\Omega(n^{4/3})$ time [11]. Finally, Gan and Tao show that a δ-approximate DBSCAN clustering can be computed in $O(n/\delta^{d-1})$ expected time, using a modified version of the exact algorithm. Their approximate clustering is the same as the approximate clustering defined by Chen *et al.* [9], who already showed how to compute it in $O(n \log n + n/\delta^{d-1})$ time deterministically. (Gan and Tao were unaware of the paper by Chen *et al.*.) As we remark in Section 5 our algorithm can also be used to obtain a deterministic algorithm with $O(n \log n + n/\delta^{d/3+c})$ running time for some constant c.

2 Preliminaries on DBScan and DBScan*

Let D be a set of points in $\mathbb{R}^d$. DBSCAN distinguishes three types of points: *core points* in the "interior" of a cluster, *border points* on the boundary of a cluster, and *noise points* not in any cluster. The distinction is based on two global parameters, ε and MinPts. Define $N_\varepsilon(p, D) := \{q \in D : |pq| \leqslant \varepsilon\}$ to be the *neighborhood* of a point p, where $|pq|$ denotes the (Euclidean) distance between p and q; the point p itself is included in $N_\varepsilon(p, D)$. A point $p \in D$ is a *core point* if $|N_\varepsilon(p, D)| \geqslant$ MinPts, and a non-core point q in the neighborhood of a core point is a *border point*. We denote the set of core points by D_{core}, and the set of border points by D_{border}. The remaining points are *noise*. In DBSCAN* [8] border points are not part of a cluster but are considered noise.

Ester *et al.* [12] define the DBSCAN clusters based on the concept of density-reachability.

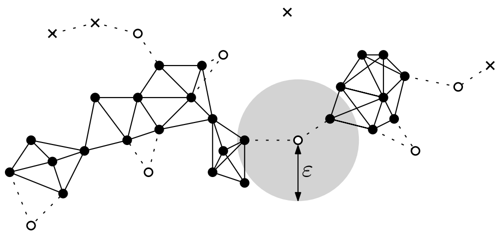


Figure 1 A neighborhood graph with MinPts $= 4$ and ε as indicated. Solid disks are core points, open circles are border points, and crosses are noise. Edges between core points are solid, other edges are dotted. The solid disks and edges form the core graph.

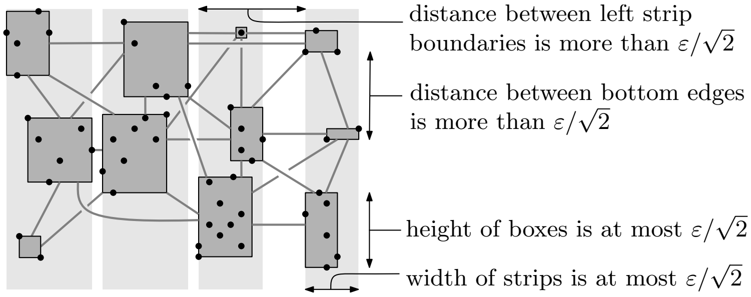

distance between left strip boundaries is more than $\varepsilon/\sqrt{2}$
distance between bottom edges is more than $\varepsilon/\sqrt{2}$
height of boxes is at most $\varepsilon/\sqrt{2}$
width of strips is at most $\varepsilon/\sqrt{2}$

Figure 2 Example of a box graph.

Equivalently, we can define the clusters as the connected components of a certain graph.

To this end, define the *neighborhood graph* $\mathcal{G}(D, E)$ as the undirected graph with node set D and edges connecting pairs of points within distance ε; see Fig. 1. Note that a point $p \in D$ is a core point if and only if its degree in $\mathcal{G}$ is at least MinPts $- 1$, since then its neighborhood contains at least MinPts points, including p itself. Now consider the subgraph $\mathcal{G}_{\text{core}}(D_{\text{core}}, E_{\text{core}})$ induced by the core points, that is, $\mathcal{G}_{\text{core}}$ is the graph whose nodes are the core points and whose edges connect two core points when they are within distance ε from each other. We call $\mathcal{G}_{\text{core}}$ the *core graph*. The connected components of $\mathcal{G}_{\text{core}}$ are the clusters in DBSCAN*. The clusters in DBSCAN are the same, except that they also contain border points. Formally, a border point q belongs to a cluster C if q has an edge in $\mathcal{G}$ to a core point $p \in C$. Thus a border point can belong to multiple clusters. The original DBSCAN algorithm construct clusters one by one and assigns a border point p to the first cluster that finds p; we assign border points to the cluster of their nearest core point.

3 A fast algorithm for DBScan

The original DBSCAN algorithm reports, while generating and exploring the clusters, for each point $p \in D$ all its neighbors. In other words, it spends time on every edge in the neighborhood graph. Our new algorithm avoids this by working with a smaller graph, the *box graph* $\mathcal{G}_{\text{box}}$. Its nodes are disjoint rectangular boxes with a diameter of at most ε that together contain all the points in D, and its edges connect pairs of boxes within distance ε; see Fig. 2.

The boxes are generated such that (i) any two points in the same box are in each other's neighborhood, and (ii) the degree of any node in the box graph is $O(1)$. Property (i) allows us to immediately classify all points in a box as core points when it contains at least MinPts points, and property (ii) allows us to quickly retrieve the neighbors of any given point in a box. Next we describe the algorithm, which consists of four easy steps, in detail.

Step 1: Compute the box graph $\mathcal{G}_{\text{box}}$. To compute $\mathcal{G}_{\text{box}}$, we first construct a collection of vertical *strips* that together cover all the points. Let $p_1, \ldots, p_n$ be the points in D sorted by x-coordinate, with ties broken arbitrarily. The first strip has p_1 on its left boundary. We continue from left to right, adding points to the first strip as we go, until we encounter a point p_i whose distance to the left strip boundary is more than $\varepsilon/\sqrt{2}$. We then start a new strip with p_i on its left boundary, and we add points to that strip until we encounter a point whose distance to the left strip boundary is more than $\varepsilon/\sqrt{2}$, and so on, until we handled all the points. Constructing the strips takes $O(n)$ time, after sorting the points by x-coordinate.

Within each strip we perform a similar procedure, going over the points within the strip in order of increasing y-coordinate and creating boxes instead of strips. Thus the first box in the strip has the lowest point on its bottom edge, and we keep adding points to this box (enlarging it so that the new point fits, ensuring a tight bounding box) until we encounter a point whose vertical distance to the bottom edge is more than $\varepsilon/\sqrt{2}$. We then start a new box, and so on, until we handled all points in the strip. If the number of points in the j-th strip is n_j, then the time needed to handle all the strips is $\sum_j O(n_j \log n_j) = O(n \log n)$.

Let m be the number of strips and $\mathcal{B}_j$ the set of boxes in the j-th strip. We sometimes refer to a set $\mathcal{B}_j$ as a strip, even though formally $\mathcal{B}_j$ is a set of boxes. Let $\mathcal{B} := \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_m$. The nodes of the box graph $\mathcal{G}_{\text{box}}$ are the boxes in $\mathcal{B}$ and there is an edge (b, b') when $\text{dist}(b, b') \leqslant \varepsilon$, where $\text{dist}(b, b')$ denote the minimum distance between b and b'. Two boxes b, b' are *neighbors* when they are connected by an edge. Let $\mathcal{N}_\varepsilon(b, \mathcal{B})$ be the neighbors of b.

▶ **Lemma 1.** *$\mathcal{G}_{\text{box}}$ has at most n nodes, each having $O(1)$ neighbors.*

The lemma above follows from the fact that any box $b \in \mathcal{B}_j$ can have neighbors only in $\mathcal{B}_{j-2}$, $\mathcal{B}_{j-1}$, $\mathcal{B}_j$, $\mathcal{B}_{j+1}$, or $\mathcal{B}_{j+2}$, and within any of these five strips, b can have at most five neighbors. (A more precise proof giving a bound of 22 neighbors can be found in the full version [7].) This also gives us an easy way to compute the edge set E_{box} of the box graph, because the edges between boxes in strips $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ with $|j - j'| \leqslant 2$ can be computed in $O(|\mathcal{B}_j| + |\mathcal{B}_{j'}|)$ time in total by scanning the boxes in $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ in a coordinated manner. The total time to compute all edges of the box graph is thus

$$O\left(\sum_{j=1}^{m} \sum_{j'=\max(j-2,1)}^{\min(j+2,m)} (|\mathcal{B}_j| + |\mathcal{B}_{j'}|)\right) = O\left(\sum_{j=1}^{m} |\mathcal{B}_j|\right) = O(n).$$

Adding the time to construct the strips and boxes, we see that Step 1 takes $O(n \log n)$ time and we obtain the following lemma.

▶ **Lemma 2.** *The box graph $\mathcal{G}_{\text{box}}(\mathcal{B}, E_{\text{box}})$ can be computed in $O(n \log n)$ time.*

An alternative for Step 1. An alternative approach is to define the boxes as the non-empty cells in a grid whose cells have height and width $\varepsilon/\sqrt{2}$. If we store the boxes in a hash-table based on the coordinates of their lower left corners, then finding the neighbors of a box b can be done by checking each potential neighbor cell for existence in the hash-table – we do not need to store the box graph explicitly. Creating the boxes (with their corresponding point sets) can be done in $O(n)$ time if the floor function can be computed in $O(1)$ time.

Step 2: Find the core points. The graph $\mathcal{G}_{\text{box}}$ allows us to determine the core points in a simple and efficient manner. The key observation is that the maximum distance between any two points in the same box is at most ε. Hence, if a box contains more than MinPts points, then all of them are core points. The following algorithm suffices to find the core points.

For a box $b \in \mathcal{B}$, let $D(b) := D \cap b$ be the set of point inside b, and let $n_b := |D(b)|$. If $n_b \geqslant \textsc{MinPts}$ then label all points in b as core points. Otherwise, for each point $p \in D(b)$, count the number of points q in neighboring boxes of b for which $|pq| \leqslant \varepsilon$. If this number is at least $\textsc{MinPts} - n_b$, then label p as core point. The counting is done brute-force, by checking all points in neighboring boxes. Hence, this takes $O(\sum_{b' \in \mathcal{N}_\varepsilon(b,\mathcal{B})} n_{b'})$ time for each point $p \in b$.

▶ **Lemma 3.** *Given $\mathcal{G}_{\text{box}}$, we can find all core points in D in $O(n)$ time.*

Proof. The total time spent to handle boxes b with $n_b \geqslant \textsc{MinPts}$ is clearly $O(n)$. The time needed to handle a box b with $n_b < \textsc{MinPts}$ is

$$O\left(n_b \cdot \sum_{b' \in \mathcal{N}_\varepsilon(b,\mathcal{B})} n_{b'}\right) = O\left(\textsc{MinPts} \cdot \sum_{b' \in \mathcal{N}_\varepsilon(b,\mathcal{B})} n_{b'}\right).$$

Now charge $O(\textsc{MinPts}) = O(1)$ time to each point in every $b' \in \mathcal{N}_\varepsilon(b,\mathcal{B})$. Because any box b' is the neighbor of $O(1)$ other boxes by Lemma 1, each point is charged $O(1)$ times. ◀

Step 3: Compute the cluster cores. The *core of a cluster* is the set of core points in that cluster. In Step 3 we assign to each core point a *cluster-id* so that core points in the same cluster have the same cluster-id. Again, this can be done in an efficient manner using $\mathcal{G}_{\text{box}}$. To this end, we first remove certain boxes and edges from $\mathcal{G}_{\text{box}}$ to obtain a reduced box graph $\mathcal{G}^*_{\text{box}}$. More precisely, we keep only the boxes with at least one core point, and we keep only the edges (b, b') for which there are core points $p \in b$, $p' \in b'$ with $|pp'| \leqslant \varepsilon$. Because any two core points in a given box b are connected in $\mathcal{G}_{\text{core}}$, we have the following lemma.

▶ **Lemma 4.** *The connected components in $\mathcal{G}^*_{\text{box}}$ correspond one-to-one to the connected components in the core graph $\mathcal{G}_{\text{core}}$ and, hence, to the* DBSCAN* *clusters.*

Thus the cluster cores can be computed by computing the connected components in $\mathcal{G}^*_{\text{box}}$. The latter can be done in $O(n)$ time using DFS [10]. We then give every core point p a cluster-id corresponding to the connected component of the box b that contains p.

To construct $\mathcal{G}^*_{\text{box}}$, we need to decide for two given boxes b, b' whether there are core points $p \in D(b)$, $p' \in D(b')$ with $|pp'| \leqslant \varepsilon$. For ease of discussion we call the points in $D(b)$ blue and those in $D(b')$ red. It is well known [2] that the bichromatic closest pair defines an edge of the Delaunay triangulation of the points, so it suffices to compute the Delaunay triangulation of $D(b) \cup D(b')$ and find the shortest red-blue edge. If it is at most ε we connect b and b' in $\mathcal{G}^*_{\text{box}}$ and otherwise we do not. This leads to the following lemma.

▶ **Lemma 5.** *Computing the cluster cores can be done in $O(n \log n)$ time.*

Proof. The most time consuming part of the construction of $\mathcal{G}^*_{\text{box}}$ is to determine for each pair of neighboring boxes in $\mathcal{B}$ whether there are core points $p \in b$, $p' \in b'$ with $|pp'| \leqslant \varepsilon$. Let $\mathcal{B}^*$ be the set of boxes containing at least $\textsc{MinPts}$ points. Then the total time spent on the pairs of boxes from $\mathcal{B}^*$ is

$$\sum_{b \in \mathcal{B}^*} \sum_{b' \in \mathcal{N}_\varepsilon(b,\mathcal{B}^*)} O((n_b + n_{b'}) \log(n_b + n_{b'})),$$

which is $O(n \log n)$ because $|\mathcal{N}_\varepsilon(b,\mathcal{B}^*)| = O(1)$ for any box b and $\sum_{b \in \mathcal{B}^*} n_b \leqslant n$. ◀

▶ Remark. In practice, computing the Delaunay triangulation is not necessary. Instead we can use a brute-force algorithm that checks every pair of points in b and b' and stops when a sufficiently close pair is found. The number of points in each box is expected to be small and if it is large one may expect many pairs to have a short distance, hence, testing pairs in random order should find such a pair fairly quickly.

Step 4: Assigning border points to clusters. It remains to decide for non-core points p whether p is a border point or noise. If p is a border point, it has to be assigned to the nearest cluster. Again, a brute-force method suffices: for each box $b \in B$ and each non-core point $p \in b$, we check all points in b and its neighboring boxes to find p's nearest core point, p'. If $|pp'| \leqslant \varepsilon$, then p is a border point in the same cluster as p', otherwise p is noise. We only need to consider boxes b with $n_b <$ MinPts – otherwise all points in b are core points – so the argument from the proof of Lemma 3 shows that this takes $O(n)$ time.

Putting it all together. Steps 1 and 3 take $O(n \log n)$ time and Steps 2 and 4 take $O(n)$ time. We thus obtain the following theorem.

▶ **Theorem 6.** *Let D be a set of n points in $\mathbb{R}^2$, and ε and* MinPts *be given constants. Then we can compute a* dbscan *clustering on D according to ε and* MinPts *for the Euclidean metric in $O(n \log n)$ time.*

▶ Remark (Extension to higher dimensions.). The algorithm just described can easily be extended to $\mathbb{R}^d$ for $d > 2$, as already observed by Gan and Tao [13]. The resulting running time is $O(n^{2-\frac{2}{\lceil d/2 \rceil+1}+\gamma})$.

4 A fast algorithm for HDBScan in the plane

Campello *et al.* [8] introduced hdbscan, a hierarchical version of dbscan* similar to optics [3]. The algorithm described by Campello *et al.* to compute the hdbscan hierarchy runs in quadratic time. We show that in $\mathbb{R}^2$ and under the Euclidean metric, the hdbscan hierarchy can be computed in $O(n \log n)$ time.

Preliminaries on HBScan. Recall that dbscan* is the version of dbscan in which border points are considered noise. The hdbscan hierarchy is a tree structure encoding the clusterings of dbscan* that arise as ε increases from $\varepsilon = 0$ to $\varepsilon = \infty$ for a fixed MinPts. Initially, when $\varepsilon = 0$, all points are noise. As ε increases, three types of events can happen to the dbscan* clustering:

- *Type (i): the status of a point changes.* In this event, a point changes from being noise to being a core point. The value of ε at which this happens for a point p is called the *core distance* of p; we denote it by $d_{\text{core}}(p)$.
- *Type (ii): a new cluster starts.* This event is triggered by a type (i) event, when a point becoming a core point forms a new singleton cluster.
- *Type (iii): two clusters merge.* This event can be triggered by a type (i) event or it can happen when $\varepsilon = |pq|$ for core points p, q from different clusters.

Note that all events happen at values of ε such that $\varepsilon = |pq|$ for some pair of points $p, q \in D$. This process can be modeled as a *dendrogram*: a tree whose leaves correspond to the points in D and whose nodes correspond to clusters arising during the process. This dendrogram, where each node stores the value of ε at which the corresponding cluster was created, is the hdbscan hierarchy. Campello *et al.* compute the hdbscan hierarchy as follows.

For two points $p, q \in D$, define $d_{\mathrm{mr}}(p,q) := \max\left(d_{\mathrm{core}}(p), d_{\mathrm{core}}(q), |pq|\right)$ to be the *mutual reachability distance* of p and q. The *mutual reachability graph* $\mathcal{G}_{\mathrm{mr}}$ is defined as the complete graph with node set D in which each edge (p,q) has weight $d_{\mathrm{mr}}(p,q)$. Campello *et al.* observe that HDBSCAN hierarchy can easily be computed from a minimum spanning tree (MST) on $\mathcal{G}_{\mathrm{mr}}$. (Indeed, the cluster-growing process corresponds to the the computation of an MST on $\mathcal{G}_{\mathrm{mr}}$ using Kruskal's algorithm [10].) Hence, they compute the HDBSCAN hierarchy as follows.

1. Compute the core distances $d_{\mathrm{core}}(p)$ for all points $p \in D$.
2. Compute an MST $\mathcal{T}$ of the mutual reachability graph $\mathcal{G}_{\mathrm{mr}}$.
3. Convert $\mathcal{T}$ into a dendrogram where each internal node stores the value of ε at which the corresponding cluster is formed.

Our planar algorithm. The most time-consuming parts in the algorithm above are Steps 1 and 2; Step 3 takes $O(n)$ time after sorting the edges of $\mathcal{T}$ by weight.

For Step 1 we observe that $d_{\mathrm{core}}(p)$ is the distance of point p to its ℓ-th nearest neighbor for $\ell = \text{MinPts} - 1$. Hence, to compute all core distances it suffices to compute for each point its k nearest neighbors. This can be done in any fixed dimension in $O(n\ell \log n)$ time [22]. Since $\ell = \text{MinPts} - 1 = O(1)$ this implies that Step 1 takes $O(n \log n)$ time.

Step 2 is more difficult to do in subquadratic time. The main problem is that we cannot afford to look at all edges of $\mathcal{G}_{\mathrm{mr}}$ when computing $\mathcal{T}$. To overcome this problem we need the following generalization of Delaunay triangulations, introduced by Gudmundsson *et al.* [14]. Recall that a pair of points $p, q \in D$ forms an edge in the Delaunay triangulation of D if and only if there is a circle with p and q on its boundary and no points from D in its interior [6]. We say that the pair $p, q \in D$ forms a *k-th order Delaunay edge*, or *k-OD edge* for short, if and only if there exists a circle with p and q on its boundary and at most k points from D in its interior [14]. Thus the 0-OD edges are precisely the edges of the Delaunay triangulation. The k-OD edges are useful for us because of the following lemma.

▶ **Lemma 7.** *Let $\overline{\mathcal{G}}_{\mathrm{mr}}$ be the subgraph of $\mathcal{G}_{\mathrm{mr}}$ that contains only the k-OD edges, where $k := \max(\text{MinPts} - 3, 0)$. Then an* MST *of $\overline{\mathcal{G}}_{\mathrm{mr}}$ is also an* MST *of $\mathcal{G}_{\mathrm{mr}}$.*

Proof. Imagine computing an MST $\mathcal{T}$ on $\mathcal{G}_{\mathrm{mr}}$ using Kruskal's algorithm [10]. This algorithm treats the edges (p,q) of $\mathcal{G}_{\mathrm{mr}}$ in order of increasing weight, that is, increasing values of $d_{\mathrm{mr}}(p,q)$. When it processes (p,q) it checks if p and q are already in the same connected component – in our application this component corresponds to a cluster at the current value of ε – and, if not, merges these components. We will argue that whenever we process an edge (p,q) that is not in $\overline{\mathcal{G}}_{\mathrm{mr}}$, that is, an edge that is not a k-OD edge, then p and q are already in the same connected component. Hence, there is no need to process (p,q), which proves that an MST of $\overline{\mathcal{G}}_{\mathrm{mr}}$ is also an MST of $\mathcal{G}_{\mathrm{mr}}$.

Let C_{pq} be the circle such that p and q form a diametrical pair of C, and let $D(C_{pq}) \subset D$ be the set of points lying in the interior of C_{pq}. If $|D(C_{pq})| \leqslant k$, then (p,q) is a k-OD edge, so assume $|D(C_{pq})| \geqslant k+1$. Note that $d_{\mathrm{core}}(r) < |pq|$ for all $r \in D(C_{pq})$. Indeed, since r is an interior point in a disk with diameter $|pq|$, the distance from r to any other point in C_{pq}, including p and q, is smaller than $|pq|$. Hence, for $\varepsilon = |pq|$ we have $|N_\varepsilon(r,D)| \geqslant |D(C_{pq})| + 2 = k+3 \geqslant \text{MinPts}$. Thus all points $r \in C_{pq}$ are core points when we process (p,q). Moreover, for all edges (s,t) with $s,t \in D(C_{pq}) \cup \{p,q\}$ we have $d_{\mathrm{mr}}(s,t) \leqslant |pq|$. Hence, it suffices to prove the following.

▶ **Claim.** *Let C be a circle with two points p, q on its boundary and let $D(C) \subset D$ be the set of points from D in the interior of C. Then there is a path from p to q in $\overline{\mathcal{G}}_{\mathrm{mr}}$ that uses only points in $D(C) \cup \{p,q\}$.*

We prove this claim by induction on $|D(C)|$. If $|D(C)| \leqslant k$ then (p, q) is a k-OD edge itself and we are done. Otherwise, pick any point $r \in D(C)$. Now shrink C, while keeping p in its boundary, until we obtain a circle C_1 that also has r on its boundary. By induction, there is a path from p to r in $\overline{\mathcal{G}}_{\mathrm{mr}}$ that uses only points in $D(C_1) \cup \{p, r\} \subset D(C) \cup \{p, q\}$. A similar argument shows that there is a path from r to q that uses only points in $D(C) \cup \{p, q\}$. This proves the claim and, hence, the lemma. ◀

Gudmundsson *et al.* showed that the number of k-OD edges is $O(n(k+1))$ and that the set of all k-OD edges can be computed in $O(n(k+1)\log n)$ time with a randomized incremental algorithm. Lemma 7 implies that after computing the core distances and the k-OD edges in $O(n \log n)$ time with $k = \max(\text{MinPts} - 3, 0) = O(1)$ we can compute the MST for $\mathcal{G}_{\mathrm{mr}}$ by considering only $O(n)$ edges. Thus computing the MST can be done in $O(n \log n)$ time [10]. Since the rest of the algorithm takes linear time, we obtain the following theorem.

▶ **Theorem 8.** *Let D be a set of n points in $\mathbb{R}^2$ and* MinPts *be a given constant. We can compute the* HDBSCAN *hierarchy on D for the Euclidean metric with a randomized algorithm in $O(n \log n)$ expected time.*

5 Approximate HDBScan

In this section we introduce an approximate version of HDBSCAN which can be computed in near-linear time in any fixed dimension.

Approximate DBScan*. Before we can define approximate HDBSCAN, we need to define approximate DBSCAN*. Our definition of approximate DBSCAN* is essentially the same as the definitions of Chen *et al.* [9] and Gan and Tao [13]. The main difference is that we base our definition on DBSCAN* instead of DBSCAN, which avoids some technical difficulties in the definition.

Let MinPts be a fixed constant. Let $\mathcal{C}_\varepsilon(D)$ denote the set of clusters in the DBSCAN* clustering for a given value of ε. We call a clustering $\mathcal{C}_1$ a *refinement* of a clustering $\mathcal{C}_2$, denoted by $\mathcal{C}_1 \prec \mathcal{C}_2$, when for every cluster $C_1 \in \mathcal{C}_1$ there is a cluster $C_2 \in \mathcal{C}_2$ with $C_1 \subseteq C_2$. Recall that, as ε increases, the DBSCAN* clusters merge or expand and new singleton clusters may appear, but clusters do not shrink or disappear. Hence, if $\varepsilon < \varepsilon'$ then[2] $\mathcal{C}_\varepsilon(D) \prec \mathcal{C}_{\varepsilon'}(D)$. An approximate DBSCAN* clustering is now defined as follows.

▶ **Definition 9.** A δ*-approximate* DBSCAN* *clustering* of a data set D, for given parameters ε and MinPts, and a given error $\delta > 0$, is defined as a clustering $\mathcal{C}^*$ of D into clusters and noise such that $\mathcal{C}_{(1-\delta)\varepsilon}(D) \prec \mathcal{C}^* \prec \mathcal{C}_\varepsilon(D)$.

Thus if we choose δ sufficiently small, then a δ-approximate DBSCAN* clustering is very similar to the exact DBSCAN* clustering for the given parameter values.

▶ Remark. An approximate DBSCAN* clustering can be computed by using the *approximate bichromatic closest pair* algorithm by Arya and Chan [4] as a subroutine in our exact algorithm. The resulting algorithm finds a δ-approximate DBSCAN* clustering in $\mathbb{R}^d$ in $O(n \log n + n/\delta^{d/3+c})$ time. This is similar to the running time of Chen *et al.* [9] and the expected running time of Gan and Tao [13], but it has a better dependency on δ. Note, however, that Gao and Tao are able to avoid the $O(n \log n)$ term. The easy details can be found in the full version [7].

[2] Here it is important that we consider DBSCAN* and not DBSCAN. Indeed, in DBSCAN border points can "flip" between clusters as ε increases, and so we do not necessarily have $\mathcal{C}_\varepsilon(D) \prec \mathcal{C}_{\varepsilon'}(D)$.

Approximate HDBScan. Our definition of an approximate HDBSCAN hierarchy is based on the definition of δ-approximate DBSCAN* clusterings: we say that a hierarchy is a *δ-approximate* HDBSCAN *hierarchy* if, for any value of ε, the clustering extracted from the hierarchy is a δ-approximate DBSCAN* clustering for that value of ε. Next we show how to compute a δ-approximate HDBSCAN hierarchy in $O(n \log n)$ time, in any fixed dimension d.

As in Section 4 we follow the algorithm by Campello *et al.* [8]. Steps 1 and 3 can still be done in $O(n \log n)$ and $O(n)$ time, respectively. We speed up Step 2 of the algorithm by computing an MST on a subgraph of the mutual reachability graph $\mathcal{G}_{\mathrm{mr}}$ rather than on the whole graph. The difference with the exact algorithm of Section 4 is that we will select the edges of the subgraph in a different manner, using ideas from so-called θ-graphs [19].

Let $p \in D$ be a point. We partition $\mathbb{R}^d$ into simplicial cones with apex p and whose angular diameter is θ, where θ will be specified later. (The angular diameter of a cone c with apex p is the maximum angle between any two vectors emanating from p and inside c.) Let Γ_p be the resulting collection of cones and consider a cone $c \in \Gamma_p$. Let $D(c) \subseteq D$ denote the set of points inside c. If a point lies on the boundaries of several cones we can assign it to one of these cones arbitrarily. Pick a half-line ℓ_c with endpoint p that lies inside c. A θ-graph would now be obtained by projecting all points from $D(c)$ orthogonally onto ℓ_c, and adding an edge from p to the point closest to p in this projection, with ties broken arbitrarily. We do the same, except that we add edges to the k closest points for $k := 2 \cdot \textsc{MinPts} - 3$. If c contains fewer than k points, we simply connect p to all points in $D(c)$. Doing this for all the cones $c \in \Gamma_p$ gives us a set E_p of $O(k/\theta^{d-1}) = O(1/\theta^{d-1})$ edges for point p. Let $E(\theta) := \bigcup_{p \in D} E_p$. The set $E(\theta)$ can be computed by making a straightforward adaptation to the algorithm to compute a θ-graph in $\mathbb{R}^d$ [19, Chapter 5], leading to the following result.

▶ **Lemma 10.** *$E(\theta)$ has $O(n/\theta^{d-1})$ edges and can be computed in $O((n/\theta^{d-1}) \log^{d-1} n)$ time.*

The set $E(\theta)$, where θ is chosen such that $\cos\theta \geqslant 1 - \delta$, defines the subgraph $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)$ on which we compute the MST in Step 2. Since $\cos\theta > 1 - \theta^2/2$, we have $\cos\theta \geqslant 1 - \delta$ when $\theta := \sqrt{2\delta}$. Next we show that an MST on $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)$ defines a δ-approximate HDBSCAN clustering.

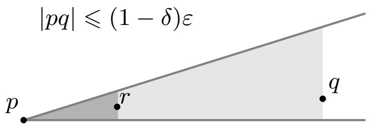
▶ **Lemma 11.** *Let $\mathcal{T}$ be an* MST *of $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)$ and let $\varepsilon > 0$. Let $\mathcal{C}(\mathcal{T}, \varepsilon)$ be the clustering induced by $\mathcal{T}$. Then $\mathcal{C}$ is a δ-approximate* DBSCAN* *clustering for the given ε.*

Proof. For a weighted graph $\mathcal{G}$ and threshold weight τ, let $\mathcal{G}[\tau]$ denote the subgraph obtained by removing all edges of weight greater than τ. In order to show that $\mathcal{C}(\mathcal{T}, \varepsilon) \prec \mathcal{C}_\varepsilon(D)$ we must show that any connected component of $\mathcal{T}[\varepsilon]$ is contained in a connected component of $\mathcal{G}_{\mathrm{mr}}[\varepsilon]$. Since $\mathcal{T}$ is a subgraph of $\mathcal{G}_{\mathrm{mr}}$ this is obviously the case.

Next we prove that $\mathcal{C}_{(1-\delta)\varepsilon}(D) \prec \mathcal{C}(\mathcal{T}, \varepsilon)$. For this we must prove that any connected component of $\mathcal{G}_{\mathrm{mr}}[(1-\delta)\varepsilon]$ is contained in a connected component of $\mathcal{T}[\varepsilon]$. Since $\mathcal{T}$ is an MST of $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)$, the connected components of $\mathcal{T}[\varepsilon]$ are the same as the connected components of $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)[\varepsilon]$. It thus suffices to show the following: for any edge $(p, q) \in \mathcal{G}_{\mathrm{mr}}[(1-\delta)\varepsilon]$, there is a path from p to q in $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)[\varepsilon]$. We show this by induction on $|pq|$, similarly to the way in which it is shown that a θ-graph has a small dilation.

Let (p, q) be an edge in $\mathcal{G}_{\mathrm{mr}}[(1-\delta)\varepsilon]$. Consider the set Γ_p of cones with apex p that was used to define the edge set E_p, and let $c \in \Gamma_p$ be the cone containing q. Recall that we added an edge from p to the k points in c that are closest to p when projected onto the half-line ℓ_c, where $k := 2 \cdot \textsc{MinPts} - 3$. Hence, when q is one of these k closest points we are done. Otherwise, let $r \in D(c)$ be the $(\textsc{MinPts} - 1)$-th closest point.

▶ **Claim.** *(i) $d_{\mathrm{core}}(r) \leqslant (1-\delta)\varepsilon$, (ii) $|pr| \leqslant \varepsilon$, and (iii) $|rq| < |pq|$.*



Both the dark and the light grey region contain at least MinPts−2 points, not counting p, q, r. Depending on the position of r, all points in the light region or all points in the dark region have distance at most $(1-\delta)\varepsilon$ from r.

Figure 3 Illustration for the proof of Lemma 11.

Before we prove this claim, we first we argue that the claim allows us to finish our inductive proof. Since (p, q) is an edge in $\mathcal{G}_{\mathrm{mr}}[(1-\delta)\varepsilon]$ we have $d_{\mathrm{mr}}(p, q) \leqslant (1-\delta)\varepsilon$. Thus $|pq| \leqslant (1-\delta)\varepsilon$ and $d_{\mathrm{core}}(q) \leqslant (1-\delta)\varepsilon$. Together with parts (i) and (iii) of the claim this implies that (r, q) is an edge in $\mathcal{G}_{\mathrm{mr}}[(1-\delta)\varepsilon]$ with $|rq| < |pq|$.

In the base case of our inductive proof, where (p, q) is the shortest edge in $\mathcal{G}_{\mathrm{mr}}[(1-\delta)\varepsilon]$, this cannot occur. Thus q must be one of the k closest points in the cone c, and we have an edge between p and q in $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)[\varepsilon]$ by construction.

If we are not in the base case, then we have a path from r to q in $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)[\varepsilon]$ by the induction hypothesis. Moreover, (p, r) is an edge in $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)$ by construction. Since $|pr| \leqslant \varepsilon$ by part (ii) of the claim, we have a path from p to q in $\overline{\mathcal{G}}_{\mathrm{mr}}(\delta)[\varepsilon]$.

It remains to prove the claim. For this we use the following fact [19, Lemma 4.1.4], which is also used to prove that a θ-graph has small dilation. Note that although Lemma 4.1.4 in [19] is stated in 2 dimensions, but the proof never assumes that the line on which is projected lives in the same plane and clearly three points s, t, p live in a single plane.

▶ **Fact.** *Let s, t be any two points in a cone $c \in \Gamma_p$ such that, when projected onto the half-line ℓ_c, the distance from p to s is smaller than the distance from p to t. Then $|ps| \leqslant |pt|/\cos\theta$ and $|st| < |pt| - (\cos\theta - \sin\theta)|ps| \leqslant |pt|$, since we can assume θ is sufficiently small that* $\cos\theta - \sin\theta > 0$.

Part (iii) of the claim immediately follows from this fact by taking $s := r$ and $t := q$. Part (ii) follows again by taking $s := r$ and $t := q$, using that $|pq| \leqslant (1-\delta)\varepsilon$ and that we have chosen δ such that $\cos\theta = 1-\delta$. For part (i) we must prove that there are at least MinPts − 1 points within distance $(1-\delta)\varepsilon$ from r. Recall that r is the (MinPts − 1)-th closest point to p in the cone c, measured in the projection onto the half-line ℓ_c. Let $r_1, \ldots, r_k$ be the k closest points; thus $r = r_i$ for $i =$ MinPts − 1. We distinguish two cases: $|pr| \leqslant (1-\delta)\varepsilon$ and $|pr| > (1-\delta)\varepsilon$. See also Fig. 3.

In the former case we can conclude that $|r_i r| \leqslant (1-\delta)\varepsilon$ for all $1 \leqslant i \leqslant$ MinPts − 2 by setting $s := r_i$ and $t := r$ and using $|pr| \leqslant (1-\delta)\varepsilon$. Thus, including the point p, we know that r has at least MinPts − 1 points within distance $(1-\delta)\varepsilon$.

In the latter case we will argue that $|r_i r| \leqslant (1-\delta)\varepsilon$ for all MinPts $\leqslant i \leqslant 2 \cdot$ MinPts − 3. Since by part (iii) of the claim we have $|rq| \leqslant (1-\delta)\varepsilon$, we conclude that also in the latter case r has at least MinPts − 1 points within distance $(1-\delta)\varepsilon$. To argue that $|r_i r| \leqslant (1-\delta)\varepsilon$ we first note that for any point $s \in c$ we have $|ss^*| \leqslant \sin\theta \cdot |ps|$, where s^* denotes the orthogonal projection of s onto ℓ_c. Thus

$$\begin{aligned}
|rr_i| &\leqslant |rr^*| + |r^* r_i^*| + |r_i r_i^*| \\
&\leqslant \sin\theta \cdot |pr| + |r^* q^*| + \sin\theta \cdot |pr_i| \\
&\leqslant 2\sin\theta \cdot |pq|/\cos\theta + |r^* q^*| \\
&= 2\sin\theta \cdot |pq|/\cos\theta + |pq^*| - |pr^*| \\
&\leqslant 2\sin\theta \cdot |pq|/\cos\theta + |pq| - |pr|\cos\theta \\
&\leqslant \left(2\tfrac{\sin\theta}{\cos\theta} + 1 - \cos\theta\right) \cdot (1-\delta)\varepsilon
\end{aligned}$$

where the last inequality uses $|pq| \leqslant (1-\delta)\varepsilon$ and that we are now considering the case $|pr| > (1-\delta)\varepsilon$. Since we can assume that θ is small enough to ensure $2\sin\theta < \cos^2\theta$, we conclude that, indeed, $|rr_i| \leqslant (1-\delta)\varepsilon$. This finishes the proof part (i) of the claim and hence, of the lemma. ◀

Combining the previous two lemmas we obtain the following theorem.

▶ **Theorem 12.** *Let D be a set of n points in $\mathbb{R}^d$, and let ε and* MinPts *be given constants. Then, for any given $\delta > 0$, we can compute a δ-approximate* hdbscan *clustering on D with respect to ε and* MinPts *for the Euclidean metric in $O((n/\delta^{(d-1)/2})\log^{d-1} n)$ time.*

References

1 P. Afshani and T.M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th ACM-SIAM Symp. on Discr. Alg.*, pages 180–186, 2009.

2 P.K. Agarwal, H. Edelsbrunner, and O. Schwarzkopf. Euclidean minimum spanning trees and bichromatic closest pairs. *Discr. Comput. Geom.* 6:407–422 (1991).

3 M. Ankerst, M.M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. *SIGMOD Rec.* 28:49–60 (1999).

4 S. Arya and T.M. Chan. Better ε-dependencies for offline approximate nearest-neighbor search, Euclidean minimum spanning trees, and ε-kernels. In *Proc. 30th Symp. on Comput. Geom.*, pages 416–425, 2014.

5 B. Borah and D. Bhattacharyya. An improved sampling-based DBSCAN for large spatial databases. In *Proc. Int. Conf. on Intelligent Sensing and Inf. Proc.*, pages 92–96, 2004.

6 M. de Berg, O. Cheong, M. van Kreveld and M. Overmars. *Computational Geometry: Algorithms and Applications (3rd edition).* Springer-Verlag, 2008.

7 M. de Berg, A. Gunawan, M. Roeloffzen. Faster DB-scan and HDB-scan in Low-Dimensional Euclidean Spaces *CoRR:abs/1702.08607*, 2017.

8 R.J.G.B. Campello, D. Malouvi and J. Sander. Density-based clustering based on hierarchical density estimates. In *Proc. 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, LNCS 7819, pages 160–172, 2013.

9 D.Z. Chen, M.H. Smid and B. Xu. Geometric Algorithms for Density-based Data Clustering. *Int. J. Comput. Geometry Appl.* 15:239–260 (2005)

10 T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein. *Introduction to Algorithms* (3rd edition), MIT Press, 2009.

11 J. Erickson. On the relative complexities of some geometric problems. In *Proc. 7th Canadian Conf. Comput. Geom. (CCCG)* pages 85–90, 1995.

12 M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

13 J. Gan and Y. Tao. DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *Proc. 2015 ACM SIGMOD Int. Conf. on Management of Data*, pages 519–530.

14 J. Gudmundsson, M. Hammer and M. van Kreveld. Higher order Delaunay triangulations. *Computational Geometry: Theory and Applications* 23: 85–98 (2002).

15 A. Gunawan. A faster algorithm for DBSCAN. Master's thesis, TU Eindhoven, March 2013.

16 B. Liu. A fast density-based clustering algorithm for large databases. In *Proc. Int. Conf. on Machine Learning and Cybernetics*, pages 996–1000, 2006.

17 S. Mahran and K. Mahar. Using grid for accelerating density-based clustering. In *8th Int. Conf. on Computer and Information Technology*, pages 35–40, 2008.

18 J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications* 2: 169–186 (1993).

19 G. Narasimhan and M. Smid. *Geometric Spanner Networks.* Cambridge Univ. Press, 2007.

20 M.M.A. Patwary, D. Palsetia, A. Agrawal, W.-K. Liao, F. Manne, and A. Choudhary. Scalable parallel OPTICS data clustering using graph algorithmic techniques. In *Proc. Int. Conf. on High Perf. Computing, Networking, Storage and Analysis* pages 49:1–49:12, 2013.

21 P. Tan, M. Steinbach and V. Kumar. *Introduction to Data Mining.* Addison-Wesley (2006).

22 P.M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbor problem. *Discr. Comput. Geom.* 4:101–115 (1989).

Fully-Dynamic and Kinetic Conflict-Free Coloring of Intervals with Respect to Points[*]

Mark de Berg[1], Tim Leijsen[2], Aleksandar Markovic[3], André van Renssen[†4], Marcel Roeloffzen[‡5], and Gerhard Woeginger[6]

1 **TU Eindhoven, Eindhoven, The Netherlands**
mdberg@win.tue.nl

2 **TU Eindhoven, Eindhoven, The Netherlands**

3 **TU Eindhoven, Eindhoven, The Netherlands**
a.markovic@tue.nl

4 **National Institute of Informatics, Tokyo and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
andre@nii.ac.jp

5 **National Institute of Informatics, Tokyo and JST, ERATO, Kawarabayashi Large Graph Project, Japan**
marcel@nii.ac.jp

6 **TU Eindhoven, Eindhoven, The Netherlands**
g.woeginger@tue.nl

Abstract

We introduce the fully-dynamic conflict-free coloring problem for a set S of intervals in $\mathbb{R}^1$ with respect to points, where the goal is to maintain a conflict-free coloring for S under insertions and deletions. A coloring is conflict-free if for each point p contained in some interval, p is contained in an interval whose color is not shared with any other interval containing p. We investigate trade-offs between the number of colors used and the number of intervals that are recolored upon insertion or deletion of an interval. Our results include:

- a lower bound on the number of recolorings as a function of the number of colors, which implies that with $O(1)$ recolorings per update the worst-case number of colors is $\Omega(\log n/\log\log n)$, and that any strategy using $O(1/\varepsilon)$ colors needs $\Omega(\varepsilon n^{\varepsilon})$ recolorings;
- a coloring strategy that uses $O(\log n)$ colors at the cost of $O(\log n)$ recolorings, and another strategy that uses $O(1/\varepsilon)$ colors at the cost of $O(n^{\varepsilon}/\varepsilon)$ recolorings;
- stronger upper and lower bounds for special cases.

We also consider the kinetic setting where the intervals move continuously (but there are no insertions or deletions); here we show how to maintain a coloring with only four colors at the cost of three recolorings per event and show this is tight.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Conflict-free colorings, Dynamic data structures, Kinetic data structures

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.26

[*] A full version of the paper is available at https://arxiv.org/abs/1701.03388.

[†] A. v. R. and was supported by JST ERATO Grant Number JPMJER1201, Japan.

[‡] M. R. was supported by JST ERATO Grant Number JPMJER1201, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 26; pp. 26:1–26:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider a set S of fixed base stations that can be used for communication by mobile clients. Each base station has a transmission range, and a client can potentially communicate via that base station when it lies within the transmission range. However, when a client is within reach of several base stations that use the same frequency, the signals will interfere. Hence, the frequencies of the base stations should be assigned in such a way that this problem does not arise. Moreover, the number of used frequencies should not be too large. Even *et al.* [10] and Smorodinsky [14] introduced conflict-free colorings to model this problem, as follows. Let S be a set of disks in the plane, and for a point $q \in \mathbb{R}^2$ let $S(q) \subseteq S$ denote the set of disks containing the point q. A coloring of the disks in S is *conflict-free* if, for any point $q \in \mathbb{R}^2$ with non-empty $S(q)$, the set $S(q)$ has at least one disk with a color that is unique among the disks in $S(q)$. Even *et al.* [10] proved that any set of n disks in the plane admits a conflict-free coloring with $O(\log n)$ colors, and this bound is tight in the worst case.

The concept of conflict-free colorings can be generalized and extended in several ways, giving rise to a host of challenging problems. Below we mention some of them; for lack of space we only discuss the papers most directly related to our work. A more extensive overview is given by Smorodinsky [15]. One obvious generalization is to work with types of regions other than disks. For instance, Even *et al.* [10] showed how to find a coloring with $O(\log n)$ colors for a set of translations of any single centrally symmetric polygon. Har-Peled and Smorodinsky [12] extended this result to regions with near-linear union complexity. One can also consider the dual setting, where one wants to color a given set P of n points in the plane, such that any disk – or rectangle, or other range from a given family – contains at least one point with a unique color (if it contains any point at all). This too was studied by Even *et al.* [10] and they show that this can be done with $O(\log n)$ colors when the ranges are disks or scaled translations of a single centrally symmetric convex polygon.

The results mentioned above deal with the static setting, in which the set of objects to be colored is known in advance. This may not always be the case, leading Fiat *et al.* [11] to introduce the *online* version of the conflict-free coloring problem. Here the objects to be colored arrive one at a time, and each object must be colored upon arrival. Fiat *et al.* show that when coloring points in the plane with respect to disks, n colors may be needed in the online version. Hence, they turn their attention to the 1-dimensional problem of online coloring points with respect to intervals. They prove that this can be done deterministically with $O(\log^2 n)$ colors and randomized with $O(\log n \log\log n)$ colors with high probability. Later Chen [8] gave a randomized algorithm that uses $O(\log n)$ colors with high probability. In the same paper, similar results were obtained for conflict-free colorings of points with respect to halfplanes, unit disks and axis-aligned rectangles of almost the same size. In these cases the colorings use $O(\text{polylog } n)$ colors with high probability. Bar-Noy, Cheilaris, and Smorodinsky [3] discussed several versions of the deterministic one-dimensional variant. Furthermore, Abam *et al.* [1] studied the dual version of coloring intervals on a line with respect to points. Later, Bar-Noy *et al.* [2] considered the case where recolorings are allowed for each insertion. They prove that for coloring points in the plane with respect to halfplanes, one can obtain a coloring with $O(\log n)$ colors in an online setting at the cost of $O(n)$ recolorings in total. More recent variants include strong conflict-free colorings [7, 13], where we require several unique colors, and conflict-free multicolorings [4], which allow assigning multiple colors to a point. Even more variants of online conflict-free colorings can be found in the survey [15].

Our contributions. We introduce a variant of the conflict-free coloring problem where the objects to be colored arrive and disappear over time. This *fully-dynamic conflict-free coloring problem* models a scenario where new base stations may be deployed (to deal with increased capacity demands, for example) and existing base stations may break down or be taken out of service (either permanently or temporarily). We also define the *semi-dynamic conflict-free coloring problem* as the online variant where recolorings are allowed (or the fully-dynamic variant without deletions). Note that when we talk about the *dynamic* variant, we mean *fully-dynamic*. These natural variants have, to the best of our knowledge, not been considered so far. It is easy to see that, unless one maintains a coloring in which any two intersecting objects have distinct colors, there is always a sequence of deletions that invalidates a given conflict-free coloring. Hence, recolorings are needed to ensure that the new coloring is conflict-free. This leads to the question: how many recolorings are needed to maintain a coloring with a certain number of colors? We initiate the study of fully-dynamic conflict-free colorings by considering the problem of coloring intervals with respect to points. In this variant, we are given a (dynamic) set S of intervals in $\mathbb{R}^1$, which we want to color such that for any point $q \in \mathbb{R}^1$ the set $S(q)$ of intervals containing q contains an interval with a unique color. This version of the problem can be used to model the case where the base stations are located along a highway, for instance, and 1-dimensional range and frequency assignment problems have already been studied in various settings [2, 7, 11]. Moreover, the lower bounds that we prove hold for the 2-dimensional problem as well. In the static setting, coloring intervals is rather easy: a simple procedure yields a conflict-free coloring with three colors. The dynamic version turns out to be much more challenging.

In Section 2 we prove lower bounds on the possible tradeoffs between the number of colors used and the worst-case number of recolorings per update: for any algorithm that maintains a conflict-free coloring on a sequence of n insertions of intervals with at most $c(n)$ colors and at most $r(n)$ recolorings per insertion, we must have $r(n) > n^{1/(c(n)+1)}/(8c(n))$. This implies that for $O(1/\varepsilon)$ colors we need $\Omega(\varepsilon n^{\varepsilon})$ recolorings per updated, and with only $O(1)$ recolorings per update we must use $\Omega(\log n/ \log\log n)$ colors.

In Section 3 we then present several algorithms that achieve bounds close to our lower bound. All bounds are worst-case, unless specifically stated otherwise. First, we present two algorithms for the case where the interval endpoints come from a universe of size U. One algorithm uses $O(\log U)$ colors and two recolorings per update; the other uses $O(\log_t U)$ colors and $O(t)$ recolorings per update in the worst case, where $2 \leqslant t \leqslant U$ is a parameter. We then extend the second algorithm to an unbounded universe, leading to two results: we can use $O(\log_t n)$ colors and perform at most $O(t \log_t n)$ recolorings per update for any fixed $t \geqslant 2$, or we can use $O(1/\varepsilon)$ colors and $O(n^{\varepsilon}/\varepsilon)$ recolorings, amortized, for any fixed $\varepsilon > 0$.

Finally, in Section 4 we turn our attention to *kinetic conflict-free colorings*. Here the intervals do not appear or disappear, but their endpoints move continuously on the real line. At each *event* where two endpoints of different intervals cross each other, the coloring may need to be adapted so that it stays conflict-free. One way to handle this is to delete the two intervals involved in the event, and re-insert them with the new endpoint order. We show that a specialized approach is much more efficient: we show how to maintain a conflict-free coloring with four colors at the cost of three recolorings per event. We also show that on average $\Theta(1)$ recolorings per event are needed in the worst case when using only four colors.

Due to space constraints some proofs have been deferred to the full version [6].

2 Lower bounds for semi-dynamic conflict-free colorings

In this section we present lower bounds on the semi-dynamic (insertion only) conflict-free coloring problem for intervals. More precisely, we present lower bounds on the number of recolorings necessary to guarantee a given upper bound on the number of colors. We prove a general lower bound and a stronger bound for so-called local algorithms. The general lower bound uses a construction where the choice of segments to be added depends on the colors of the segments already inserted. This adaptive construction is also valid for randomized algorithms, but it does not give a lower bound on the expected behavior.

▶ **Theorem 1.** *Let* ALG *be a deterministic algorithm for the semi-dynamic conflict-free coloring of intervals. Suppose that on any sequence of $n > 0$ insertions,* ALG *uses at most $c(n)$ colors and $r(n)$ recolorings per insertion, where $r(n) > 0$. Then $r(n) > n^{1/(c(n)+1)}/(8c(n))$.*

Proof. We first fix a value for n and define $c := c(n)$ and $r := r(n)$. Our construction will proceed in rounds. In the i-th round we insert a set R_i of n_i disjoint intervals – which intervals we insert depends on the current coloring provided by ALG. After R_i has been inserted (and colored by ALG), we choose one of the colors used by ALG for R_i to be the *designated color* for the i-th round. We denote this designated color by c_i. We will argue that in each round we can pick a different designated color, so that the number of rounds, ρ, is a lower bound on the number of colors used by ALG. We then prove a lower bound on ρ in terms of n, c, and r, and derive the theorem from the inequality $\rho \leqslant c$.

To describe our construction more precisely, we need to introduce some notation and terminology. Let $R_i := \{I_1, \dots, I_{n_i}\}$, where the intervals are numbered from left to right. (Recall that the intervals in R_i are disjoint.) To each interval $I = I_j$ we associate the set $I^e := (a, b)$, where a is the right endpoint of I, and b is the left endpoint of I_{j+1} if $j < n_i$ and $+\infty$ if $j = n_i$, that is, I^e represents the empty space to the right of I. We call (I, I^e) an *i-brick*. We define the color of a brick (I, I^e) to be the color of I, and we say a point or an interval is contained in this brick if it is contained in $I \cup I^e$. Recall that each round R_i has a designated color c_i. We say that an i-brick $B := (I, I^e)$ is *living* if:

- I has the designated color c_i;
- if $i > 1$ then both I and I^e contain living $(i-1)$-bricks.

A brick that is not alive is called *dead* and an event such as a recoloring that causes a brick to become dead is said to *kill* the brick. By recoloring an interval I, ALG can kill the brick $B = (I, I^e)$ and the death of B may cause some bricks containing B to be killed as well.

We can now describe how we generate the set R_i of intervals we insert in the i-th round and how we pick the designated colors. (Note that the designated color of a round is fixed once it is picked; it is not updated when recolorings occur.) We denote by R_i^* the subset of intervals $I \in R_i$ such that (I, I^e) is a living i-brick. Note that R_i^* can be defined only after the i-th round, when we have picked the designated color c_i.

1. The set R_1 contains the $\frac{n}{2}$ intervals $[0, 1], [2, 3], \dots, [n-2, n-1]$, and the designated color c_1 of the first round is the color used most often in the coloring produced by ALG after insertion of the last interval in R_1.
2. To generate R_i for $i > 1$, we proceed as follows. Partition R_{i-1}^* into groups of $4r$ consecutive intervals. (If $|R_{i-1}^*|$ is not a multiple of $4r$, the final group will be smaller than $4r$. This group will be ignored.) For each group $G := I_1, \dots, I_{4r}$ we put an interval I_G into R_i, which starts at the left endpoint of I_1 and ends slightly before the left endpoint of I_{2r+1}; see Fig. 1 for an illustration.

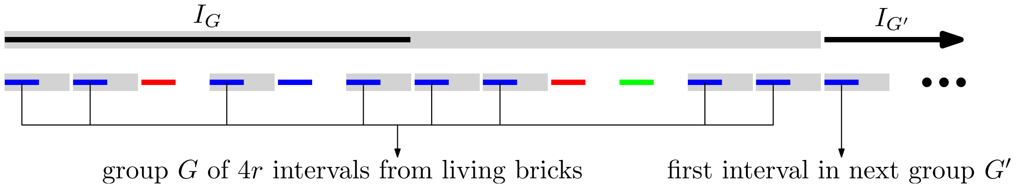

I_G
$I_{G'}$
group G of $4r$ intervals from living bricks
first interval in next group G'

Figure 1 Example of how the intervals are created when $r = 2$. The designated color c_{i-1} is blue, and the grey rectangles around them indicate living $(i-1)$-bricks. The grey rectangle around I_G indicates the brick (I_G, I_G^e). Note that $I_{G'}$ extends further to the right.

The designated color c_i is picked as follows. Consider the coloring after the last interval of R_i has been inserted, and let $C(i)$ be the set of colors assigned by ALG to intervals in R_i and that are not a designated color from a previous round – we argue below that $C(i) \neq \emptyset$. Then we pick c_i as the color from $C(i)$ that maximizes the number of living i-bricks.

We continue generating sets R_i in this manner until $|R_i^*| < 4r$, at which point the construction finishes. Below we prove that in each round ALG must introduce a new designated color, and we prove a lower bound on the number of rounds in the construction.

▶ **Claim.** *Let $B = (I, I^e)$ be a living i-brick. Then for any $j \in \{1, \ldots, i\}$ there is a point $q_j \in I \cup I^e$ that is contained in a single interval of color c_j and in no other interval from $\bigcup_{\ell=1}^{i-1} R_\ell$. Moreover, there is a point $q_j \in I \cup I^e$ not contained in any interval from $\bigcup_{\ell=1}^{i-1} R_\ell$.*

Proof of claim. We prove this by induction on i. For $i = 1$ the statement is trivially true, so suppose $i > 1$. By definition, both I and I^e contain living $(i-1)$-bricks, $\overline{B}$ and $\overline{B}^e$. Using the induction hypothesis we can now select a point q_j with the desired properties: for $j = i$ we use that $\overline{B}$ contains a point that is not contained in any interval from $\bigcup_{\ell=1}^{i-1} R_\ell$, for $j < i$ we use that $\overline{B}^e$ contains a point in an interval of color c_j and in no other interval from $\bigcup_{\ell=1}^{i-1} R_\ell$, and to find a point not contained in any interval from $\bigcup_{\ell=1}^{i-1} R_\ell$ we can also use $\overline{B}^e$. ◀

Now consider the situation after the i-th round, but before we have chosen the designated color c_i. We say that a color c is *eligible* (to become c_i) if $c \neq c_1, \ldots, c_{i-1}$, and we say that an i-brick (I, I^e) is eligible if its color is eligible and (I, I^e) would be living if we were to choose its color as the designated color c_i. Note that due to some recolorings, some of the newly inserted intervals might not contain any living brick and hence can never be living no matter the designated color; the next claim shows that at most half intervals inserted this round are eligible.

▶ **Claim.** *Immediately after the i-th round, at least half of the i-bricks are eligible.*

Proof of claim. Consider an i-brick (I, I^e). At the beginning of the i-th round, before we have actually inserted the intervals from R_i, both the interval I and its empty space I^e contain $2r$ living $(i-1)$-bricks. As the intervals from R_i are inserted, ALG may recolor certain intervals from $R_1 \cup \ldots \cup R_{i-1}$, thereby killing some of these $(i-1)$-bricks. Now suppose that ALG recolored at most $2r - 1$ of the intervals from $R_1 \cup \ldots \cup R_{i-1}$ that are contained in $I \cup I^e$. Then both I and I^e still contain a living $(i-1)$-brick. By the previous claim and the definition of a conflict-free coloring, this implies ALG cannot use any of the colors c_j with $j < i$ for I. Hence, the color of I is eligible and the i-brick (I, I^e) is eligible as well.

It remains to observe that ALG can do at most rn_i recolorings during the i-th round. We just argued that to prevent an i-brick from becoming eligible, ALG must do at least $2r$ recolorings inside that brick. Hence, ALG can prevent at most half of the i-bricks from becoming eligible. ◀

Recall that after the i-th round we pick the designated color c_i that maximizes the number of living i-bricks. In other words, c_i is chosen to maximize $|R_i^*|$. Next we prove a lower bound on this number. Recall that ρ denotes the number of rounds.

▶ **Claim.** *For all $1 \leqslant i \leqslant \rho$ we have $|R_i^*| \geqslant n_1/(8rc)^i - 1$.*

Proof of claim. Since ALG can use at most c colors, we have $|R_1^*| \geqslant n_1/c$. Moreover, for $i > 1$ the number of intervals we insert is $\lfloor |R_{i-1}^*|/4r \rfloor$. By the previous claim at least half of these are eligible. The eligible intervals have at most c different colors, so if we choose c_i to be the most common color among them we see that $|R_i^*| \geqslant \lfloor |R_{i-1}^*|/4r \rfloor /2c$. We thus obtain the following recurrence:

$$|R_i^*| \geqslant \begin{cases} \dfrac{\lfloor |R_{i-1}^*|/4r \rfloor}{2c} & \text{if } i > 1, \\ \dfrac{n_1}{c} & \text{if } i = 1. \end{cases} \tag{1}$$

We can now prove the result using induction.

$$|R_i^*| \quad \geqslant \quad \frac{\lfloor |R_{i-1}^*|/4r \rfloor}{2c} \quad \geqslant \quad \frac{1}{2c} \cdot \left(\left(\frac{n_1}{(8rc)^{i-1}} - 1 \right) /4r - 1 \right) \quad > \quad \frac{n_1}{(8rc)^i} - 1. \qquad ◀$$

Finally we can derive the desired relation between n, c, and r. Since $n_1 = n/2$ and $n_{i+1} < n_i/2$ for all $i = 1, \ldots, \rho - 1$, the total number of insertions is less than n. The construction finishes when $|R_i^*| < 4r$. Hence, ρ, the total number of rounds, must be such that $n/(2(8rc)^\rho) - 1 \leqslant |R_\rho^*| < 4r$, which implies $\rho > \log_{8rc}(n/(8r+2)) > \log_{8rc} n - 1$. The number of colors used by ALG is at least ρ, since every round has a different designated color. Thus $c > \log_{8rc} n - 1$ and so $n \leqslant (8rc)^{c+1}$, from which the theorem follows. ◀

Two interesting special cases of the theorem are the following: with $r = O(1)$ we will have $c = \Omega(\log n / \log\log n)$, and for $c = O(1/\varepsilon)$ (for some small fixed $\varepsilon > 0$) we need $r = \Omega(\varepsilon n^\varepsilon)$. Note that the theorem requires $r > 0$. Obviously the $\Omega(\log n/\log\log n)$ lower bound on c that we get for $r = 1$ also holds for $r = 0$. For the special case of $r = 0$ – this is the standard online version of the problem – we can prove a stronger result, however: here we need at least $\lfloor \log n \rfloor + 1$ colors. This bound even holds for a nested set of intervals, that is, a set S such that $I \subset I'$, $I' \subset I$, or $I \cap I' = \emptyset$ for any two intervals $I, I' \in S$. We also show in the full paper [6] that a greedy algorithm achieves this bound for nested intervals.

Local algorithms. We now prove a stronger lower bound for so-called local algorithms. Intuitively, these are deterministic algorithms where the color assigned to a newly inserted interval I only depends on the structure and the coloring of the connected component where I is inserted – hence the name *local*. More precisely, local algorithms are defined as follows.

Suppose we insert an interval I into a set S of intervals that have already been colored. The union of the set $S \cup \{I\}$ consists of one or more connected components. We define $S(I) \subseteq S$ to be the set of intervals from S that are in the same connected component as I. (In other words, if we consider the interval graph induced by $S \cup \{I\}$ then the intervals in $S(I)$ form a connected component with I.) Order the intervals in $S(I) \cup \{I\}$ from left

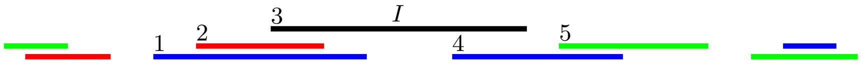


Figure 2 Example of a signature. The set $S(I)$ contains the segments labeled 1,2,4,5. The signature of I is $\langle 2, 1, 3, 4, 5, \text{red}, \text{blue}, \textsc{nil}, \text{blue}, \text{green}\rangle$.

to right according to their left endpoint, and then assign to every interval its rank in this ordering as its label. (Here we assume that all endpoints of the intervals in $S(I) \subseteq S$ are distinct. It suffices to prove our lower bound for this restricted case.) Based on this labeling we define a signature for $S(I) \cup \{I\}$ as follows. Let $\lambda_1, \ldots, \lambda_k$, where $k := |S(I)| + 1$, be the sequence of labels obtained by ordering the intervals from left to right according to their right endpoint. Furthermore, let c_i be the color of the interval labeled i, where $c_i = \textsc{nil}$ if the interval labeled i has not yet been colored. Then we define the *signature* of $S(I) \cup I$ to be the sequence $\text{sig}(I) := \langle \lambda_1, \ldots, \lambda_k, c_1, \ldots, c_k\rangle$; see Fig. 2.

We now define a semi-dynamic algorithm ALG to be *local* if upon insertion of an interval I the following holds: (i) ALG only performs recoloring in $S(I)$, and (ii) the color assigned to I and the recolorings in $S(I)$ are uniquely determined by $\text{sig}(I)$, that is, the algorithm is deterministic with respect to $\text{sig}(I)$. Note that randomized algorithms are not local.

To strengthen Theorem 1 for the case of local algorithms, it suffices to observe that the intervals inserted in the same round must all receive the same color. Hence, the factors c in the denominators of Inequality (1) disappear, leading to the theorem below. Note that for $r(n) = O(1)$, we now get the lower bound $c(n) = \Omega(\log n)$.

▶ **Theorem 2.** *Let* ALG *be a local algorithm for the semi-dynamic conflict-free coloring of intervals. Suppose that on any sequence of $n > 0$ insertions,* ALG *uses at most $c(n)$ colors and $r(n)$ recolorings per insertion, where $r(n) > 0$. Then $r(n) \geqslant n^{1/(c(n)+2)} - 2$.*

3 Upper bounds for fully-dynamic conflict-free colorings

Next we present algorithms to maintain a conflict-free coloring for a set S of intervals under insertions and deletions. The algorithms use the same structure, which we describe first. From now on, we use n to denote the current number of intervals in S.

Let P be the set of $2n$ endpoints of the intervals in S. (To simplify the presentation we assume that all endpoints are distinct, but the solution is easily adapted to the general case.) We will maintain a B-tree on the set P. A B-tree of minimum degree t on a set of points in $\mathbb{R}^1$ is a multi-way search tree in which each internal node has between t and $2t$ children (except the root, which may have fewer children) and all leaves are at the same level; see the book by Cormen *et al.* [9, Chapter 18] for details. Thus each internal or leaf node stores between $t-1$ and $2t-1$ points from P (again, the root may store fewer points). We denote the set of points stored in a node v by $P(v) := \{p_1(v), \ldots, p_{n_v}(v)\}$, where $n_v := |P(v)|$ and the points are numbered from left to right. For an internal node v we denote the i-th subtree of v, where $1 \leqslant i \leqslant n_v + 1$, by $\mathcal{T}_i(v)$. Note that the search-tree property guarantees that all points in $\mathcal{T}_i(v)$ lie in the range $(p_{i-1}(v), p_i(v))$, where $p_0 = -\infty$ and $p_{n_v+1} = \infty$.

We now associate each interval $I \in S$ to the highest node v such that I contains at least one of the points in $P(v)$, either in its interior or as one of its endpoints. Thus our structure is essentially an interval tree [5, Chapter 10] but with a B-tree as underlying tree structure. We denote the set of intervals associated to a node v by $S(v)$. Note that if $\text{level}(v) = \text{level}(w) = \ell$, for some nodes $v \neq w$, and $I \in S(v)$ and $I' \in S(w)$, then I and I' are separated by a point $p_i(z)$ of some node z at level $m < \ell$. Hence, $I \cap I' = \emptyset$.

We partition $S(v)$ into subsets $S_1(v), \dots, S_{n_v}(v)$ such that $S_i(v)$ contains all intervals $I \in S(v)$ for which $p_i(v)$ is the leftmost point from $P(v)$ contained in I. From each subset $S_i(v)$ we pick at most two *extreme intervals*: the *left-extreme interval* $I_{i,\text{left}}(v)$ is the interval from $S_i(v)$ with the leftmost left endpoint, and the *right-extreme interval* $I_{i,\text{right}}(v)$ is the interval from $S_i(v)$ with the rightmost right endpoint. Since all intervals from $S_i(v)$ contain the point $p_i(v)$, every interval from $S_i(v)$ is contained in $I_{i,\text{left}}(v) \cup I_{i,\text{right}}(v)$. Note that it may happen that $I_{i,\text{left}}(v) = I_{i,\text{right}}(v)$. Finally, we define $S_{\text{extr}}(v) := \bigcup_{i=1}^{n_v} \{I_{i,\text{left}}(v), I_{i,\text{right}}(v)\}$ to be the set of all extreme intervals at v.

Our two coloring algorithms both maintain a coloring with the following properties.

(A.1) For each level ℓ of the tree $\mathcal{T}$, there is a set $C(\ell)$ of colors such that the color sets of different levels are disjoint.

(A.2) For each node v at level ℓ in $\mathcal{T}$, the intervals from $S_{\text{extr}}(v)$ are colored locally conflict-free using colors from $C(\ell)$. Here *locally conflict-free* means that the coloring of $S_{\text{extr}}(v)$ is conflict-free if we ignore all other intervals.

(A.3) All non-extreme intervals receive a universal *dummy color*, which is distinct from any of the other colors used, that is, the dummy color is not in $C(\ell)$ for any level ℓ.

The two coloring algorithms that we present differ in the size of the sets $C(\ell)$ and in which local coloring algorithm is used for the sets $S_{\text{extr}}(v)$. It is not hard to show that the properties above guarantee a conflict-free coloring.

▶ **Lemma 3.** *Any coloring with properties (A.1)–(A.3) is conflict free and uses at most* $1 + \sum_\ell |C(\ell)|$ *colors.*

Next we describe two algorithms based on this general framework: one for the easy case where the interval endpoints come from a finite universe, and one for the general case.

Solutions for a polynomially-bounded universe. The framework above uses a B-tree on the interval endpoints. If the interval endpoints come from a universe of size U – for concreteness, assume the endpoints are integers in the range $0, \dots, U-1$ – then we can use a B-tree on the set $\{0, \dots, U-1\}$. Thus the B-tree structure never has to be changed.

▶ **Theorem 4.** *Let S be a dynamic set of intervals whose endpoints are integers in the range* $0, \dots, U-1$.

(i) *We can maintain a conflict-free coloring on S that uses $O(\log U)$ colors and that performs at most two recolorings per insertion and deletion.*

(ii) *For any t with $2 \leqslant t \leqslant U$, we can maintain a conflict-free coloring on S that uses $O(\log_t U)$ colors and performs $O(t)$ recolorings per insertion and deletion.*

When U is polynomially bounded in n – that is, $U = O(n^k)$ for some constant k – this gives very efficient coloring schemes. In particular, we can then get $O(\log n)$ colors with at most two recolorings using method (i), and we can get $O(1/\varepsilon)$ colors with $O(n^\varepsilon)$ recolorings (for any fixed $\varepsilon > 0$) by setting $t = U^{\varepsilon/k}$ in method (ii).

Note finally that we do not need to explicitly store the whole tree as it is enough to compute the location of any node when needed, yielding a linear space complexity.

A general solution. If the interval endpoints do not come from a bounded universe then we cannot use a fixed tree structure. Next we explain how to deal with this when we apply the method from Theorem 4(ii), which colors the sets $S_{\text{extr}}(v)$ using the so-called *chain method*: we take the interval with the leftmost left endpoint, and color it blue. Then, among all intervals whose left endpoint lies in the blue interval, we pick the one with the rightmost right endpoint and color it red. We then repeat this process, alternating between blue and red, until we reach the rightmost interval. Finally, we color all uncolored intervals grey.

Suppose we want to insert a new interval I into the set S. We first insert the two endpoints of I into the B-tree $\mathcal{T}$. Inserting an endpoint p can be done in a standard manner. The basic operations for an insertion are (i) to split a full node and (ii) to insert a point into a non-full leaf node.

Splitting a full node v (that is, a node with $2t-1$ points) is done by moving the median point into the parent of v, creating a node containing the $t-1$ points to the left of the median and another node containing the $t-1$ points to the right of the median. Note that this means that some intervals from $S(v)$ may have to be moved to $S(\mathit{parent}(v))$. Thus splitting a node v involves recoloring intervals in $S(v)$ and $S(\mathit{parent}(v))$. Observe that an interval only needs to be recolored if it was extreme before or after the change. Hence, we recolor $O(t)$ intervals when we split a node v.

Since an insertion splits only nodes on a root-to-leaf path and the depth of $\mathcal{T}$ is $O(\log_t n)$, the total number of recolorings due to node splitting is $O(t \log_t n)$. Moreover, inserting a point into a non-full leaf node only takes $O(t)$ recolorings. We conclude that an insertion performs $O(t \log_t n)$ recolorings in total. For deletions the argument is similar. Since recoloring at a single node induces $O(t)$ recolorings, the total number of recolorings is $O(t \log_t n)$.

▶ **Theorem 5.** *Let S be a dynamic set of intervals.*

(i) *For any fixed $t \geqslant 2$ we can maintain a conflict-free coloring on S that uses $O(\log_t n)$ colors and that performs $O(t \log_t n)$ recolorings per insertion and deletion, where n is the current number of intervals in S. In particular, we can maintain a conflict-free coloring with $O(\log n)$ colors using $O(\log n)$ recolorings per update.*

(ii) *For any fixed $\varepsilon > 0$ we can maintain a conflict-free coloring on S that uses $O(1/\varepsilon)$ colors and that performs $O(n^\varepsilon/\varepsilon)$ recolorings per insertion or deletion. The bound on the number of recolorings is amortized.*

The idea behind part (ii) is to use a t with $n^\varepsilon/2 \leqslant t \leqslant 2n^\varepsilon$. This causes the bound in (ii) to be amortized, since now we need to change t when n has halved or doubled.

We have not been able to efficiently generalize the first method of Theorem 4 to an unbounded universe. The problem is that splitting a node v may require many intervals in $S_{\text{extr}}(v)$ to be recolored, since many intervals may be moved to $\mathit{parent}(v)$. Hence, the method would use the same number of recolorings as the chain method, but more colors.

Bounded-length intervals. Next we present a simple method that allows us to improve the bounds when the intervals have length between 1 and L for some constant $L > 1$.

▶ **Theorem 6.** *Let S be a dynamic set of intervals with lengths in the range $[1, L)$ for some fixed $L > 1$. Suppose we have a dynamic conflict-free coloring algorithm for a general set of intervals that uses at most $c(n)$ colors and at most $r(n)$ recolorings for any insertion or deletion. Then we can obtain a dynamic conflict-free coloring algorithm on S that uses at most $2 \cdot c(2L) + 1$ colors and at most $2 \cdot r(2L) + 1$ recolorings for any insertion or deletion.*

For instance, by applying Theorem 5(i) we can maintain a coloring with $O(\log L)$ colors and $O(\log L)$ recolorings. We can also plug in a trivial dynamic algorithm with $c(n) = n$ and $r(n) = 0$ to obtain $4L + 1$ colors with only 1 recoloring per update; when L is sufficiently small this gives a better result.

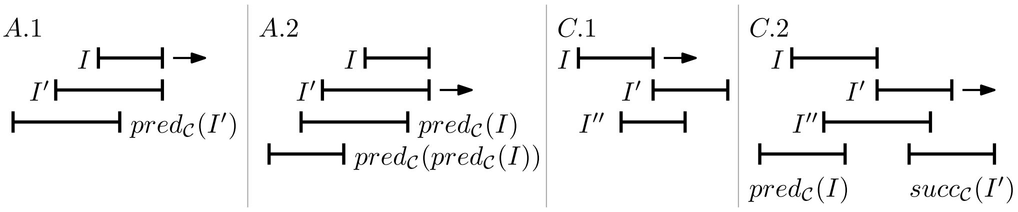


Figure 3 Illustration of the different events in the KDS.

4 Kinetic conflict-free colorings

In this section we consider conflict-free colorings of a set of intervals in $\mathbb{R}^1$ whose endpoints move continuously. Note that we allow the endpoints of an interval to move independently, that is, we allow the intervals to expand or shrink over time. We show that by using only three recolorings per event – an event is when two endpoints cross each other – we can maintain a conflict-free coloring consisting of only four colors. Our recoloring strategy is based on the chain method discussed in the proof of Theorem 4(ii). This method uses three colors: two colors for the chain and one dummy color. To be able to maintain the coloring in the kinetic setting without using many recolorings, we relax the chain properties and we allow ourselves three colors for the chain. Next we describe the invariants we maintain on the chain and its coloring, and we explain how to re-establish the invariants when two endpoints cross each other. In the remainder we assume that the endpoints of the chains are in general position except at events, and that events do not coincide (that is, we never have three coinciding endpoints and we never have two events at the same time). These conditions can be removed by using consistent tie-breaking.

The chain invariants. Let S be the set of intervals to be colored, where all interval endpoints are distinct. (Recall that we assumed this to be the case except at event times.) Consider a subset $\mathcal{C} \subseteq S$ and order the intervals in $\mathcal{C}$ according to their left endpoint. We denote the predecessor of an interval $I \in \mathcal{C}$ in this order by $pred_{\mathcal{C}}(I)$, and we denote its successor by $succ_{\mathcal{C}}(I)$. A *chain* (for S) is defined as a subset $\mathcal{C}$ with the following three properties.

(C1) Any interval $I \in \mathcal{C}$ can intersect only two other intervals in $\mathcal{C}$, namely $pred_{\mathcal{C}}(I)$ and $succ_{\mathcal{C}}(I)$.

(C2) Any interval $I \in S \setminus \mathcal{C}$ is completely covered by the intervals in $\mathcal{C}$.

(C3) No interval $I \in \mathcal{C}$ is fully contained in any other interval $I' \in S$.

Now consider a set S and a chain $\mathcal{C}$ for S. We maintain the following *color invariant*: each interval $I \in \mathcal{C}$ has a non-dummy color and this color is different from the color of $succ(I)$, and each interval in $S \setminus \mathcal{C}$ has the dummy color.

▶ **Lemma 7.** *Let S be a set of intervals and $\mathcal{C}$ be a chain for S. Then any coloring of S satisfying the color invariant is conflict-free.*

Handling events. Our kinetic coloring algorithm maintains a chain $\mathcal{C}$ for I and a coloring with three colors (excluding the dummy color) satisfying the color invariant. Later we show how to re-establish the color invariant at each event, but first we show how to update the chain by adding at most one interval to the chain and removing at most two. We distinguish several cases.

- *Case A: The right endpoints of two intervals I and I' cross.*
  Assume without loss of generality that I is shorter than I'. We have two subcases.
  - *Subcase A.1: Interval I is contained in I' before the event.* In this case I was not a chain interval before the event. If after the event I is still fully covered by the chain intervals, then there is nothing to do: we can keep the same chain. Otherwise, property (C2) is violated after the event. We now proceed as follows. First we add I to the chain. If I intersects $pred_{\mathcal{C}}(I')$ – note that I' must be a chain interval if (C2) is violated – then we remove I' from the chain.
  - *Subcase A.2: Interval I is contained in I' after the event.* If I was not a chain interval, there is nothing to do. Otherwise property (C3) no longer holds after the event, and we have to remove I from the chain. If I' is also a chain interval then this suffices. Otherwise we add I' to the chain, and remove $pred_{\mathcal{C}}(I)$ if $pred_{\mathcal{C}}(pred_{\mathcal{C}}(I))$ intersects I'.
- *Case B: The left endpoints of two intervals I and I' cross.*
  This case is symmetric to Case A.
- *Case C: The right endpoint of an interval I crosses the left endpoint of an interval I'.*
  Again we have two subcases.
  - *Subcase C.1: Intervals I and I' start intersecting.* Note that properties (C2) and (C3) still hold after the event. The only possible violation is in property (C1), namely when both I and I' are chain intervals and there is a chain interval I'' with $pred_{\mathcal{C}}(I'') = I$ and $succ_{\mathcal{C}}(I'') = I'$. In this case we simply remove I'' from the chain.
  - *Subcase C.2: Intervals I and I' stop intersecting.* First note that this cannot violate properties (C1) and (C3). The only possible violation is property (C2), namely when both I and I' are chain intervals and there is at least one non-chain interval containing the common endpoint of I and I' at the event. Of all such non-chain intervals, let I'' be the interval with the leftmost left endpoint. Note that I'' is not contained in any other interval, so we can add I'' to the chain without violating (C3). After adding I'' we check if we have to remove I and/or I': if I'' intersects $pred_{\mathcal{C}}(I)$ then we remove I from the chain, and if I'' intersects $succ_{\mathcal{C}}(I')$ then we remove I' from the chain.

It is easy to check that in each of these cases the new chain that we generate has the chain properties (C1)–(C3). Next we show that each case requires at most three recolorings and summarize the result.

▶ **Lemma 8.** *In each of the above cases, the changes to the chain require at most three recolorings to re-establish the color invariant.*

▶ **Theorem 9.** *Let S be a kinetic set of intervals in $\mathbb{R}^1$. We can maintain a conflict-free coloring for S with four colors at the cost of at most three recolorings per event, where an event is when two interval endpoints cross each other.*

A lower bound. Now consider the simple scenario where the intervals are rigid – each interval keeps the same length over time – and each interval is either stationary or moves with unit speed to the right. Our coloring algorithm may perform recolorings whenever two endpoints cross, which means that we do $O(n^2)$ recolorings in total. We show that even in this simple setting, this bound is tight in the worst case if we use at most four colors.

Consider four intervals I_1, I_2, I_3, I_4 where $I_i = (a_i, b_i)$, with $a_i < b_i$ as shown in Figure 4. Here $I_2 \subset I_1$, $I_4 \subset I_3$, the right endpoints of I_1 and I_2 are contained in $I_3 \cap I_4$, and the left endpoints of I_3 and I_4 are contained in $I_1 \cap I_2$. The exact locations of the endpoints with respect to each other is not important and we focus on the different overlap sets of the

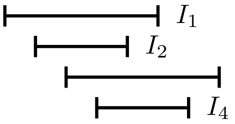


Figure 4 The gadget used to show the lower bound.

gadget. Specifically within a gadget there is a point contained in each of the following sets,

$$G_1, \dots, G_7 := \{I_1\}, \{I_1, I_2\}, \{I_1, I_2, I_3, \}, \{I_1, I_2, I_3, I_4\}, \{I_1, I_3, I_4\}, \{I_3, I_4\}, \{I_3\}.$$

Based on these sets we can show that no coloring for crossing gadgets exists that provides a valid conflict-free coloring for each combination of intersection sets between the two gadgets. The proof relies on the following lemma.

▶ **Lemma 10.** *Let $G = \{I_1, I_2, I_3, I_4\}$ and $H = \{J_1, J_2, J_3, J_4\}$ be two gadgets, with overlap sets $G_1, \dots, G_7$ and $H_1, \dots, H_7$ as defined above. There is no 4-coloring for G and H such that all sets $\{G_1, \dots, G_7\} \cup \{H_1, \dots, H_7\} \cup \{1 \leqslant i, j \leqslant 7 \mid G_i \cup H_i\}$ are conflict-free.*

Proof. We can assume that not both I_1, I_2, I_3, I_4 and J_1, J_2, J_3, J_4 use all four colors, otherwise $G_4 \cup H_4 = \{I_1, I_2, I_3, I_4, J_1, J_2, J_3, J_4\}$ is not conflict-free. It is also not possible to use at most two colors, since each gadget by itself needs to be conflict-free. Hence, suppose that there are exactly three colors among I_1, I_2, I_3, I_4 (the other case is symmetric), say two are red, one is blue, and one is green. We define $\text{col}(G_i)$, respectively $\text{col}(H_i)$, to be the multiset of the colors used by the intervals in G_i, respectively H_i. Then $\text{col}(G_4) = \{\text{red}, \text{red}, \text{blue}, \text{green}\}$ and without loss of generality, $\text{col}(G_2) = \{\text{red}, \text{blue}\}$ and $\text{col}(G_6) = \{\text{red}, \text{green}\}$. We now have two cases.

1. One interval among J_1, J_2, J_3, J_4 uses the fourth color, say yellow. If J_1 or J_2 is yellow, then either $\text{col}(H_6) = \{\text{red}, \text{blue}\}$, implying that $G_2 \cup H_6$ is not conflict-free; or $\text{col}(H_6) = \{\text{red}, \text{green}\}$ implying that $G_6 \cup H_6$ is not conflict-free; or $\text{col}(H_6) = \{\text{blue}, \text{green}\}$ implying that $G_4 \cup H_6$ is not conflict-free. A similar argument holds when J_3 or J_4 is yellow.
2. Two intervals among J_1, J_2, J_3, J_4 use yellow. It follows that H_4 contains two yellow intervals and the remaining two intervals are colored either $\{\text{red}, \text{blue}\}$, implying that $G_2 \cup H_4$ is not conflict-free; or $\{\text{red}, \text{green}\}$, implying that $G_6 \cup H_4$ is not conflict-free; or $\{\text{blue}, \text{green}\}$, implying that $G_4 \cup H_4$ is not conflict-free. ◀

Now we place $\Omega(n)$ of these gadgets in two groups and for simplicity assume a gadget has width of 1. The gadgets in the first group are spaced with distance 2 between them, so a gadget from the second group can fit between any two consecutive gadgets. In the second group the gadgets are spaced with distance $3n$ between them, so that all gadgets of the first group fit between them. All gadgets of the first group then move at the same speed, starting somewhere to the left of the second group and moving to the right. The gadgets of the second group remain stationary. These motions ensure that each gadget of first group will cross each gadget of the second group, generating $\Omega(n^2)$ crossing events, each of which results in at least one recoloring by Lemma 10.

▶ **Theorem 11.** *For any $n > 0$, there is a set of $8n$ intervals, each of which is either stationary or moves with unit speed to the right, so that when coloring the intervals using four colors at least n^2 recolorings are required to maintain a conflict-free coloring.*

References

1 M.A. Abam, M.J. Rezaei Seraji, and M. Shadravan. Online conflict-free coloring of intervals. *Scientia Iranica*, 21(6):2138–2141, 2014. URL: http://scientiairanica.sharif.edu/article_3607.html.

2 A. Bar-Noy, P. Cheilaris, S. Olonetsky, and S. Smorodinsky. Online conflict-free colouring for hypergraphs. *Combinatorics, Probability & Computing*, 19(4):493–516, 2010. doi:10.1017/S0963548309990587.

3 A. Bar-Noy, P. Cheilaris, and S. Smorodinsky. Deterministic conflict-free coloring for intervals: From offline to online. *ACM Trans. Algorithms*, 4(4):44:1–44:18, 2008. doi:10.1145/1383369.1383375.

4 Andreas Bärtschi and Fabrizio Grandoni. On conflict-free multi-coloring. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2015. doi:10.1007/978-3-319-21840-3_9.

5 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed. edition, 2008.

6 M. de Berg, T. Leijsen, A. Markovic, A. van Renssen, M. Roeloffzen, and G. Woeginger. Dynamic and kinetic conflict-free coloring of intervals with respect to points. *CoRR*, 2016. URL: http://arxiv.org/abs/1701.03388.

7 P. Cheilaris, L. Gargano, A.A. Rescigno, and S. Smorodinsky. Strong conflict-free coloring for intervals. *Algorithmica*, 70(4):732–749, 2014. doi:10.1007/s00453-014-9929-x.

8 K. Chen. How to play a coloring game against a color-blind adversary. In *Proc. 22nd ACM Symp. Comput. Geom.*, pages 44–51, 2006. doi:10.1145/1137856.1137865.

9 T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

10 G. Even, Z. Lotker, D. Ron, and S. Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003. doi:10.1137/S0097539702431840.

11 A. Fiat, M. Levy, J. Matousek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, U. Wagner, and E. Welzl. Online conflict-free coloring for intervals. In *Proc. 16th ACM-SIAM Symp. Discr. Alg.*, pages 545–554, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070506.

12 S. Har-Peled and S. Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry*, 34(1):47–70, 2005. doi:10.1007/s00454-005-1162-6.

13 E. Horev, R. Krakovski, and S. Smorodinsky. Conflict-free coloring made stronger. In *Proc. 12th Scandinavian Workshop. Alg. Theory*, pages 105–117, 2010.

14 S. Smorodinsky. *Combinatorial Problems in Computational Geometry*. PhD thesis, Tel-Aviv University, 2003.

15 S. Smorodinsky. Conflict-free coloring and its applications. In *Geometry — Intuitive, Discrete, and Convex: A Tribute to László Fejes Tóth*, pages 331–389. Springer Berlin Heidelberg, 2013.

Dynamic Conflict-Free Colorings in the Plane[*]

Mark de Berg[†1] and Aleksandar Markovic[‡2]

1 TU Eindhoven, Eindhoven, The Netherlands
mdberg@win.tue.nl

2 TU Eindhoven, Eindhoven, The Netherlands
a.markovic@tue.nl

Abstract

We study dynamic conflict-free colorings in the plane, where the goal is to maintain a conflict-free coloring (CF-coloring for short) under insertions and deletions.

- First we consider CF-colorings of a set S of unit squares with respect to points. Our method maintains a CF-coloring that uses $O(\log n)$ colors at any time, where n is the current number of squares in S, at the cost of only $O(\log n)$ recolorings per insertion or deletion We generalize the method to rectangles whose sides have lengths in the range $[1, c]$, where c is a fixed constant. Here the number of used colors becomes $O(\log^2 n)$. The method also extends to arbitrary rectangles whose coordinates come from a fixed universe of size N, yielding $O(\log^2 N \log^2 n)$ colors. The number of recolorings for both methods stays in $O(\log n)$.
- We then present a general framework to maintain a CF-coloring under insertions for sets of objects that admit a unimax coloring with a small number of colors in the static case. As an application we show how to maintain a CF-coloring with $O(\log^3 n)$ colors for disks (or other objects with linear union complexity) with respect to points at the cost of $O(\log n)$ recolorings per insertion. We extend the framework to the fully-dynamic case when the static unimax coloring admits weak deletions. As an application we show how to maintain a CF-coloring with $O(\sqrt{n} \log^2 n)$ colors for points with respect to rectangles, at the cost of $O(\log n)$ recolorings per insertion and $O(1)$ recolorings per deletion.

These are the first results on fully-dynamic CF-colorings in the plane, and the first results for semi-dynamic CF-colorings for non-congruent objects.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Conflict-free colorings, Dynamic data structures

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.27

1 Introduction

Consider a set of base stations in the plane that can be used for mobile communication. To ensure a good coverage, the base stations are typically positioned in such a way that the communication ranges of different base stations overlap. However, if a user is within range of several base stations using the same frequency, then interference occurs and the communication is lost. Therefore, we want to assign frequencies to the base stations such that any user within range of at least one base station, is also within range of at least one

[*] A full version of the paper is available at [12], `https://arxiv.org/abs/1709.10466`.

[†] MdB is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.

[‡] AM is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 27; pp. 27:1–27:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

base station using an interference-free frequency. The easy solution would be to give all stations a different frequency. However, this is undesirable as the set of available frequencies is limited. The question then arises: how many different frequencies are needed to ensure that any user that is within range of at least one base station has an interference-free base station at his disposal? Motivated by this and other applications, Even *et al.* [14] and Smorodinsky [17] introduced the notion of *conflict-free colorings* or *CF-colorings* for short. Here the ranges of the base stations are modeled as regions (disks, or other objects) in the plane, and frequencies are represented by colors. A CF-coloring is now defined as follows.

Let S be a set of objects in the plane. For a point $q \in \mathbb{R}^2$, let $S_q := \{S \in S | q \in S\}$ be the subset of objects containing q. A coloring $col : S \to \mathbb{N}$ of the objects in S – here we identify colors with non-negative integers – is said to be *conflict-free (with respect to points)* if for each point q with $S_q \neq \emptyset$ there is an object $S \in S_q$ whose color is unique among the objects in S_q. A CF-coloring is called *unimax* when the maximum color in S_q is unique.

We can also consider a dual version of planar CF-colorings. Here we are given a set S of points and a family $\mathcal{F}$ of geometric ranges, and the goal is to color the points in S such that any range from $\mathcal{F}$ containing a least one point, contains a point with a unique color.

Conflict-free colorings have received a lot of attention since they were introduced by Even *et al.* [14] and Smorodinsky [17]; see the overview paper by Smorodinsky [18], which surveys the work up to 2010. We review the work most relevant to our results.

Even *et al.* proved that it is always possible to CF-color a set of disks in the plane using $O(\log n)$ colors, and that $\Omega(\log n)$ colors are needed in the worst case. The authors extended the result to sets of translates of any given centrally symmetric polygon. Later, Har-Peled and Smorodinsky [15] further generalized the result to regions with near-linear union complexity. The dual version of the problem was also studied by Even *et al.* [14]; they showed it is possible to CF-color points using $O(\log n)$ colors with respect to disks, or with respect to scaled translations of a centrally symmetric convex polygon. Moreover, Ajwani *et al.* [1] showed how to CF-color points with respect to using $O(n^{0.382})$ colors.

Recall that CF-colorings correspond to interference-free frequency assignments in a cellular network. When a node in the network fails, the resulting assignment may no longer be interference-free. This leads to the study of *k-fault-tolerant* CF-colorings, where we want $\min(k, |S_q|)$ objects from S_q to have a unique color. In other words, a k-fault-tolerant CF-coloring allows the deletion of k objects without losing the conflict-free property. Cheilaris *et al.* [5] studied the 1-dimensional case, and presented a polynomial-time algorithm with approximation ratio $5 - \frac{2}{k}$ for the problem of finding a CF-coloring with a minimum number of colors. For $k = 1$ – that is, the regular CF-coloring – the algorithm gives a 2-approximation. Horev *et al.* [16] studied the 2-dimensional case and proved a $O(k \log n)$ bound for disks and, more generally, regions with near-linear union complexity.

To increase coverage or capacity in a cellular network it may be necessary to increase the number of base stations. This led Fiat *et al.* [7] to study *online* CF-colorings. Here the objects to be CF-colored arrive over time, and as soon as an object appears it must receive a color which cannot be changed later on. For CF-coloring points with respect to intervals, they proposed a deterministic algorithm using $O(\log^2 n)$ colors as well as two randomized algorithms, one of which is using at most $O(\log n \log \log n)$ colors in expectation and always producing a valid coloring. Later, Chen *et al.* [6] improved the bound with an algorithm using an expected $O(\log n)$ colors.

Chen *et al.* [8] considered the problem of online CF-coloring of points with respect to geometric ranges. They showed that for ranges that are half-planes, unit disks, or bounded-size rectangles – i.e. rectangles whose heights and widths all lie in the range $[1, c]$, for some fixed constant c – there is an online CF-coloring using $O(\log n)$ colors with high probability.

For bounded-size rectangles they also presented a deterministic result using $O(\log^3 n)$ colors. Bar-Noy *et al.* [3] provided a general strategy for online CF-coloring of hypergraphs. Their method uses $O(k \log n)$ colors with high probability, where k is the so-called *degeneracy* of the hypergraph. Their method can for instance be applied for points with respect to half-planes using $O(\log n)$ colors, which implies [8] the same result for unit disks with respect to points. They also introduced a deterministic algorithm for points with respect to intervals in $\mathbb{R}^1$ if *recolorings* are allowed. Their method uses at most $n - \log n$ recolorings in total; they did not obtain a bound on the number of recolorings for an individual insertion. Note that the results for online colorings in $\mathbb{R}^2$ are rather limited: for the primal version of the problem – online CF-coloring objects with respect to points – there are essentially only results for unit disks or unit squares (where the problem is equivalent to the dual version of coloring points with respect to unit disks and unit squares, respectively). Moreover, most of the results are randomized.

De Berg *et al.* [11] introduced the *fully dynamic* variant of the CF-coloring problem, which generalizes and extends the fault-tolerant and online variants. Here the goal is to maintain a CF-coloring under insertions and deletions. It is easy to see that if we allow deletions and we do not recolor objects, we may need to give each object in S its own color. Using n colors is clearly undesirable. On the other hand, recoloring all objects after each update – using then the same number of colors as in the static case – is not desirable either. Thus the main question is which trade-offs can we get between the number of colors and the number of recolorings? De Berg *et al.* proved a lower bound on this trade-off for the 1-dimensional problem of CF-coloring intervals with respect to points. (For this case it is straightforward to give a static CF-coloring with only three colors.) Their lower bound implies that if we want $O(1/\varepsilon)$ colors, we must sometimes re-color $\Omega(\varepsilon n^{\varepsilon})$ intervals, and that if we allow only $O(1)$ recolorings we must use $\Omega(\log n / \log \log n)$ colors in the worst case. They also presented a strategy that uses $O(\log n)$ colors at the cost of $O(\log n)$ recolorings. The main goal of our paper is to study fully dynamic CF-colorings for the 2-dimensional version of the problem.

Our contributions. In Section 2 we give an algorithm for CF-coloring unit squares using $O(\log n)$ colors and $O(\log n)$ recolorings per update. Note that $\Omega(\log n)$ is a lower bound on the number of colors for a CF-coloring of unit squares even in the static case, so the number of colors our fully dynamic method uses is asymptotically optimal. We also present an adaptation for bounded-size rectangles which uses $O(\log^2 n)$ colors. The method also extends to arbitrary rectangles whose coordinates come from a fixed universe of size N, yielding $O(\log^2 N \log^2 n)$ colors. Both methods still use $O(\log n)$ recolorings per update. These constitute the first results on fully-dynamic CF-colorings in $\mathbb{R}^2$.

In Section 3, we give two general approaches that can be applied in many cases. The first uses a static coloring to solve insertions-only instances.

It can be applied in settings where the static version of the problem admits a unimax coloring with a small number of colors. The method can for example be used to maintain a CF-coloring for pseudodisks with $O(\log^3 n)$ colors and $O(\log n)$ recolorings per update, or to maintain a CF-coloring for fat regions. This is the first result for the semi-dynamic CF-coloring problem for such objects: previous online results for coloring objects with respect to points in $\mathbb{R}^2$ only applied to unit disks or unit squares. We extend the method to obtain a fully-dynamic solution, when the static solution allows what we call weak deletions. We can apply this technique for instance to CF-coloring points with respect to rectangles, using $O(\sqrt{n} \log^2 n)$ colors and $O(\log n)$ recolorings per insertion and $O(1)$ recolorings per deletion.

2 Dynamic CF-Colorings for Unit Squares and Rectangles

In this section we explain how to color unit squares with $O(\log n)$ colors and $O(\log n)$ recolorings per update. We then generalise this coloring to bounded-size rectangles, and to rectangles with coordinates from a fixed universe. We first explain our basic technique on so-called anchored rectangles.

2.1 A Subroutine: Maintaining a CF-coloring for Anchored Rectangles

We say that a rectangle r is *anchored* if its bottom-left vertex lies at the origin. Let S be a set of n anchored rectangles. We denote the x- and y-coordinate of the top-right vertex of a rectangle r by r_x and r_y, respectively. Our CF-coloring of S is based on an augmented red-black tree, as explained next.

To simplify the description we assume that the x-coordinates of the top-right vertices (and, similarly, their y-coordinates) are all distinct – extending the results to degenerate cases is straightforward. We store S in a red-black tree $\mathcal{T}$ where r_x (the x-coordinate of the top-right vertex of r) serves as the key of the rectangles $r \in S$. It is convenient to work with a *leaf-oriented* red-black tree, where the keys are stored in the leaves of the tree and the internal nodes store splitting values.[1] We can assume without loss of generality that the splitting values lie strictly in between the keys.

For a node $v \in \mathcal{T}$, let $\mathcal{T}_v$ denote the subtree rooted at v and let $S(v)$ denote the set of rectangles stored in the leaves of $\mathcal{T}_v$. We augment $\mathcal{T}$ by storing a rectangle $r_{\max}(v)$ at every (leaf or internal) node of v, define as follows:

$$r_{\max}(v) := \text{the rectangle } r \in S(v) \text{ that maximizes } r_y.$$

Let *left*(v) and *right*(v) denote the left and right child, respectively, of an internal node v. Notice that $r_{\max}(v)$ is the rectangle whose top-right vertex has maximum y-value among $r_{\max}(\mathit{left}(v))$ and $r_{\max}(\mathit{right}(v))$, so $r_{\max}(v)$ can be found in $O(1)$ time from the information at v's children. Hence, we can maintain the extra information in $O(\log n)$ time per insertion and deletion [9].

Next we define our coloring function. To this end we define for each rectangle $r \in S$ a set $N(r)$ of nodes in $\mathcal{T}$, as follows.

$$N(r) := \{v \in \mathcal{T} : \ v \text{ is the leaf storing } r, \text{ or } v \text{ is an internal node with } r_{\max}(\mathit{right}(v)) = r\}.$$

Observe that $N(r)$ only contains nodes on the search path to the leaf storing r and that $N(r) \cap N(r') = \emptyset$ for any two rectangles $r, r' \in S$. Let $\text{height}(v)$ denote the height of $\mathcal{T}_v$. Thus $\text{height}(v) = 0$ when v is a leaf, and for non-leaf nodes v we have $\text{height}(v) = \max(\text{height}(\mathit{left}(v)), \text{height}(\mathit{right}(v)) + 1$. We now define the color of a rectangle $r \in S$ as $\mathit{col}(r) := \max_{v \in N(r)} \text{height}(v)$. Since $N(r)$ always contains at least one node, namely the leaf storing r, this is a well-defined coloring.

▶ **Lemma 1.** *The coloring defined above is conflict-free.*

▶ **Theorem 2.** *Let S be a set of anchored rectangles in the plane. Then it is possible to maintain a CF-coloring on S with $O(\log n)$ colors using $O(\log n)$ recolorings per insertions and deletion, where n is the current number of rectangles in S.*

[1] Such a leaf-oriented red-black tree can be seen as a regular red-black tree on a set $X'(S)$ that contains a splitting value between any two consecutive keys. Hence, all the normal operations can be done in the standard way.

Proof. Consider the coloring method described above, which by Lemma 1 is conflict-free. Since red-black trees have height $O(\log n)$, the number of colors used is $O(\log n)$ as well.

Now consider an update on S. The augmented red-black tree can be updated in $O(\log n)$ time in a standard manner [9]. The color of a rectangle $r \in S$ can only change when (i) the set $N(r)$ changes, or (ii) the height of a node in $N(r)$ changes. We argue that this only happens for $O(\log n)$ rectangles. Consider an insertion; the argument for deletions is similar. In the first phase of the insertion algorithm for red-black trees [9] a new leaf is created for the rectangle to be inserted. This may change height(v) or $r_{\max}(v)$ only for nodes v on the path to this leaf, so it affects the color of $O(\log n)$ rectangles. In the second phase the balance is restored using $O(1)$ rotations. Each rotation changes height(v) or $r_{\max}(v)$ for only $O(\log n)$ nodes, so also here only $O(\log n)$ rectangles are affected. ◀

2.2 Maintaining a CF-Coloring for Unit Squares

Let S be a set of unit squares. We first assume that all squares in S contain the origin.

A naive way to use the result from the previous section is to partition each square $s \in S$ into four rectangular parts by cutting it along the x-axis and the y-axis. Note that the set of north-east rectangle parts (i.e., the parts to the north-east of the origin) are all anchored rectangles, so we can use the method described above to maintain a CF-coloring on them. The other part types (south-east, south-west, and north-west) can be treated similarly. Thus every square $s \in S$ receives four colors. If we now assign a final color to s that is the four-tuple consisting of those four colors, then we obtain a CF-coloring with $O(\log^4 n)$ colors. (This trick of using a "product color" was also used by, among others, Ajwani *et al.* [1].)

It is possible to improve this by using the following fact: the ordering of the x-coordinates of the top-right corners of the squares in S is the same as the ordering of their bottom-right (or bottom-left, or top-left) corners. This implies that instead of working with four different trees we can use the same tree structure for all part types. Moreover, even the extra information stored in the internal nodes is the same for the north-east and north-west parts, since the y-coordinates of the top-right and top-left vertices are the same. Similarly, the extra information for the south-east and south-west parts are the same. Therefore, we can modify the augmented red-black tree to store two squares per internal node instead of one:

- $s_{\max}(v) :=$ the square $s \in S(v)$ that maximizes s_y,
- $s_{\min}(v) :=$ the square $s \in S(v)$ that minimizes s_y.

Next we modify our coloring function. Therefore we first redefine the set $N(s)$ of nodes for each square $s \in S$:

$$N(s) := \{\text{the leaf storing } s\} \cup N_{\text{NE}}(s) \cup N_{\text{SE}}(s) \cup N_{\text{SW}}(s) \cup N_{\text{NW}}(s),$$

where

- $N_{\text{NE}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\max}(\mathit{right}(v)) = s\}$,
- $N_{\text{SE}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\min}(\mathit{right}(v)) = s\}$,
- $N_{\text{SW}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\min}(\mathit{left}(v)) = s\}$,
- $N_{\text{NW}}(s) := \{v \in \mathcal{T} : v \text{ is an internal node with } s_{\max}(\mathit{left}(v)) = s\}$.

The coloring is as follows. We now allow four colors per height-value, namely for height-value h we give colors $4h + j$ for $j \in \{0, 1, 2, 3\}$. These colors essentially correspond to the colors we would give out for the four part types. The color of a square s is now defined as

$$\mathit{col}(s) := \begin{cases} 0 & \text{if } \max\limits_{v \in N(s)} \text{height}(v) = 0 \ (s \text{ is only stored at a leaf}), \\ 4 \cdot \max\limits_{v \in N(s)} \text{height}(v) + j & \text{if } \max\limits_{v \in N(s)} \text{height}(v) > 0, \end{cases}$$

where

$$j := \begin{cases} 0 & \text{if } \mathrm{height}(s) = \max_{v \in N_{\mathrm{NE}}(s)} \mathrm{height}(v), \\ 1 & \text{if the condition for } j = 0 \text{ does not apply and } \mathrm{height}(s) = \max_{v \in N_{\mathrm{SE}}(s)} \mathrm{height}(v), \\ 2 & \text{if the conditions for } j = 0, 1 \text{ do not apply and } \mathrm{height}(s) = \max_{v \in N_{\mathrm{SW}}(s)} \mathrm{height}(v), \\ 3 & \text{otherwise (we now must have } \mathrm{height}(s) = \max_{v \in N_{\mathrm{NW}}(s)} \mathrm{height}(v)). \end{cases}$$

The following lemma can be proven in exactly the same was ay Lemma 1. The only addition is that, when considering a set $S(v)$, we need to make a distinction depending on in which quadrant the query point q lies. If it lies in the north-east quadrant we can follow the proof verbatim, and the other cases are symmetric.

▶ **Lemma 3.** *The coloring defined above is conflict-free.*

It remains to remove the restriction that all squares contain the origin. To this end we use a grid-based method, similar to the one used by, e.g., Chen *et al.* [8]. Consider the integer grid, and assign each square in S to the grid point it contains; if a square contains multiple grid points, we assign it to the lexicographically smallest one. Thus we create for each grid point (i, j) a set $S(i, j)$ of squares that all contain the point (i, j). We maintain a CF-coloring for each such set using the method described above. Note that a square in $S(i, j)$ can only intersect squares in $S(i', j')$ when (i', j') is one of the eight neighboring grid points of (i, j). Hence, when $i' = i \mod 2$ and $j' = j \mod 2$ we can re-use the same color set, and so we only need four color sets of $O(\log n)$ colors each.

▶ **Theorem 4.** *Let S be a set of unit squares in the plane. Then it is possible to maintain a CF-coloring on S with $O(\log n)$ colors using $O(\log n)$ recolorings per insertions and deletion, where n is the current number of squares in S.*

2.3 Maintaining a CF-Coloring for Bounded-Size Rectangles

Let S be a set of *bounded-size rectangles*: rectangles whose widths and heights are between 1 and c for some fixed constant c. Note that in practice, two different base stations have roughly the same coverage, hence it makes sense to assume the ratio is bounded by some constant c.

First consider the case where all rectangles in S contain the origin. Here, the x-ordering of the top-right corners of the rectangles may be different from the x-ordering of the top-left corners as we no longer use unit squares. Therefore the trees for the east (that is, north-east and south-east) parts no longer have the same structure. Note that the x-ordering of the top-left and bottom-left corners are the same, hence only one tree suffices for the east parts, and the same holds for the west parts. Hence, we build and maintain two separate trees, one for the east parts of the rectangles and one for the west parts. In the east tree we only work with the sets $N_{\mathrm{NE}}(s)$ and $N_{\mathrm{SE}}(s)$, and in the west tree we only work with $N_{\mathrm{SW}}(s)$ and $N_{\mathrm{NW}}(s)$; for the rest the structures and colorings are defined in the same as before. We then use the product coloring to obtain our bound: we give each rectangle a pair of colors – one coming from the east tree, one coming from the west tree – resulting in $O(\log^2 n)$ different color pairs.

To remove the restriction that each rectangle contains the origin we use the same grid-based approach as for unit squares. The only difference is that a rectangle in a set $S(i, j)$ can now intersect rectangles from up to $(1 + 2c)^2 - 1$ sets $S(i', j')$, namely with $i - c \leqslant i' \leqslant i + c$ and $j - c \leqslant j' \leqslant j + c$. Since c is a fixed constant, we still need only $O(1)$ color sets.

▶ **Theorem 5.** *Let S be a set of bounded-size rectangles in the plane. Then it is possible to maintain a CF-coloring on S with $O(\log^2 n)$ colors using $O(\log n)$ recolorings per insertion and deletion, where n is the current number of rectangles in S.*

2.4 Maintaining a CF-Coloring for Rectangles with Coordinates from a Fixed Universe

The solution can also be extended to rectangles of arbitrary sizes, if their coordinates come from a fixed universe $U := \{0, \ldots, N-1\}$ of size N. Again, from a practical point of view it makes sense as in a city for instance, the places a base station can be created are limited.

To this end we construct a balanced tree $\mathcal{T}_x$ over the universe U, and we associate each rectangle $r = [r_{x,1}, r_{x,2}] \times [r_{y,1}, r_{y,2}]$ to the highest node v in $\mathcal{T}_x$ whose x-value $x(v)$ is contained in $[r_{x,1}, r_{x,2}]$. Let $S(v)$ be the set of objects associated to v. For each node $v \in \mathcal{T}_x$ we construct a balanced tree $\mathcal{T}_y(v)$ over the universe, and we associate each rectangle $r \in S(v)$ to the highest node w in $\mathcal{T}_y(v)$ whose y-value $y(w)$ is contained in $[r_{y,1}, r_{y,2}]$. (In other words, we are constructing a 2-level interval tree [10] on the rectangles, using the universe to provide the skeleton of the tree. The reason for using a skeleton tree is that otherwise we have to maintain balance under insertions and deletions, which is hard to do while ensuring worst-case bounds on the number of recolorings.) Let $S(w)$ be the set of objects associated to a node w in any second-level tree $\mathcal{T}_y(v)$. All rectangles in $S(w)$ have a point in common, namely the point $(x(v), y(w))$. Therefore we can proceed as in the previous section, and maintain a CF-coloring on $S(w)$ with a color set of size $O(\log^2 n)$, using $O(\log n)$ recolorings per insertions and deletion.

Note that for any two nodes w, w' at the same level in a tree $\mathcal{T}_y(v)$, any two rectangles $r \in S(w)$ and $r' \in S(w')$ are disjoint. Hence, over all nodes $w \in \mathcal{T}_x(v)$ we only need $O(\log N)$ different color sets. Similarly, for any two nodes v, v' of $\mathcal{T}_x$ at the same level, any two rectangles $r \in S(v)$ and $r' \in S(v')$ are disjoint. Hence, the total number of color sets we need is $O(\log^2 N)$. This leads to the following result.

▶ **Theorem 6.** *Let S be a set of rectangles in the plane, whose coordinates come from a fixed universe of size N. Then it is possible to maintain a CF-coloring on S with $O(\log^2 N \log^2 n)$ colors using $O(\log n)$ recolorings per insertions and deletion, where n is the current number of rectangles in S.*

▶ Remark. Instead of assuming a skeleton tree and working with a fixed skeleton for our 2-level interval tree, we can also use randomized search trees. Then, assuming the adversary doing the insertions and deletions is oblivious of our structure and coloring, the tree is expected to be balanced at any point in time. Hence, we obtain $O(\log^4 n)$ colors in expectation, at the cost of $O(\log n)$ recolorings (worst-case) per update.

3 A General Technique

In this section we present a general technique to obtain a dynamic CF-coloring scheme in cases where there exists a static unimax coloring. (Recall that a unimax coloring is a CF-coloring where for any point q the object from S_q with the maximum color is unique.) Our technique results in a dynamic CF-coloring that uses $O(\gamma_{\mathrm{um}}(n) \log^2 n)$ colors, where $\gamma_{\mathrm{um}}(n)$ is the number of colors used in the static unimax coloring, at the cost of $O(\log n)$ recolorings per update. We first describe our technique for the case of insertions only. Then we extend the technique to the fully-dynamic setting, for the case where the unimax coloring allows for so-called weak deletions.

We remark that even though we describe our technique in the geometric setting in the plane, the techniques provided in this section can be applied in the abstract hypergraph setting as well.

3.1 An Insertion-Only Solution

Let S be a set of objects in the plane and assume that S can be colored in a unimax fashion using $\gamma_{\mathrm{um}}(n)$ colors, where γ_{um} is a non-decreasing function. Here it does not matter if S is a set of geometric objects that we want to CF-color with respect to points, or a set of points that we want to CF-color with respect to a family of geometric ranges. For concreteness we refer to the elements from S as objects.

Our technique to maintain a CF-coloring under insertions of objects into S is based on the *logarithmic method* [4], which is also used to make static data structures semi-dynamic. Thus at any point in time we have $\lceil \log n \rceil + 1$ sets S_i such that each set S_i, for $i = 0, \ldots, \lceil \log n \rceil$, is either empty or contains exactly 2^i objects. The idea is to give each set S_i its own color set, consisting of $\gamma_{\mathrm{um}}(2^i)$ colors. Maintaining a CF-coloring under insertions such that the *amortized* number of recolorings is small, is easy (and it does not require the coloring to be unimax): when inserting a new object we find the first empty set S_i, and we put all objects in $S_0 \cup \cdots \cup S_{i-1}$ together with the new object into S_i. The challenge is to achieve a *worst-case* bound on the number of recolorings per insertion. Note that for the maintenance of data structures, it is known how to achieve worst-case bounds using the logarithmic method. The idea is to build the new data structure for S_i "in the background" and switch to the new structure when it is ready. For us this does not work, however, since we would still need many recolorings when we switch. Hence, we need a more careful approach.

When moving all objects from $S_0 \cup \cdots \cup S_{i-1}$ (together with the new object) into S_i, we do not recolor them all at once but we do so over the next 2^i insertions. As long as we still need to recolor objects from S_i, we say that S_i is *in migration.* We need to take care that the coloring of a set that is in migration, where some objects still have the color from the set S_j they came from and others have already received their new color in S_i, is valid. For this we need to recolor the objects in a specific order, which requires the static coloring to be unimax as explained below. Another complication is that, because the objects in S_j that are being moved to S_i still have their own color, we have to be careful when we create a new set S_j. To avoid any problems, we need several color sets per set. Next we describe our scheme in detail.

As already mentioned, we have sets $S_0, \ldots, S_\ell$, where $\ell := \lceil \log n \rceil$. Each set can be in one of three states: *empty*, *full*, or *in migration.* For each i with $0 \leqslant i \leqslant \ell$ we have $\ell - i + 1$ color sets of size $\gamma_{\mathrm{um}}(2^i)$ available, denoted by $\mathcal{C}(i,t)$ for $0 \leqslant t \leqslant \ell - i$. The insertion of an object s into S now proceeds as follows.

1. Let i be the smallest index such that S_i is empty. Note that i might be $\ell + 1$, in which case we introduce a new set and redefine ℓ. Note that this only happens when the number of objects reaches a power of 2.
2. Set $S_i := \{s\} \cup S_0 \cup \cdots \cup S_{i-1}$. Mark $S_0, \ldots, S_{i-1}$ as *empty*, and mark S_i as *in migration.*
3. Take an unused color set $\mathcal{C}(i,t)$ – we argue below that at least one color set $\mathcal{C}(i,t)$ with $0 \leqslant t \leqslant \ell - i$ is currently unused – and compute a unimax coloring of S_i using colors from $\mathcal{C}(i,t)$. We refer to the color from $\mathcal{C}(i,t)$ that an object in S_i receives as its *final color* (for the current migration). Except for the newly inserted object s, we do not recolor any objects to their final color in this step; they all keep their current colors.

4. For each set S_k in migration – this includes the set we just created in Step 3 – we recolor one object whose final color is different from its current color and whose final color is maximal among such objects. When multiple objects share that property, we arbitrarily choose one of them. If all objects in S_k now have their final color, we mark S_k as *full*.

▶ **Lemma 7.** *Suppose that when we insert an object s into S, the first empty set is S_i. Then the sets $S_0, \ldots, S_{i-1}$ are full.*

Proof. Suppose for a contradiction that S_j, for some $0 \leqslant j < i$ is in migration. Consider the last time at which S_j was created – that is, the last time at which we inserted an object s' that caused the then-empty set S_j to be created and marked as in migration. Upon insertion of s', we already perform one recoloring in S_j. At that point all sets $S_0, \ldots, S_{j-1}$ were marked empty and it takes $\sum_{t=0}^{j-1} 2^t = 2^j - 1$ additional insertions to fill them, giving us as many recolorings in S_j. Thus before we create any set S_i with $i > j$, we have already marked S_j as full. Since s' was the last object whose insertion created S_j, by the time we create S_i the set S_j must still be full – it cannot in the mean time have become empty and later be re-created (and thus be in migration). ◀

Next we show that in Step 3 we always have an unused color set at our disposal.

▶ **Lemma 8.** *When we create a new set S_i in Step 3, at least one of the color sets $\mathcal{C}(i,t)$ with $0 \leqslant t \leqslant \ell - i$ is currently unused.*

Proof. Consider a color set $\mathcal{C}(i,t)$. The reason we may not be able to use $\mathcal{C}(i,t)$ when we create S_i is that there is a set $S_{i'}$ with $i' > i$ that is currently in migration: the objects from a previous instance of S_i (that were put into $S_{i'}$ when we created $S_{i'}$) may not all have been recolored yet. By Lemma 7 this previous instance was full when it was put into $S_{i'}$ and so it only blocks a single color set, namely one for S_i. Thus the number of color sets $\mathcal{C}(i,t)$ currently in use is at most $\ell - i$. Since we have $\ell - i + 1$ such colors sets at our disposal, one must be unused. ◀

▶ **Theorem 9.** *Let $\mathcal{F}$ be a family of objects such that any subset of n objects from $\mathcal{F}$ admits a unimax coloring with $\gamma_{\text{um}}(n)$ colors, where $\gamma_{\text{um}}(n)$ is non-decreasing. Then we can maintain a CF-coloring on a set S of objects from $\mathcal{F}$ under insertions, such that the number of used colors is $O(\gamma_{\text{um}}(n) \log^2 n)$ and the number of recolorings per insertion is at most $\lceil \log n \rceil$, where n is the current number of objects in S.*

Proof. The number of colors used is $\sum_{i=0}^{\ell} (\ell - i + 1)\gamma_{\text{um}}(2^i)$, where $\ell = \lceil \log n \rceil$. Since $\gamma_{\text{um}}(n)$ is non-decreasing, this is bounded by $O(\gamma_{\text{um}}(n) \log^2 n)$. The number of recolorings per insertion is at most one per set S_i, so at most $\lceil \log n \rceil$ in total. (The total number of sets is actually $\lceil \log n \rceil + 1$, but not all of them can be in migration.)

It remains to prove that the coloring is conflict-free. Consider a point $q \in \mathbb{R}^2$. (Here we use terminology from CF-coloring of objects with respect to points. In the dual version, q would be a range.) Let S_i be a set containing an object s with $q \in s$; if no such set exists there is nothing to prove.

If S_i is full then it has a unimax coloring using a color set $\mathcal{C}(i,t)$ not used by any other set S_j. Hence, there is an object containing q with a unique color.

Now suppose that S_i is in migration. We have two cases: (i) q is contained in an object from S_i that has already received its final color, (ii) all objects in S_i containing q still have their old color.

In case (i) the object containing q with the highest final color must have a unique color, because of the following easy-to-prove fact.

▶ **Fact.** *Consider any set A colored with a unimax coloring. Let z be an integer, and let $B \subseteq A$ be a subset that contains all objects of color greater than z, some objects of color z, and at most one other object. Then the coloring of B is unimax.*

This fact proves the statement above for case (i), because we recolor the objects in decreasing order of their colors and the coloring we are migrating to is a unimax coloring. (The "at most one other object" mentioned in the fact is needed because the object that caused the migration immediately receives its color, and this color needs not be the highest color.)

In case (ii), q is contained in an object from some old set S_j with $j < i$. At the time we created S_i this set S_j was CF-colored, and since we did not yet recolor any object from S_j that contains q – otherwise we are in case (i) – we conclude that q is contained in an object with a unique color. ◀

Application: Objects with Near-Linear Union Complexity. Har-Peled and Smorodinsky [15] proved that any family of objects with linear union complexity (for example disks, or pseudodisks) can be colored in a unimax fashion using $O(\log n)$ colors. In fact, their result is more general: if the union complexity is at most $n \cdot \beta(n)$ then the number of colors is $O(\beta(n) \log n)$. Note that for disks and pseudodisks we have $\beta(n) = O(1)$, for fat triangles we have $\beta(n) = O(\log^* n)$ [2] and for locally fat objects we have $\beta(n) = O(2^{O(\log^* n)})$ [2]. This directly implies the following result.

▶ **Corollary 10.** *Let $\mathcal{F}$ be a family of objects such that the union complexity of any subset of n objects from $\mathcal{F}$ is at most $n\beta(n)$. Then we can maintain a CF-coloring on a set S of objects from $\mathcal{F}$ under insertions, such that the number of used colors is $O(\beta(n) \log^3 n)$ and the number of recolorings per insertion is $O(\log n)$, where n is the current number of objects in S.*

3.2 A Fully-Dynamic Solution

We now present a generalisation of our method to the fully-dynamic case. For lack of space we only give a brief sketch; details can be found in the full version [12]. As before, we assume we have a family $\mathcal{F}$ of objects such that any set of n objects from $\mathcal{F}$ can be unimax-colored with $\gamma_{\mathrm{um}}(n)$ colors. We further assume that such a coloring admits *weak deletions*: once we have colored a given set S of n_0 objects using $\gamma_{\mathrm{um}}(n_0)$ colors, we can delete objects from it using $r(n_0)$ recolorings per deletion such that the number of colors never exceeds $\gamma_{\mathrm{um}}(n_0)$. The functions $\gamma_{\mathrm{um}}(n)$ and $r(n)$ are assumed to be non-decreasing.

The insertions in this method are similar to those in the semi-dynamic solution, except that now a set S_i can be in four states: *empty*, *non-empty*, *in upwards migration*, and *in downwards migration*. It is worth pointing out that the upwards migrations are very similar to the migrations of the previous section, and that only S_ℓ (i.e., the last set) can be in downwards migration. The deletion procedure makes use of the weak deletions. Furthermore, since now a set S_i can have less than 2^i elements due to deletions, we need to make sure the number of set stays logarithmic. To that purpose, we make sure that the last set is at least half full. When this is no longer true, we combine the last three sets using a downwards migration, which is similar to the upwards migration. The details are fairly intricate and can be found in the full version [12]. We obtain the following theorem.

▶ **Theorem 11.** *Let $\mathcal{F}$ be a family of objects such that any subset of n objects from $\mathcal{F}$ admits a unimax coloring with $\gamma_{\mathrm{um}}(n)$ colors and that allows weak deletions at the cost of*

$r(n)$ recolorings, where $\gamma_{\text{um}}(n)$ and $r(n)$ are non-decreasing. Then we can maintain a CF-coloring on a set S of objects from $\mathcal{F}$ under insertions and deletions, such that the number of used colors is $O(\sum_{i=0}^{k} \gamma_{\text{um}}(2^i) \log n)$, where $k = \Theta(\log n)$. The number of recolorings per insertion is $O(\log n)$, and the number of recolorings per deletion is $O(r(8n) + 1)$, where n is the current number of objects in S.

Application: Points with Respect to Rectangles. We now make use of Theorem 11 to maintain a CF-coloring of points with respect to rectangles. But first we present a simple technique to color points with respect to intervals in $\mathbb{R}^1$, which we use as a subroutine.

▶ **Lemma 12.** *We can maintain a unimax coloring of n points in $\mathbb{R}^1$ with respect to intervals under deletions, using $\lceil \log n_0 \rceil$ colors and at the cost of one recoloring per deletion. Here n_0 is the initial number of points.*

Proof. We start with a static unimax coloring of points with respect to intervals using $\lceil \log n_0 \rceil$ colors [18]. Recoloring after deleting a point p with color i is done as follows. If both neighbors of p have a higher color then we do nothing, otherwise we pick a neighbor with color smaller than i and recolor it to i. To prove the coloring stays unimax we only need to consider intervals I containing a neighbor of p. Now consider $I \cup \{p\}$. If before the deletion the maximum color was larger than i then that color is still present and unique. Otherwise i was the unique maximum color. Now either I contains a neighbor of p that was recolored to i, or no point in I was recolored; in both cases the maximum color in I is unique. ◀

Let now S be a set of points in the plane and $\mathcal{F}$ be the family of all rectangles in the plane. The following lemma shows how to perform weak deletions.

▶ **Lemma 13.** *There is a conflict free coloring of n points with respect to rectangles using $O(\sqrt{n} \log n)$ colors that allows weak deletions at the cost of one recoloring per deletion.*

Proof. We first partition the point set into at most $\sqrt{n}$ subsets such that each set is monotone using Dilworth's theorem [13]. Then, each point set behaves exactly as points with respect to intervals in one dimension. We can then apply Lemma 12 to finish the proof. ◀

▶ **Corollary 14.** *Let S be a set of points in the plane and $\mathcal{F}$ be a family of rectangles. Then we can maintain a CF-coloring on S under insertions and deletions such that the number of used colors is $O(\sqrt{n} \log^2 n)$ and the number of recolorings per insertion is $O(\log n)$ and $O(1)$ per deletion, where n is the current number of objects in S.*

4 Concluding Remarks

We studied the maintenance of a CF-coloring under insertions and deletions, presenting the first fully-dynamic solution for objects in $\mathbb{R}^2$. We showed how to maintain a CF-coloring for unit squares and for bounded-size rectangles, with $O(\log n)$ resp. $O(\log^2 n)$ colors and $O(\log n)$ recolorings per update. The method extends to arbitrary rectangles with coordinates from a fixed universe of size N, yielding $O(\log^2 N \log^2 n)$ colors and $O(\log n)$ recolorings per update. We also presented general techniques for the semi-dynamic (insertions-only) and the fully-dynamic case (insertions and deletions). Our insertions-only technique can be applied to objects with near-linear union complexity, giving for instance a CF-coloring of $O(\log^3 n)$ colors for pseudodisks using $O(\log n)$ recolorings per update. Our fully-dynamic solution applies to any class of object on which weak deletions are possible, giving for instanct a CF-coloring of $O(\sqrt{n}\log^2 n)$ colors for points with respect to rectangles at the cost of $O(\log n)$ recolorings per insertion and $O(1)$ recolorings per deletion.

References

1 Deepak Ajwani, Khaled M. Elbassioni, Sathish Govindarajan, and Saurabh Ray. Conflict-free coloring for rectangle ranges using $o(n^{.382})$ colors. *Discrete & Computational Geometry*, 48(1):39–52, 2012. doi:10.1007/s00454-012-9425-5.

2 Boris Aronov, Mark de Berg, Esther Ezra, and Micha Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM Journal on Computing*, 43:534–572, 2014.

3 Amotz Bar-Noy, Panagiotis Cheilaris, Svetlana Olonetsky, and Shakhar Smorodinsky. Online conflict-free colouring for hypergraphs. *Combinatorics, Probability & Computing*, 19(4):493–516, 2010. doi:10.1017/S0963548309990587.

4 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

5 Panagiotis Cheilaris, Luisa Gargano, Adele A. Rescigno, and Shakhar Smorodinsky. Strong conflict-free coloring for intervals. *Algorithmica*, 70(4):732–749, 2014. doi:10.1007/s00453-014-9929-x.

6 Ke Chen. How to play a coloring game against a color-blind adversary. In *Proceedings of the 22nd ACM Symposium on Computational Geometry*, pages 44–51, 2006. doi:10.1145/1137856.1137865.

7 Ke Chen, Amos Fiat, Haim Kaplan, Meital Levy, Jiří Matoušek, Elchanan Mossel, János Pach, Micha Sharir, Shakhar Smorodinsky, Uli Wagner, and Emo Welzl. Online conflict-free coloring for intervals. *SIAM Journal on Computing*, 36(5):1342–1359, 2007. doi:10.1137/S0097539704446682.

8 Ke Chen, Haim Kaplan, and Micha Sharir. Online conflict-free coloring for halfplanes, congruent disks, and axis-parallel rectangles. *ACM Transactions on Algorithms*, 5(2):16:1–16:24, 2009. doi:10.1145/1497290.1497292.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

10 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed. edition, 2008.

11 Mark de Berg, Tim Leijsen, Aleksandar Markovic, André van Renssen, Marcel Roeloffzen, and Gerhard Woeginger. Dynamic and kinetic conflict-free coloring of intervals with respect to points. In *Proceedings of the 28th International Symposium on Algorithms and Computation*, 2017.

12 Mark de Berg and Aleksandar Markovic. Dynamic conflict-free colorings in the plane. *CoRR*, abs/1709.10466, 2017.

13 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. URL: http://www.jstor.org/stable/1969503.

14 Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003. doi:10.1137/S0097539702431840.

15 Sariel Har-Peled and Shakhar Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry*, 34(1):47–70, 2005. doi:10.1007/s00454-005-1162-6.

16 Elad Horev, Roi Krakovski, and Shakhar Smorodinsky. Conflict-free coloring made stronger. In *Proceedings ot the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 105–117, 2010. doi:10.1007/978-3-642-13731-0_11.

17 Shakhar Smorodinsky. *Combinatorial Problems in Computational Geometry*. PhD thesis, Tel-Aviv University, 2003.

18 Shakhar Smorodinsky. Conflict-free coloring and its applications. *CoRR*, abs/1005.3616, 2010. URL: `http://arxiv.org/abs/1005.3616`.

Temporal Hierarchical Clustering*†

Tamal K. Dey[1], Alfred Rossi[2], and Anastasios Sidiropoulos[3]

1 **Department of Computer Science and Engineering, The Ohio State University, Columbus, USA**
dey.8@osu.edu

2 **Department of Computer Science and Engineering, The Ohio State University, Columbus, USA**
rossi.49@osu.edu

3 **Department of Computer Science and Engineering, The Ohio State University, Columbus, USA**
sidiropoulos.1@osu.edu

Abstract

We study hierarchical clusterings of metric spaces that change over time. This is a natural geometric primitive for the analysis of dynamic data sets. Specifically, we introduce and study the problem of finding a temporally coherent sequence of hierarchical clusterings from a sequence of unlabeled point sets. We encode the clustering objective by embedding each point set into an ultrametric space, which naturally induces a hierarchical clustering of the set of points. We enforce temporal coherence among the embeddings by finding correspondences between successive pairs of ultrametric spaces which exhibit small distortion in the Gromov-Hausdorff sense. We present both upper and lower bounds on the approximability of the resulting optimization problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms, I.5.3 Clustering

Keywords and phrases clustering, hierarchical clustering, multi-objective optimization, dynamic metric spaces, moving point sets, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.28

1 Introduction

Clustering is a primitive in data analysis which simultaneously serves to summarize data and elucidate its hidden structure. In its most common form a clustering problem consists of a pair (P, k), where P is a metric space, and k indicates the desired number of clusters. The goal of the problem is to try to find a partition of the points of P into k sets such that some objective is minimized. Because of the fundamental nature of such a primitive, clustering enjoys broad application in a variety of settings and an extensive body of work exists to explain, refine, and adapt its methodology [3, 8, 12, 14, 18, 19].

Having to decide the number of clusters in advance can be a source of difficulty in practice. When faced with this problem, one common approach is to use hierarchical clustering to produce a parameter free summary of the input. That is, instead of producing a single partition of the input points, the goal is to find a rooted tree (called a *dendrogram*) where the leaves are the points of P and the internal nodes of the tree indicate the distance at which its subtrees merge.

[*] This work was partially supported by the NSF grants CCF 1318595, CCF 1423230, DMS 1547357, and NSF award CAREER 1453472.

[†] The full version of this paper [11] is available at http://arxiv.org/abs/1707.09904.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 28; pp. 28:1–28:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

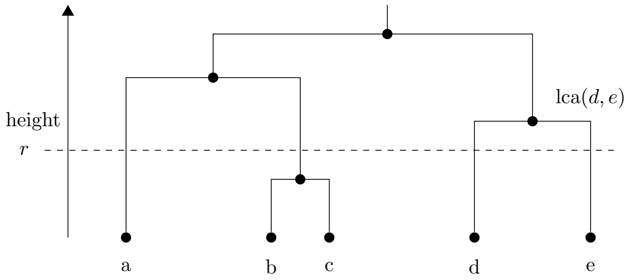


Figure 1 The dendrogram of an ultrametric, (U, μ), on points $\{a, b, c, d, e\}$. The points of μ are the leaves of the dendrogram (height 0). The distance between two points $x, y \in U$ is given by the height of their lowest common ancestor, $\mathrm{lca}(x, y)$, that is at $\mu(x, y)$. The dashed cut at r induces a natural clustering $\{\{a\}, \{b, c\}, \{d, e\}\}$ of the points of U by grouping points which belong to the same subtree. Each of these groups are contained in disjoint balls of radius r.

We aim to address the analogous question of how to avoid having to decide the number of clusters in advance in the case of dynamic data. Here, we adopt the temporal clustering framework of [9, 10]. In this framework, the input is a sequence of clustering problems, and the goal is to ensure that the solutions of successive instances remain close according to some objective. This differs from incremental [2, 7] and kinetic clustering [1, 4, 15, 17] in that there is no constraint that the clustering instances in the input must be incrementally related. Further, an optimal sequence of spatial clusterings is not automatically a low cost solution to the temporal clustering instance.

In this paper we present a natural adaptation of hierarchical clustering to the temporal setting. We study the problem of finding a temporally coherent sequence of hierarchical clusterings from a sequence of unlabeled point sets. Our goal is to produce a sequence of hierarchical clusterings (dendrograms) corresponding to each set of points in the input such that successive pairs of clusterings have similar dendrograms. We show that the corresponding optimization problem is NP-hard. However, a polynomial-time approximation algorithm exists when the metric spaces in the input are taken from a common ambient metric space. We explore the properties of this algorithm and find that it is unstable under perturbations of the metric. We then show how to restore stability with only a slight loss in the guarantee.

Problem formulation

An idea used in this paper is that we may hierarchically cluster a metric space by trying to find a low distortion embedding of it into an ultrametric. An *ultrametric* is a metric space which satisfies a stronger version of the triangle inequality. Formally, an ultrametric space is a metric space $U = (X, \mu)$ such that $\mu(x, z) \leq \max\{\mu(x, y), \mu(y, z)\}$, for all $x, y, z \in X$.

Ultrametric spaces have interesting geometry. For instance, in an ultrametric all points contained in a ball of radius r are *centers* of the ball. That is, for any $q \in \mathrm{B}_U(p; r)$, we have $\mathrm{B}_U(q; r) = \mathrm{B}_U(p; r)$, where $\mathrm{B}_M(p; r)$ denotes the ball of radius r about a point p in a metric space M. Further, given any pair of balls $B \subseteq U$, $B' \subseteq U$ with non-empty intersection, one has $B \subseteq B'$ or $B' \subseteq B$. This simple fact implies that any ultrametric space has the structure of a tree where items in a common subtree are close. That is, an ultrametric induces a natural hierarchical clustering, commonly depicted as a dendrogram (see Figure 1).

Similarity of dendrograms. For dendrograms over sets of points with identical labelings there is a natural dissimilarity measure given by comparing the merge heights for any pair of corresponding points. Namely, $\max_{u,u' \in P} |h_1(u, u') - h_2(u, u')|$, where h_1, and h_2 give the merge heights for a respective pair of dendrograms.

One immediate obstacle to adopting this formalization is that our model does not require that the sets of points comprising the input have the same cardinality. For this reason, we take the point of view that two dendrograms are similar if there exists a *correspondence* between their leaves such that the merge heights of corresponding points are close. Formally, a correspondence between U and V is a relation $\mathcal{C} \subseteq U \times V$ such that $\pi_U(\mathcal{C}) = U$, $\pi_V(\mathcal{C}) = V$. Here, π_U, π_V denote the canonical projections of $U \times V$ to U and V (respectively). Further we use the notation $\mathrm{Corr}(U, V)$ to denote the set of correspondences between U, V. Given a correspondence $\mathcal{C}$ between two sets of points P_1, P_2, we have the following dissimilarity measure which accounts for differences in the merge heights of a pair of dendrograms under a correspondence $\mathcal{C}$. This measure is called the *distortion* [5], or the *merge distortion distance* with respect to $\mathcal{C}$ [12], and is given by $\mathrm{dis}(h_1, h_2; \mathcal{C}) := \max_{(u,v),(u',v') \in \mathcal{C}} |h_1(u, u') - h_2(v, v')|$.

Generalized version. Our goal, then, is not only to output a sequence of hierarchical clusterings corresponding to the point sets of the input, but also to produce an interstitial sequence of low distortion correspondences linking successive pairs of dendrograms. We quantify the extent to which an ultrametric faithfully represents an input metric space under the ℓ^∞ norm. Specifically, let $U = (P, d_U)$, $V = (P, d_V)$ be a pair of finite pseudometric spaces on the same set of points. We define $L^\infty(U, V) = \max_{p,p' \in P} |d_U(p, p') - d_V(p, p')|$. In other words, a pseudometric space V is a good fit for U (and vice-versa) whenever $L^\infty(U, V)$ is small.

Let $M := (X, d)$ be a pseudometric space. If for any $u, v, w \in X$, we have $d(u, v) \leq \max\{d(u, w), d(w, v)\}$ then we say that d is a *pseudo-ultrametric* and M is a *pseudo-ultrametric space*. We now formally define this general version of the problem.

▶ **Definition 1** (Temporal Hierarchical Clustering (Generalized Version)). Let $\mathcal{M} := \{M_i\}_{i=1}^t$ be a sequence of metric spaces, where for each $i \in [t]$, $M_i = (P_i, \cdot)$, and let $\chi, \rho \in \mathbb{R}_{\geq 0}$. The goal of the GENERALIZED TEMPORAL HIERARCHICAL CLUSTERING problem is to find a sequence of pseudo-ultrametric spaces, $\{U_i := (P_i, \mu_i)\}_{i=1}^t$ and a sequence of correspondences $\{\mathcal{C}_i\}_{i=1}^{t-1}$, where for each $i \in [t]$, we have $L^\infty(M_i, U_i) \leq \chi$, and for any $i \in [t-1]$, $\mathcal{C}_i \in \mathrm{Corr}(P_i, P_{i+1})$ with $\mathrm{dis}(\mu_i, \mu_{i+1}; \mathcal{C}_i) \leq \rho$. Such a clustering is called a GENERALIZED (χ, ρ)-CLUSTERING of $\mathcal{M}$.

We show in Section 4 that the GENERALIZED HIERARCHICAL TEMPORAL CLUSTERING problem is NP-hard.

Local version. Absent the ambient metric space, the above notion of distortion would be sufficient to capture the intuitive idea that consecutive hierarchical clusterings should be close. However, it is easy to produce examples where symmetries in the input permit low-distortion correspondences which are manifestly non-local in the ambient space. Thus it makes sense to further require that any correspondence be local in the ambient metric. We say that a correspondence $\mathcal{C}$ is *δ-local* provided that $\max_{(u,v) \in \mathcal{C}} d(u, v) \leq \delta$, where d is the distance in the ambient space.

We now formalize this version of the problem. Here, the input $P := \{P_i\}_{i=1}^t$, consists of a sequence of unlabeled, finite, non-empty subsets of a metric space M. We call such a sequence a *temporal-sampling* of M of length t, and refer to individual elements of the sequence (P_i for some $i \in [t]$) as a *level* of P (see [9, 10]). The *size* of P is simply the sum of the number of points in each level of P, that is $\sum_{i=1}^t |P_i|$. Let $M = (X, d)$ be a metric space. For any $P \subseteq X$ we use the notation $M[P]$ to denote the restriction of M to P, that is, $M[P] = (P, d|_P)$. We have the following definition:

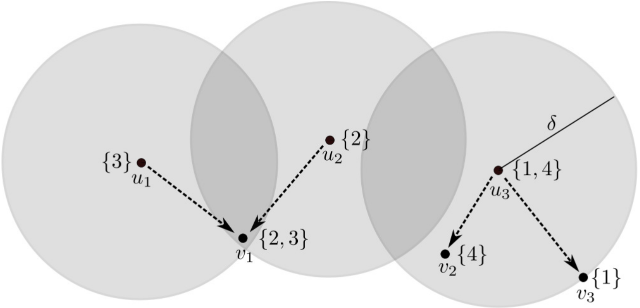


Figure 2 A δ-contiguous 4-labeling of $P_1, P_2 \subset \mathbb{R}^2$, $P_1 = \{u_1, u_2, u_3\}$, $P_2 = \{v_1, v_2, v_3\}$. Balls of radius δ are drawn about the points of P_1. Note that the labels used by points of P_2 "come from" points of P_1 which are δ-close, demonstrating condition 2 of Definition 3. The symmetric condition also holds. Further note that there is no requirement that $|P_1| = |P_2|$.

▶ **Definition 2** (Temporal Hierarchical Clustering (Local Version)). Let $P := \{P_i\}_{i=1}^t$ be a temporal-sampling over a metric space $M = (X, d)$, and let $\chi, \delta \in \mathbb{R}_{\geq 0}$. The goal of the LOCAL TEMPORAL HIERARCHICAL CLUSTERING problem is to find a sequence of pseudo-ultrametric spaces, $\{U_i\}_{i=1}^t$, where for each $i \in [t]$, $U_i = (P_i, \cdot)$, and $L^\infty(M[P_i], U_i) \leq \chi$, together with a sequence of correspondences $\{\mathcal{C}_i\}_{i=1}^{t-1}$ where for any $i \in [t-1]$, $\mathcal{C}_i \in \mathrm{Corr}(P_i, P_{i+1})$ with $\max_{(u,v)\in\mathcal{C}_i} d(u,v) \leq \delta$. Such a clustering is called a LOCAL (χ, δ)-CLUSTERING.

While the general version of the problem is NP-hard, the local version is trivial and can be computed in $O(n^2)$-time by computing a correspondence minimizing the Hausdorff distance for each pair of successive levels. We highlight this problem for expository purposes as well as a prelude to a labeled version of the problem.

This version of the problem is further of interest in that it can be used to approximate the general version such that the resulting distortion is bounded in terms of χ, and δ. We discuss this topic further in Section 4.

Labeled version. There are already several drawbacks with previous versions of the problem in regard to making concrete cluster assignments. In particular it is unclear how to coherently assign cluster labels to points given a correspondence. Moreover, we must account for the fact that the number of points can vary across levels. Taking the point of view that a good labeling is one in which labels in successive levels remain close, we opt to allow points to be given multiple labels. Doing so affords us additional bookkeeping to help ensure that labelings for near by levels remain local, even across levels which require relatively few labels.

To this end, given a set P, a k-labeling of P is a function $L : P \to 2^{[k]}$ such that $\{L(p) : p \in P\}$ is a partition of $[k]$. Informally, we say two labelings are δ-contiguous if the copies of the same label in a pair of assignments are no farther than δ. We have the following definition:

▶ **Definition 3.** Given a pair of sets P_1, P_2 of points from a metric space M, and a pair k-labelings L_1, L_2 of P_1, P_2 (respectively), we say that L_1 and L_2 are δ-contiguous in M if

1. for all $u \in P_1$, $L_1(u) \subseteq \bigcup_{v \in \mathrm{B}_M(u,\delta)\cap P_2} L_2(v)$,
2. for all $v \in P_2$, $L_2(v) \subseteq \bigcup_{u \in \mathrm{B}_M(v,\delta)\cap P_1} L_1(u)$.

See Figure 2 for an example.

Since points can be multi-labeled, we need a tie-breaking rule to determine which label applies. By convention we take the label of any set of points to be the smallest label among all labels of points in the set. Moreover, a good solution should never use more than n labels on an input of size n. We are now ready to define the main version of the problem.

▶ **Definition 4** (Temporal Hierarchical Clustering). Let $P := \{P_i\}_{i=1}^t$ be a temporal-sampling of size n over a metric space M with distance, d, and let $\chi, \delta \in \mathbb{R}_{\geq 0}$. The goal of the Temporal Hierarchical Clustering problem is to find a sequence of pseudo-ultrametric spaces, $\{U_i\}_{i=1}^t$, such that for any $i \in [t]$, $L^\infty(M[P_i], U_i) \leq \chi$, and a sequence of k-labelings, $\{L_i\}_{i=1}^t$, for $k \leq n$, such that for any $i \in [t-1]$, L_i, L_{i+1} are δ-contiguous. Such a clustering is called a Labeled (χ, δ)-Clustering.

Overview. In Section 2 we show how to find an optimal solution to the local version of the problem in $O(n^2)$-time. Then, in Section 3, we give an $O(n^3)$-time algorithm which converts any Local (χ, δ)-Clustering into a Labeled (χ, δ)-Clustering. This combined with Section 2 implies an optimal solution for the labeled version of the problem. In Section 4 we show that the general version is NP-hard, but observe that the local version provides an approximate solution in the special case where the inputs comes from a common metric space. In Section 5 we show that the optimal algorithms are unstable with respect to perturbations of the metric, and how to ensure stability by changing the ultrametric construction. Last, Section 6 contains an experiment.

2 Local Version

In this section we present a straightforward solution to the local version of temporal hierarchical clustering in $O(n^2)$-time. We are not directly interested in the solution of this problem. Instead, this section serves as a prelude to solving the labeled version.

Algorithm. The algorithm is trivial. Let $\mathcal{A}$ be a scheme for finding the ℓ^∞-nearest ultrametric to a metric. For each set of points in the input we use $\mathcal{A}$ to find an ultrametric. To compute correspondences between successive levels P_i, P_{i+1}, we add all pairs of points $(u, v) \in P_i \times P_{i+1}$ such that u and v are at a distance of at most the Hausdorff distance of P_i, P_{i+1}. Formally, the algorithm takes a temporal-sampling $P = \{P_i\}_{i=1}^t$ of a metric space M as input and consists of the following steps:

Step 1: Fitting by ultrametrics. For each $i \in [t]$, find an ultrametric $U_i = \mathcal{A}(M[P_i])$ near to $M[P_i]$ via a chosen scheme.

Step 2: Build correspondences. For each $i \in [t-1]$, compute $\mathcal{C}_i = \{(u, v) \in P_i \times P_{i+1} : d(u, v) \leq d_H^M(P_i, P_{i+1})\}$. Here, d_H^M denotes the Hausdorff distance in the ambient metric space.

Step 3: Return $\left(\{U_i\}_{i=1}^t, \{\mathcal{C}_i\}_{i=1}^{t-1}\right)$.

Analysis. Let n denote the size of the temporal sampling. In this section we argue that the above algorithm returns an optimal solution in $O(n^2)$ time, provided that it is equipped with a scheme for finding the ℓ^∞-nearest ultrametric to an n-point metric space in $O(n^2)$-time. The following theorem ensures that one exists.

▶ **Theorem 5** (Farach-Colton Kannan Warnow [13]). *Let M be an n-point metric space and let $\mathcal{U}(M)$ denote the set of ultrametrics on the points of M. There exists an $O(n^2)$-time algorithm which finds $\arg\min_{U \in \mathcal{U}(M)} L^\infty(U, M)$.*

We are now ready to prove the main theorem of this section.

▶ **Theorem 6.** *Let P be a temporal-sampling of size n which admits a* Local (χ, δ)-Clustering*. There exists an $O(n^2)$-time algorithm returning a* Local (χ, δ)-Clustering.

Proof. Let t denote the length of P, and M the ambient metric space. Run the algorithm of Section 2 where $\mathcal{A}$ is the algorithm of Farach-Colton, Kannan, and Warnow [13]. Let $\{U_i\}_{i=1}^t$ denote the pseudo-ultrametrics in the output. By Theorem 5, $\chi' = \max_{i\in[t]} L^\infty(U_i, M[P_i]) \leq \chi$, as otherwise $\chi > \chi'$ would imply that for some level $i \in [t]$, the algorithm of Theorem 5 fails to return an ℓ^∞-nearest ultrametric to P_i.

Let $\delta' = \max_{i\in[t-1]} d_H^M(P_i, P_{i+1})$. We now argue that δ' is smallest possible in the sense that P admits a Local (χ', δ')-Clustering, but does not admit an Local (χ, δ)-Clustering for any χ, when $\delta < \delta'$. Let $\Gamma := \{\delta : P \text{ admits a } \textsc{Local}(\cdot, \delta) - \textsc{Clustering}\}$. First we show $\delta' \leq \inf \Gamma$. Fix any Local $(\cdot, \delta)$-Clustering, and let $\{\mathcal{C}_i\}_{i=1}^{t-1}$ be the associated sequence of δ-local correspondences. Fix some $1 \leq i < t$ and some $p \in P_i$. Since $\mathcal{C}_i$ is a correspondence, $\pi_{P_i}(\mathcal{C}_i) = P_i$, and thus there exists $q \in P_{i+1}$ such that $(p, q) \in \mathcal{C}_i$. Since $\mathcal{C}_i$ is δ-local it holds that $d(p, q) \leq \delta$, and we conclude $d(p, P_{i+1}) \leq \delta$. An analogous argument for $q \in P_{i+1}$ implies $d(P_i, q) \leq \delta$. Thus, for $1 \leq i < t$, $\delta' \leq d_H^M(P_i, P_{i+1}) = \max\left(\max_{p\in P_i} d(p, P_{i+1}), \max_{q\in P_{i+1}} d(P_i, q)\right) \leq \delta$. Now we argue that δ' is feasible. Fix $1 \leq i < t$. Since $d_H(P_i, P_{i+1}) \leq \delta'$ it holds that for every point $p \in P_i$ there exists $q_p \in P_{i+1}$ such that $d(p, q_p) \leq \delta'$. Construct a set $\mathcal{C}_i^+ = \{(p, q_p) : p \in P_i, q_p \in P_{i+1}, \text{ and } d(p, q_p) \leq \delta'\}$. Analogously construct a set $\mathcal{C}_i^- = \{(p_q, q) : q \in P_{i+1}, p_q \in P_i, \text{ and } d(p_q, q) \leq \delta'\}$. The set $\mathcal{C}_i := \mathcal{C}_i^+ \cup \mathcal{C}_i^-$ is thus a δ'-local correspondence between P_i, P_{i+1}. Thus, it follows that $\delta' \in \Gamma$.

The preceding two paragraphs show that the result is a Local (χ, δ)-Clustering. It only remains to show the algorithm runs in $O(n^2)$-time. Let $n_i = |P_i|$ for $i \in [t]$. Step 1 takes $O(n^2)$-time as finding the ℓ^∞-nearest ultrametric for level i can be done in $O(n_i^2)$-time by Theorem 5. Computing the inter-level Hausdorff distance and building the correspondence for level i in Step 2 can both be done in $O(n_i^2)$-time, for a total of $O(n^2)$-time over all. ◀

3 Labeled Version

In this section we show how to convert a Local (χ, δ)-Clustering into a Labeled (χ, δ)-Clustering in $O(n^3)$-time by transforming a sequence of δ-local correspondences into a sequence of pairwise δ-contiguous labelings.

Network flow. Drawing upon an idea in [9, 10], we employ minimum cost feasible flow to find a δ-contiguous labeling with few labels. Formally, we construct the flow instance as follows: Let $P = \{P_i\}_{i=1}^t$ be a temporal-sampling. Given the δ-local correspondences of a Local $(\cdot, \delta)$-Clustering, $\{\mathcal{C}_i\}_{i=1}^{t-1}$, the following construction transforms P into a flow network, $F := F(\{\mathcal{C}_i\}_{i=1}^{t-1})$, such that corresponding points in successive levels are connected by a directed edge which points to the higher indexed level. Moreover, a source, s, connects to each of the points in the first level, while the sink s' is the target of a directed edge from each point in P_t. Formally, let $V_i(P) = \{(i, v) : v \in P_i\}$. For $i \in [t-1]$, let $E_i(P) \subseteq V_i(P) \times V_{i+1}(P)$ such that $((i, u), (i+1, v)) \in E_i(P)$ if and only if $(u, v) \in \mathcal{C}_i$. The vertices of F consist of s, s', and the contents of $V_1(P), \ldots, V_t(P)$. The edges of F consist of the union of $\{s\} \times V_1(P)$, $V_t(P) \times \{s'\}$, and $\bigcup_{i=1}^{t-1} E_i(P)$. Specifically, we seek an integral flow with minimum flow value such that the in-flow of each vertex of $\bigcup_{i=1}^{t} V_i(P)$ is at least one.

Algorithm. The main idea is to view each correspondence as a bipartite graph. We concatenate the sequence of correspondences together by merging overlapping vertices. This allows us to interpret the sequence of correspondences as a graph. Our goal is then to decompose this graph into a path cover of small size, which we do by solving a flow instance. Since this graph only contains edges between points which are close, the resulting labeling will be contiguous. Formally, we perform the following steps:

Step 1: Constructing a flow instance. Given a sequence of δ-local correspondences of a Local $(\cdot, \delta)$-Clustering, construct the minimum flow instance $F := F(\{\mathcal{C}_i\}_{i=1}^{t-1})$ as defined above.

Step 2: Solve the flow instance. Find a minimum cost integral flow f in F.

Step 3: Decompose the flow. Greedily extract unit flows from f to construct a list of paths $\{\tau_i\}_{i=1}^{k}$.

Step 4: Construct label functions. Build label functions $L_1, \ldots, L_t$ by initializing each to the empty set. Next, for each $\tau_j \in \{\tau_i\}_{i=1}^{k}$, denote τ_j as the t point sequence $p_1, \ldots, p_t$. Append label j to $L_1(p_1), \ldots, L_t(p_t)$.

Step 5: Output. Return the labelings $L_1, \ldots, L_t$.

Analysis. In this section we show that the above algorithm finds an optimal solution in $O(n^3)$-time on temporal samplings of size n. To this end we now argue that the above network flow instance is feasible.

▶ **Lemma 7.** *Let $P = \{P_i\}_{i=1}^{t}$ be a temporal-sampling. Given the δ-local correspondences of a Local $(\cdot, \delta)$-Clustering, $\{\mathcal{C}_i\}_{i=1}^{t-1}$, the flow instance $F := F(\{\mathcal{C}_i\}_{i=1}^{t-1})$ is feasible with value at most n.*

Proof. For any $1 \leq i \leq t$, any point $p \in P_i$ can be extended to a path from P_1 to P_t, by iteratively extending the ends of the path via the correspondences. Construct a feasible flow f by initializing f to be zero everywhere. Greedily extend points receiving no flow to paths from P_1 to P_t in the described manner, and increase the flow value of f along the path by 1. It follows that f remains integral and satisfies all lower bounds of F. Since we flow at most 1 unit of flow per point of P, the value of f is at most n. ◀

The next theorem shows that the algorithm outputs an optimal clustering.

▶ **Theorem 8.** *Let P be a temporal-sampling of size n. There exists an $O(n^3)$-time algorithm which is guaranteed to output a Labeled (χ, δ)-Clustering of P, for any χ, δ such that P admits a Labeled (χ, δ)-Clustering.*

Proof. Let t be the length of P. Run the algorithm of Section 2 on P. Since P admits a Labeled (χ, δ)-Clustering, it also admits a Local (χ, δ)-Clustering where for any $1 \leq i < t$, the i-th correspondence is given by $\mathcal{C}_i = \{(u, v) : (u, v) \in P_i \times P_{i+1}, L_i(u) \cap L_{i+1}(v) \neq \varnothing\}$. Thus, by Theorem 6, we are guaranteed a Local (χ, δ)-Clustering in $O(n^2)$-time. Let $\{\mathcal{C}_i\}_{i=1}^{t-1}$ be its δ-local correspondences, and run the above algorithm on it. By Lemma 7, the flow instance $F := F(\{\mathcal{C}_i\}_{i=1}^{t-1})$ is feasible with value at most n. Using an algorithm of Gabow & Tarjan [16], we can solve F in $O(n^3)$-time, yielding an integral flow f. Again in $O(n^3)$-time, we decompose f into a collection of unit flows $\{\tau_j\}_{j=1}^{k}$, for some $k \leq n$, which we interpret as paths from P_1 to P_t.

We now verify that the sequence of label functions output by the algorithm is indeed a δ-contiguous k-labeling for some $k \leq n$. For any $i \in [t]$, and any $j \in [k]$ let $\tau_j(i)$ denote the i-th vertex in the j-th path. Recall that for each $i \in [t]$, we assign each point $u \in P_i$ the set of labels $L_i(u) = \{j : j \in [k], u = \tau_j(i)\}$. Note that each label in $[k]$ is used at most once per level since for any j, $i \in [t]$, $\tau_j(i)$ is the only place where τ_j intersects P_i. Also, since each τ_j intersects all levels $i \in [t]$, each label is used at least once per level. It follows that $\{L_i(u) : u \in P_i\}$ is a partition of $[k]$. Finally, since the edges of F correspond to points that are separated by at most δ in the ambient space, any two uses of the label $j \in [k]$ for some $i \in [t-1]$ occur within $d(\tau_j(i), \tau_j(i+1)) \leq \delta$. Thus the corresponding sequence of k-labelings is indeed pairwise δ-contiguous. ◀

4 Generalized Version

In this section we show that the generalized version problem is NP-hard. However, we argue that for the special case where the points of the input share a (known) common ambient metric, the algorithm of Section 2 gives an approximate solution. It remains an open question as to how to find an approximate solution in polynomial-time when there is no ambient metric (or it is unknown).

NP-hardness. Let $G = (V, E)$ be an instance of 3-coloring. We construct an instance of Generalized Temporal Hierarchical Clustering, $\mathcal{M}(G)$, consisting of two levels. For the first level let $P = \{r, g, b\}$ be a set of three points, and let d_P be a metric on P such that distinct $p, p' \in P$ have $d_P(p, p') = 2$. Denote the corresponding metric space $M_P := (P, d_P)$. For the second level we construct a metric space $M_V := (V, d_V)$, where $d_V : V \times V \to \mathbb{R}_{\geq 0}$, such that

$$d_V(u, v) = \begin{cases} 2 & \text{if } \{u, v\} \in E \\ 1 & \text{if } \{u, v\} \notin E \text{ and } u \neq v, \\ 0 & \text{otherwise.} \end{cases}$$

▶ **Lemma 9.** *If G admits a 3-coloring requiring 3 colors, then $\mathcal{M}(G)$ admits a* Generalized *$(1, 0)$-*Clustering.

Proof. Fix a 3-coloring of $G = (V, E)$. We will exhibit a pair of pseudometric spaces and a 0-distortion correspondence between them. For the first space let $U_P = (P, \mu_P)$ be a uniform metric space where distinct points are at a distance of 1. Note that $L^\infty(M_P, U_P) = \max_{u,u' \in P} |d_P(u, u') - \mu_P(u, u')| = 1$, since for any distinct $u, u' \in P$, $|d_P(u, u') - \mu_P(u, u')| = |2 - 1| = 1$.

We will use the points of P to denote the color class of $v \in V$. Fix $c : V \to P$ be such that $c(v) = c(v')$ if and only if v, v' share the same color class. Let $U_V = (V, \mu_V)$ be the pseudometric space where for any $v, v' \in V$, $\mu_V(v, v') = 1$ if and only if $c(v) \neq c(v')$, and $\mu_V(v, v') = 0$ otherwise. We now bound $L^\infty(M_V, U_V)$ by considering $|d_V(v, v') - \mu_V(v, v')|$ for an arbitrary pair $v, v' \in V$. Since $\mu_V(v, v) = d_V(v, v) = 0$ for any $v \in V$, only distinct v, v' can contribute to the distortion. Suppose $\{v, v'\} \in E$, then $c(v) \neq c(v')$ and thus $|d_V(v, v') - \mu_V(v, v')| = |2 - 1| = 1$. Otherwise, $\{v, v'\} \notin E$, and $d_V(v, v') = 1$ while $\mu_V(v, v') \leq 1$ so that $|d_V(v, v') - \mu_V(v, v')| \leq 1$. Thus $L^\infty(M_V, U_V) \leq 1$.

Last, let $\mathcal{C} = \{(p, v) \in P \times V : c(v) = p\}$. We now verify that $\mathcal{C}$ is a 0-distortion correspondence. To see that $\mathcal{C} \in \mathrm{Corr}(P, V)$, note that $\pi_P(\mathcal{C}) = P$ since G requires 3 colors, and $\pi_V(\mathcal{C}) = V$ since every vertex $v \in V$ belongs to a color class. Finally, to bound $\mathrm{dis}(\mu_P, \mu_V; \mathcal{C})$ note that for any $(p, v), (p', v') \in \mathcal{C}$, either $p = p'$ and $|\mu_P(p, p') - \mu_V(v, v')| = |\mu_V(v, v')| = 0$ (since $c(v) = c(v')$), or $p \neq p'$ and $|\mu_P(p, p') - \mu_V(v, v')| = |1 - 1| = 0$. ◀

▶ **Lemma 10.** *If G does not admit a 3-coloring, then $\mathcal{M}(G)$ does not admit a* Generalized *$(2, 0)$-*Clustering.

Proof. Let $(V, E) = G$. Fix a Generalized $(\chi, 0)$-Clustering of $\mathcal{M}(G)$ for some $\chi < 2$ consisting of ultrametrics $U_P = (P, \mu_P)$, $U_V = (V, \mu_V)$, and a 0-distortion correspondence

$\mathcal{C} \in \mathrm{Corr}(P, V)$. We first argue that the points of P are separated. Let $p, p' \in P$, $p \neq p'$. If $\mu_P(p, p') = 0$ then $L^\infty(M_P, U_P) \geq |\mu_P(p, p') - d_P(p, p')| = |0 - 2| = 2$. Thus $\chi \geq 2$, a contradiction.

Now fix a map $c : V \to P$, such that for any $v \in V$, $c(v) = p$ such that $(p, v) \in \mathcal{C}$. First we argue that c is indeed a function by showing that for any $v \in V$, v corresponds to exactly one point in P. To see why observe that given any $(p, v), (p', v) \in \mathcal{C}$ with $p \neq p'$ it follows that $0 = \mathrm{dis}(\mu_P, \mu_V; \mathcal{C}) \geq |\mu_P(p, p') - \mu_V(v, v)| = \mu_P(p, p') > 0$. We now show how to use c to construct a 3-coloring of G. Since $\chi < 2$, for every $\{u, v\} \in E$, we have $\mu_V(u, v) > 0$, as otherwise $\chi \geq L^\infty(M_V, U_V) \geq |d_V(u, v) - \mu_V(u, v)| = 2$. Consider any pair of corresponding points $(c(u), u), (c(v), v) \in \mathcal{C}$. It must be the case that $c(u) \neq c(v)$ as otherwise $\mathrm{dis}(\mu_P, \mu_V; \mathcal{C}) \geq |\mu_P(c(u), c(v)) - \mu_V(u, v)| = \mu_V(u, v) > 0$. Color the graph by assigning each $v \in V$ to a color class given by $c(v)$. Since for adjacent $u, v \in V$, we have $\mu_V(u, v) > 0$, it follows that $c(u) \neq c(v)$, and thus there is no edge between vertices of the same color. We have exhibited a 3-coloring of G. ◀

Theorem 11 result follows directly from Lemma 9, and Lemma 10. The proof also implies that for the Generalized Temporal Hierarchical Clustering problem, for some fixed ρ, approximating χ within any factor smaller than 2 is NP-hard.

▶ **Theorem 11.** *The* Generalized Temporal Hierarchical Clustering *problem is NP-hard.*

Approximation by local version. We now show that any Local (χ, δ)-Clustering is a Generalized $(\chi, 2\chi + 2\delta)$-Clustering. That is, we can view the local version of the problem as an approximation to the general version in the special case that the points of the input come from the same metric space.

▶ **Lemma 12.** *Let P be a temporal-sampling. Any* Local (χ, δ)-Clustering *of P is a* Generalized $(\chi, 2\chi + 2\delta)$-Clustering *of P.*

Proof. Suppose P has length t and ambient metric space $M = (X, d)$. Fix a Local (χ, δ)-Clustering of P with ultrametrics $\{U_i = (P_i, \mu_i)\}_{i=1}^t$, and correspondences, $\{\mathcal{C}_i\}_{i=1}^{t-1}$, induced by labelings of successive pairs of levels. Observe that $\max_{i \in [t-1]} \mathrm{dis}(\mu_i, \mu_{i+1}, \mathcal{C}_i) = \max_{i \in [t-1]} \max_{(x,y),(x',y') \in \mathcal{C}_i} |\mu_i(x, x') - \mu_{i+1}(y, y')|$. Since $\chi \geq \max_{i \in [t]} L^\infty(M[P_i], U_i)$, it follows by definition of L^∞ that $\chi \geq |\mu_i(x, x') - d(x, x')|$ for any $i \in [t]$, $x, x' \in P_i$. Fix an arbitrary $i \in [t-1]$ and let $(x, y), (x', y') \in \mathcal{C}_i$. By triangle inequality $|\mu_i(x, x') - \mu_{i+1}(y, y')| \leq |d(x, x') - d(y, y')| + 2\chi$. Note that since $(x, y), (x', y') \in \mathcal{C}_i$, we have $d(x, y), d(x', y') \leq \delta$. Thus y, $y' \in X$ are contained in δ-balls of x, x' in X (respectively). It follows that $|d(x, x') - d(y, y')| \leq 2\delta$. We conclude that for any $i \in [t-1]$, $(x, y), (x', y') \in \mathcal{C}_i$, $|\mu_i(x, x') - \mu_{i+1}(y, y')| \leq 2\chi + 2\delta$, and thus $\max_{i \in [t-1]} \mathrm{dis}(\mu_i, \mu_{i+1}; \mathcal{C}_i) \leq 2\chi + 2\delta$. ◀

5 Stability

In this section we show that the algorithm for finding an ℓ^∞-nearest ultrametric in [13] is unstable under perturbations of the metric and, consequently, so are our algorithms. Stability, naturally, is a desirable property; as otherwise if small changes in the input are allowed to produce vastly different ultrametrics, then the observed temporal coherence of the output is lost. Furthermore, this is the case even if the cost of fitting each level to an ultrametric remains best possible. We resolve this issue in practice by instead finding the ℓ^∞-nearest subdominant ultrametric.

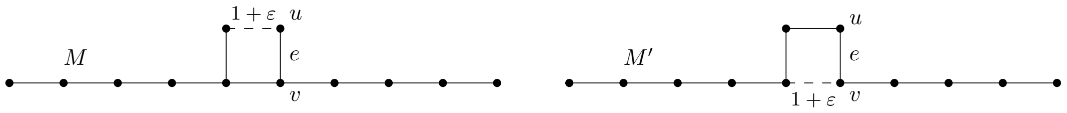


Figure 3 Two metric graphs M, M' which differ by an ε-perturbation. Solid edges have length 1 and appear in their respective MSTs. Dashed edges have length $1+\varepsilon$. In M, $p(e) = 6$, and the priority of any edge along the bottom of M is $\operatorname{diam}(M) = 9$. Let U, U' denote the result from running the algorithm in [13] on M, and M', respectively. In computing U from M the edge e is cut after all of the edges along the base, and thus u, v are assigned distance of $6 - \frac{1}{2}L^{\infty}(M, \mu_S(M)) = 6 - \frac{1}{2}(9-1) = 2$. Compare with M', where the priority p of any edge of the MST is $p = p(e) = \operatorname{diam}(M') = 9 + \varepsilon$. Thus u, v are assigned distance of $9 + \varepsilon - \frac{1}{2}L^{\infty}(M', \mu_S(M')) = 5 + \varepsilon/2$. By considering n point metric spaces with bases of length $n-2$, this example generalizes to show $L^{\infty}(U, U') = \Omega(\operatorname{diam}(M))$.

Subdominant ultrametrics. Let $M = (X, d)$ be a metric space. We will consider M to be a complete graph where the edges are weighted by distance, and use the notation T_M to refer to a minimum spanning tree on M. Further, for any $x, y \in M$, let $T_M(x, y)$ denote the unique path joining $x, y \in M$. Let $\mathcal{U}(M)$ denote the set of ultrametrics on the points of M. Let $\mathcal{U}_{\leq}(M) = \{(X, \mu) \in \mathcal{U}(M) : \mu(x, y) \leq d(x, y) \text{ for all } x, y \in M\}$. In other words, $\mathcal{U}_{\leq}(M)$ is the set of ultrametrics on the points of M such that no distance is made larger than its counterpart in M. We say that an ultrametric in $\mathcal{U}_{\leq}(M)$ is *subdominant* to M. Let $\mu_S(M) = (U, \mu)$ be a metric space on the points of M with distance function $\mu(x, y) = \max_{\{u,v\} \in T_M(x,y)} M(u, v)$. The distance function μ is independent of the choice of minimum spanning tree, and easily verified to be ultrametric and subdominant to M. It can further be shown that $\mu_S(M)$ is the unique, ℓ^{∞}-closest subdominant ultrametric to M. That is, $\mu_S(M) = \arg\min_{U \in \mathcal{U}_{\leq}(M)} L^{\infty}(U, M)$.

Instability. We now show that the algorithms of Section 2, Section 3 are unstable. To elucidate why we now restate the algorithm in [13] in a slightly modified form which helps to make our point. This procedure is equivalent to the following:

Step 1: Compute a minimum spanning tree. Given a metric space $M = (X, d)$ consider a weighted complete graph on X where the the weight of any edge $\{x, x'\}$ is $d(x, x')$. Find a minimum spanning tree of this graph, T_M.

Step 2: Compute cut-weights for each edge. Let $(X, \mu) = \mu_S(M)$. For each edge $e = \{u, v\} \in T_M$, compute and assign a priority $p(e)$ to e such that $p(e) = \max_{x,x' \in X}\{d(x, x') : e \in T_M(x, x'),\ \mu(x, x') = d(u, v)\}$,

Step 3: Assign distances. Edges are cut in order of descending priority. Any pair of vertices $u, v \in T_M$ first separated by a cut at e are assigned a distance of $p(e) - \frac{1}{2}L^{\infty}(M, \mu_S(M))$.

When an edge is cut, points first separated by the removal of that edge are assigned a distance which depends on its largest supported distance in M. The issue is that small perturbations in the metric can change the path structure of T_M so that an edge becomes responsible for linking a far pair of points. The only hope for stability is that the other term in the assigned distance, $\frac{1}{2}L^{\infty}(M, \mu_S(M))$, changes enough to offset this effect. However, Lemma 13 shows that this term is stable, and thus is not large enough to compensate. It follows that the above procedure is unstable. See Figure 3 for a concrete example.

Ensuring stability. In contrast, the ℓ^{∞}-nearest subdominant ultrametric is stable under metric perturbations. We now give a simple, direct proof of this fact for our setting. See [6] for extended discussion.

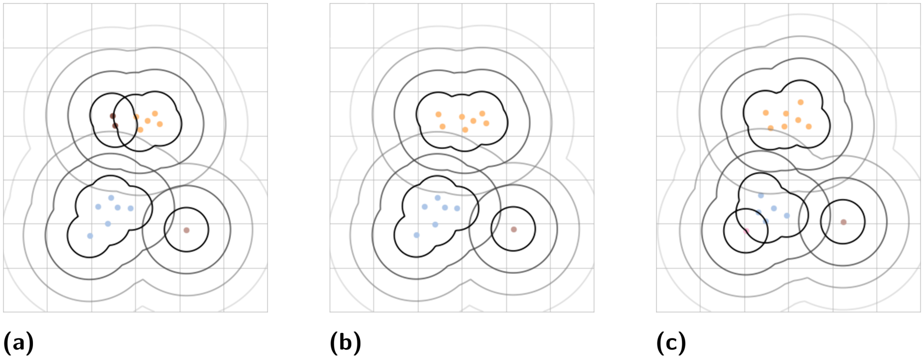


Figure 4 Three levels of a temporal hierarchical clustering. The contours show the coarse cluster structure which results from cutting the ultrametric at various offsets. Points which appear together within a contour share a cluster at that height in the ultrametric tree. **(4a)** Yellow and brown clusters are close. **(4b)** One level later, yellow and brown clusters merge. Note that the coarse structure remains stable. **(4c)** Ten levels later, a blue point (now pink) splits from its cluster.

▶ **Lemma 13.** *Let M, M' be metric spaces on the same points such that $L^\infty(M, M') \leq \varepsilon$, then $L^\infty(\mu_S(M), \mu_S(M')) \leq \varepsilon$.*

Proof. Let P denote the points of M. Fix a distance weighted MST of M, T_M, and let $(P, \mu) = \mu_S(M)$, $(P, \mu') = \mu_S(M')$. For any pair of points $x, y \in P$ let $\mathcal{P}(x, y)$ denote the set of all simple paths $x \rightsquigarrow y$ in M (when M is viewed as a complete graph). Let $w : \mathcal{P}(x, y) \to \mathbb{R}_{\geq 0}$ be the function that sends each path in $\mathcal{P}(x, y)$ to the value of its maximum weight edge. Observe that the maximum weight edge along $T_M(x, y)$ is equal to $\min_{\gamma \in \mathcal{P}(x,y)} w(\gamma)$, as otherwise it is possible to construct a spanning tree with cost strictly less than that of T_M. Thus, $\mu(x, y) = \min_{\gamma \in \mathcal{P}(x,y)} w(\gamma)$. Now since M, M' differ by an ε-perturbation, the values individual edges of the paths (and therefore the values of the paths in $\mathcal{P}(x, y)$ under w) change by at most ε. Thus, $|\mu(x, y) - \mu'(x, y)| \leq \varepsilon$ ◀

Such a choice for ultrametric embedding is suboptimal, but the next lemma shows that it is within a factor of 2 of optimal. This fact essentially follows from arguments in [13].

▶ **Lemma 14** ([13]). *Let M be a finite metric space and $U \in \mathcal{U}(M)$, then $L^\infty(\mu_S(M), M) \leq 2L^\infty(U, M)$.*

As one might expect, using the 2-approximate algorithm μ_S for $\mathcal{A}$ in the algorithm of Section 2 results in a Local $(2\chi, \delta)$-Clustering whenever the input admits a Local (χ, δ)-Clustering. Lemma 12 then implies that the result is a Generalized $(2\chi, 4\chi + 2\delta)$-Clustering. However, since the error incurred by μ_S is one-sided, there is no additional loss in the coupling distortion and the result is a Generalized $(2\chi, 2\chi + 2\delta)$-Clustering.

6 Example Output and Conclusion

In Figure 4, we present output based on synthetic data. For expository purposes we seek a data source for which many levels can reasonably be described as hierarchical, yet changes enough that the hierarchy evolves over time. We obtain such input by regularly saving snapshots of actor positions from a flocking simulation. A labeled clustering is obtained using the algorithm of Section 3 and fitting by subdominant ultrametrics.

We conclude by briefly mentioning some open questions. In Section 4 we show that the general problem is NP-hard, though our proof uses an unnatural metric space. It is unknown if the general version admits an exact algorithm on “nice” metric spaces. Further, it may still be possible to obtain optimal algorithms for the local and labeled versions of the problem which are stable under perturbations. Last, while we believe that our adaptations of hierarchical clustering are quite natural, one could consider alternative models where, say, the distortion is replaced with a tree dissimilarity measure (e.g. nearest neighbor interchange).

References

1 M. Ali Abam and M. de Berg. Kinetic spanners in $\mathbb{R}^d$. In *SOCG*, 2009. URL: `http://doi.acm.org/10.1145/1542362.1542371`.

2 M. Ackerman and S. Dasgupta. Incremental clustering: The case for extra clusters. In *NIPS*, 2014.

3 D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283494`.

4 J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *SODA*, 1997. URL: `http://dl.acm.org/citation.cfm?id=314161.314435`.

5 D. Burago, Y. Burago, and S. Ivanov. *A Course in Metric Geometry.* AMS, 2001.

6 G. E. Carlsson and F. Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *JMLR*, 11, 2010.

7 M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *STOC*, 1997.

8 T. K. Dey, A. Rossi, and A. Sidiropoulos. Spectral concentration, robust k-center, and simple clustering. *CoRR*, abs/1404.1008, 2014. URL: `http://arxiv.org/abs/1404.1008`.

9 T. K. Dey, A. Rossi, and A. Sidiropoulos. Temporal clustering. In *ESA*, volume 87 of *LIPIcs*, 2017. URL: `https://doi.org/10.4230/LIPIcs.ESA.2017.34`.

10 T. K. Dey, A. Rossi, and A. Sidiropoulos. Temporal clustering. *CoRR*, abs/1704.05964, 2017. URL: `http://arxiv.org/abs/1704.05964`.

11 T. K. Dey, A. Rossi, and A. Sidiropoulos. Temporal hierarchical clustering. *CoRR*, abs/1707.09904, 2017. URL: `http://arxiv.org/abs/1707.09904`.

12 J. Eldridge, M. Belkin, and Y. Wang. Beyond hartigan consistency: Merge distortion metric for hierarchical clustering. In *COLT*, volume 40, 2015.

13 M. Farach, S. Kannan, and T. J. Warnow. A robust model for finding optimal evolutionary trees. In *STOC*, 1993. URL: `http://doi.acm.org/10.1145/167088.167132`.

14 E. W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21, 1965.

15 S. A. Friedler and D. M. Mount. Approximation algorithm for the kinetic robust k-center problem. *Comput. Geom.*, 43(6-7), 2010.

16 H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5), 1989. URL: `https://doi.org/10.1137/0218069`.

17 J. Gao, L. J. Guibas, and A. Thanh Nguyen. Deformable spanners and applications. In *SOCG*, 2004. URL: `http://doi.acm.org/10.1145/997817.997848`.

18 D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2), 1985. URL: `https://doi.org/10.1287/moor.10.2.180`.

19 A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, 1988.

Agnostically Learning Boolean Functions with Finite Polynomial Representation[*]

Ning Ding

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai and State Key Laboratory of Cryptology, Beijing, China
dingning@sjtu.edu.cn

Abstract

Agnostic learning is an extremely hard task in computational learning theory. In this paper we revisit the results in [Kalai et al. SIAM J. Comput. 2008] on agnostically learning boolean functions with finite polynomial representation and those that can be approximated by the former. An example of the former is the class of all boolean low-degree polynomials. For the former, [Kalai et al. SIAM J. Comput. 2008] introduces the l_1-polynomial regression method to learn them to error $\mathrm{opt} + \epsilon$. We present a simple instantiation for one step in the method and accordingly give the analysis. Moreover, we show that even ignoring this step can bring a learning result of error $2\mathrm{opt} + \epsilon$ as well. Then we consider applying the result for learning concept classes that can be approximated by the former to learn richer specific classes. Our result is that the class of s-term DNF formulae can be agnostically learned to error $\mathrm{opt} + \epsilon$ with respect to arbitrary distributions for any ϵ in time $\mathrm{poly}(n^d, 1/\epsilon)$, where $d = O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon))$.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Agnostic Learning, Boolean Functions, Low-Degree Polynomials

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.29

1 Introduction

Learning various boolean function classes plays a central role in computational learning theory. In the PAC learning model [18], a boolean function class $\mathcal{C}$ is learnable if there is an efficient algorithm that, given parameters (ϵ, δ) and many labelled examples of form $(x, f(x))$ where x is chosen from some arbitrary distribution D and $f \in \mathcal{C}$ is an unknown, can with probability $1 - \delta$ output a hypothesis h satisfying $\Pr_{x \leftarrow D}[h(x) \neq f(x)] \leq \epsilon$.

In this model, there are rich boolean function classes that can be learned, such as conjunctions [18], s-term DNF formulas [14], intersections of halfspaces [13], polynomial threshold functions [13, 9] etc. If the underlying distribution D is restricted to some specific ones, some more classes can also be learned. For instance, if D is specified to be the uniform distribution, [15] shows that the Fourier spectrum of any function in AC^0 is concentrated on low-degree coefficients and then introduced the Low Degree Algorithm to learn the low-degree coefficients under the uniform distribution and thus generated a function approximately identical to the concept function. Following [15], some works present various Fourier concentration results for more expressive circuits augmented from AC^0 [10, 2, 7], monotone circuits [3] and boolean functions with small total influence or small noise sensitivity [13] and thus gain corresponding learning results with the Low Degree Algorithm.

[*] This work is supported by the National Natural Science Foundation of China (Grant No. 61572309) and National Cryptography Development Fund of China (Grant No. MMJJ20170128).

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 29; pp. 29:1–29:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Besides the PAC learning model, there is another much harder model, called the agnostic learning model [12, 8]. In this model, a boolean function class $\mathcal{C}$ is learnable if there is an efficient algorithm that, given many pairs of form (x, b) sampled from some arbitrary distribution D, can output a hypothesis f satisfying letting $\mathrm{er}_D(h)$ denote $\Pr_{(x,b)\leftarrow D}[h(x) \neq b]$, $\mathrm{er}_D(f) \leq \mathrm{opt} + \epsilon$, where $\mathrm{opt} = \min_{h\in\mathcal{C}}(\mathrm{er}_D(h))$. So far there have been a few successful attempts to agnostically learning functions. For instance, [11] shows that concept classes that can be approximated by low-degree polynomials can be agnostically learned. Some other works present relaxed requirements for this model: that the output hypothesis f is only required to satisfy $\mathrm{er}_D(f) \leq O(\mathrm{opt}) + \epsilon$ and even at the same time that the learning algorithm only needs to deal with uniform distributions or other specific ones. For instance, [11] shows that boolean function classes with Fourier concentration bounds and halfspaces can be agnostically learned under uniform distributions. [1, 5] show that halfspaces can be agnostically learned to error $O(\mathrm{opt}) + \epsilon$ under isotopic log-concave distributions.

1.1 Our Results

In this paper we revisit the results in [11] on agnostically learning boolean functions with finite polynomial representation and those that can be approximated by the former. By finite polynomial representation, we mean (in an non-rigorous way) that each one in the class admits a polynomial representation in which the number of monomials is much less than 2^n.

More precisely, let $\mathcal{S}$ denote a collection of some subsets of $[n]$. Let $\mathcal{H}_{n,\mathcal{S}}$ denote the class of boolean functions in which each $h(x) = \sum_{S\in\mathcal{S}} g_S \prod_{j\in S} x_j : \{0,1\}^n \to \{0,1\}$ where x_j denotes the jth bit of x and g_S's denote coefficients. Thus $\mathcal{H}_{n,\mathcal{S}}$ is thought of as one with finite polynomial representation if $|\mathcal{S}|$ is not large. For example, $\mathcal{H}_{n,\mathcal{S}}$ is the class of boolean low-degree polynomials if $\mathcal{S}$ consists of all S's with $|S| \leq d$ for some small d.

Recall that [11] presents a result for learning such classes, in which the l_1-polynomial regression method is introduced. Let $p(x)$ denote the polynomial generated by the method. After obtaining $p(x)$, the method outputs $\mathsf{Sign}(p(x) - t)$ for some t as the learned hypothesis. Note that the choice of t is not specified in [11]. So we use a simple sampling technique to determine t. That is, uniformly sample $t \in [0, 1]$ many times and select the one such that $\mathsf{Sign}(p(x) - t)$ is consistent with the most examples. We then show that the t selected this way can indeed satisfy that $\mathsf{Sign}(p(x) - t)$ achieves the error $\mathrm{opt} + \epsilon$. Moreover, we will also show that $t = \frac{1}{2}$ is a universal constant such that for any distribution D, $\mathsf{Sign}(p(x) - \frac{1}{2})$ achieves the error $2\mathrm{opt} + \epsilon$.

Then we consider the question of learning richer classes by applying the general result in [11] for all concept classes admitting low-degree polynomial l_1-approximation in expectation. The concept class in our consideration consists of all s-term DNF formulae. To do this, we show that each s-term DNF formula can be ϵ-uniformly approximated (i.e. l_∞ approximation) by a polynomial of degree $O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon))$. Thus the degree is less than n if $s = O(n^\kappa)$ for any $\kappa < \frac{1}{2}$. Then we have the following result.

▶ **Theorem 1.** *Let D be any distribution over $\{-1,1\}^n \times \{-1,1\}$. For the class of s-term DNF formulae, there is an algorithm that on input (ϵ, δ) and sufficiently many pairs sampled from D can with probability $1-\delta$ output a hypothesis f such that $er_D(f) \leq opt + \epsilon$ in time poly$(n^d, 1/\epsilon, \log(1/\delta))$ where opt denotes the optimal error among all such DNF formulae and $d = O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon))$.*

Our Techniques. We first outline the technique underlying the first part of this paper. The l_1-polynomial regression method in [11] converts the given examples to a l_1-norm minimization problem. Let f denote the one in $\mathcal{H}_{n,\mathcal{S}}$, achieving the optimal error. For each given pair

$(x,b) \leftarrow D$, b may not equal $f(x)$. So we introduce a variable e to denote $b - f(x)$. Then $e \in \{1, -1, 0\}$. Since $f = \sum_{S \in \mathcal{S}} g_S \prod_{j \in S} x_j$, substituting the value of x_j into f and letting $a_S = \prod_{j \in S} x_j$, we obtain $\sum_{S \in \mathcal{S}} g_S a_S + e = b$. Viewing all a_S's as coefficients, this equality is a linear equation of all g_S's. We also use $\mathbf{a}$ to denote the (row) vector $(a_{S_1}, \cdots, a_{S_N})$ (where we assume there is an order for all sets in $\mathcal{S}$ and let $N = |\mathcal{S}|$). Let $\mathbf{g}$ denote the (column) vector $(g_{S_1}, \cdots, g_{S_N})$. Thus the equation is $\mathbf{a} \cdot \mathbf{g} + e = b$.

Thus when given m random pairs, we can construct m equations of form $\mathbf{a} \cdot \mathbf{g} + e = b$. Let $\mathbf{A}$ denote the $m \times N$ matrix consists of all such $\mathbf{a}$ as rows, $\mathbf{e}$ denote the (column) vector consisting of all e's, $\mathbf{b}$ denote the (column) vector consisting of all b's. Thus m equalities can be represented as $\mathbf{A} \cdot \mathbf{g} + \mathbf{e} = \mathbf{b}$. Then the l_1-polynomial regression method finds a solution $\mathbf{g}$ such that $\mathbf{A} \cdot \mathbf{g} - \mathbf{b}$ achieves the minimal l_1-norm. Let $p(x)$ denote the polynomial formed using $\mathbf{g}$. After obtaining $p(x)$, the method outputs $\mathsf{Sign}(p(x) - t)$ for some t as the learned hypothesis.

Note that the choice of t is not specified in [11]. So we consider using uniformly sampled t. That is, uniformly sample $t \in [0,1]$ many times and select the one such that $\mathsf{Sign}(p(x) - t)$ is consistent with the most examples. We then show that the t selected this way can indeed satisfy that $\mathsf{Sign}(p(x) - t)$ achieves the error $\text{opt} + \epsilon$. Moreover, we will show that due to the l_1-polynomial strategy, there is at most 2opt-fraction of the examples such that $|p(x) - b| \geq \frac{1}{2}$, which means that there is at least $1 - 2\text{opt}$ fraction such that $|p(x) - b| < \frac{1}{2}$. Thus $\mathsf{Sign}(p(x) - \frac{1}{2})$ is correct on this $1 - 2\text{opt}$ fraction of the examples. This shows that $t = \frac{1}{2}$ is a universal constant such that $\mathsf{Sign}(p(x) - \frac{1}{2})$ achieves the error $2\text{opt} + \epsilon$.

Then we sketch the technique underlying the second part. By using the uniform approximations for OR and AND operations in [17] twice, we show that each s-term DNF formula f can be ϵ-uniformly approximated by a polynomial p of degree $O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon))$. This ensures that the expectation of $|f - p|$ is less than ϵ. Then applying the general result in [11], we obtain the learning result for s-term DNF formulae.

1.2 Organization

The rest of this paper is arranged as follows. Section 2 presents the preliminaries used throughout the paper. Section 3 recalls the l_1-polynomial regression method in [11] in which we instantiate the choice of t and show the universality of $\frac{1}{2}$. Section 4 presents the result for learning s-term DNF formulae.

2 Preliminaries

This section contains the notations and definitions used throughout this paper.

2.1 Basic Notions

Let $[n]$ denote the integers in $[1, n]$. Let $\mathbf{Z}, \mathbf{Q}, \mathbf{R}$ denote integers, rational numbers and reals. For any vector $\mathbf{z} = (z_1, \cdots, z_m) \in \mathbf{R}^m$, $\|\mathbf{z}\|_1$ denotes its l_1-norm, defined as $\sum_{i=1}^m |z_i|$. For a vector $\mathbf{v} \in \mathbf{R}^m$ and a set $I \subset [m]$, we denote by $\mathbf{v}_I$ the vector in $\mathbf{R}^m$ which coincides with $\mathbf{v}$ on the indices in I and is extended to zero outside I. We say that a vector $\mathbf{e} \in \mathbf{R}^m$ is s-sparse if the number of non-zero entries of $\mathbf{e}$ is at most s.

Let $\lfloor \cdot \rceil$ denote the operation of rounding to the nearest integer.

For any distribution D over $\{0,1\}^n \times \{0,1\}$, letting D's output be of form (x,b), we will use (x^k, b_k) to denote the output of D in the kth sampling, while we use x_j to denote the jth bit of x, $1 \leq j \leq n$.

Let $(x^1, b_1), \cdots, (x^m, b_m)$ denote m pairs drawn from D independently. We say a function f is consistent with α fraction of the pairs if $|\{k \in [m] : f(x^k) = b_k\}|/m = \alpha$. Following literatures, we say f is consistent with the pairs if $\alpha = 1$ and say it is approximate-consistent if $0 < \alpha < 1$ which differs from 1 by a small quantity.

Let $\mathsf{Sign}(\cdot)$ denote the function that on input y outputs 1 if $y \geq 0$ and outputs 0 otherwise.

For a boolean function class H, and a set S of M points in the input space X, if the restriction of H to the set S computes all 2^M functions on S, we say that H shatters S. The VC-dimension of H is the size of the largest shattered subset of X, also denoted $\mathrm{VCdim}(H)$.

2.2 Agnostic Learning

Informally, in the agnostic learning model [12, 8], there is a class of functions $\mathcal{C}$ which we wish to learn. We consider each function of $\mathcal{C}$ is boolean. Each example-label pair is chosen from a distribution D over $X \times \{0,1\}$ (X denotes the input space). When given many pairs, the learning algorithm is supposed to output a function f that can achieve almost the minimal error among all functions in $\mathcal{C}$ with respect to D.

For any function f, let $\mathrm{er}_D(f)$ denote $\Pr_{(x,b)\leftarrow D}[f(x) \neq b]$. A training sample drawn from D is of form $((x^1, b_1, \cdots, (x^m, b_m))$ where each (x^k, b_k) is drawn from D independently $1 \leq k \leq m$.

▶ **Definition 2.** (Agnostic Learning). Let D be a distribution on $X \times \{0,1\}$ and let $\mathcal{C}$ be a class of boolean functions. We say that an algorithm L agnostically learns $\mathcal{C}$ if L is given (ϵ, δ) and many random example-label pairs drawn from any D, then with probability $1-\delta$, L outputs a hypothesis f such that $\mathrm{er}_D(f) \leq \mathrm{opt} + \epsilon$, where opt denotes $\min_{h\in\mathcal{C}}(\mathrm{er}_D(h))$.

If L can only work under some specific distribution D, we say L agnostically learns $\mathcal{C}$ under D. We refer to ϵ as the accuracy parameter and δ as the confidence parameter.

We also consider a relaxation by only requiring that the f output by L is such that $\mathrm{er}_D(f) \leq O(\mathrm{opt}) + \epsilon$.

The learning algorithm sometimes needs some additional input parameters. For instance, the Low Degree algorithm has as input the maximal Fourier degree. For our learning algorithm for $\mathcal{H}_{n,\mathcal{S}}$ in this paper, it needs to have as input some representation of $\mathcal{S}$.

3 On Learning Boolean Polynomials

In this section we revisit the result of learning boolean polynomials in [11], in which the l_1-polynomial regression method is employed. We recall this method, instantiate one strategy in it and accordingly present the analysis. Moreover, we show that even ignoring this strategy can bring a learning result of error $2\mathrm{opt} + \epsilon$ as well. In Section 3.1 we demonstrate this learning task and introduce the notations. In Section 3.2 we present the the instantiation and analysis for the l_1-polynomial regression method to find hypotheses consistent with given examples. In Section 3.3 we follow the standard way to convert consistent-hypotheses to learned hypotheses.

3.1 Goal and Notations

Let $h : \{0,1\}^n \to \{0,1\}$ be any one in $\mathcal{H}_{n,\mathcal{S}}$, which can be represented as $h(x) = \sum_{S\in\mathcal{S}} g_S \prod_{j\in S} x_j$ over $x_1, \cdots, x_n$, where g_S's denote the coefficients. So the task of learning $\mathcal{H}_{n,\mathcal{S}}$ is to output a boolean function f' (not necessarily in $\mathcal{H}_{n,\mathcal{S}}$) when given many pairs of form (x, b) sampled from any distribution D over $\{0,1\}^n \times \{0,1\}$, such that f' achieves almost the optimal error among all ones in $\mathcal{H}_{n,\mathcal{S}}$. Typically, if $\mathcal{S}$ consists of all S's with $|S| \leq d$, the task is actually the agnostic learning of boolean d-degree polynomials.

Precisely, let $(x^1, b_1), \cdots, (x^m, b_m)$ denote m pairs independently sampled from D. Then the learning goal is, when given (ϵ, δ), with probability $1 - \delta$, to output a hypothesis f' satisfying $\Pr[f'(x) \neq b] \leq \text{opt} + \epsilon$ for $(x, b) \leftarrow D$, where $\text{opt} = \min_{h \in \mathcal{H}_{n,\mathcal{S}}}(\Pr_{(x,b)\leftarrow D}[h(x) \neq b])$.

Assume that $f \in \mathcal{H}_{n,\mathcal{S}}$ is the one satisfying $\text{opt} = \text{er}_D(f)$. For each pair (x^k, b_k), we view b_k as the sum of $f(x^k)$ and an error e_k. That is, $b_k = f(x^k) + e_k$. Thus, each e_k is of value in $\{0, -1, 1\}$, in which $e_k = 0$ indicates $f(x^k) = b_k$ and $e_k = \pm 1$ indicates $f(x^k) = 1 - b_k$. Let x^k_j denote the jth bit of x^k. For (x^k, b^k), we can generate an equality as follows.

$$\sum_{S \in \mathcal{S}} g_S \prod_{j \in S} x^k_j + e_k = b_k, k \in [1, m]$$

Let a^k_S be the value of $\prod_{j \in S} x^k_j$. Then list the m equalities as follows.

$$\begin{cases} \sum_{S \in \mathcal{S}} g_S a^1_S + e_1 = b_1 \\ \cdots\cdots\cdots\cdots\cdots \\ \sum_{S \in \mathcal{S}} g_S a^m_S + e_m = b_m \end{cases} \tag{1}$$

In the above equalities, all g_S's are unknown and the goal of learning is to recover them. Viewing all a^k_S as coefficients, the equalities are linear for the unknown variables g_S's. For convenience, for all $S \in \mathcal{S}$, we use $S_1, \cdots, S_N$ denote all of them where $N = |\mathcal{S}|$.

Let $\mathbf{a}^k$ denote the (row) vector $(a^k_{S_1}, \cdots, a^k_{S_N}) \in \mathbf{Z}^N$. Let $\mathbf{g}$ denote the (column) vector $(g_{S_1}, \cdots, g_{S_N}) \in \mathbf{Z}^N$. Then for the kth example, we have

$$\mathbf{a}^k \cdot \mathbf{g} + e_k = b_k$$

Let $\mathbf{e}$ denote the (column) vector $(e_1, \cdots, e_m) \in \mathbf{Z}^m$. Let $\mathbf{A}$ denote the m by N matrix which rows consist of all $\mathbf{a}^k$'s. Let $\mathbf{b}$ denote the (column) vector $(b_1, \cdots, b_m) \in \mathbf{Z}^m$. Then the m linear equations can be written as

$$\mathbf{A} \cdot \mathbf{g} + \mathbf{e} = \mathbf{b}$$

Then we can define the following problem: find a solution $\mathbf{g}^*$ such that

$$\|\mathbf{A} \cdot \mathbf{g}^* - \mathbf{b}\|_1 = \inf_{\mathbf{g}'} \|\mathbf{A} \cdot \mathbf{g}' - \mathbf{b}\|_1$$

where $\mathbf{g}', \mathbf{g}^*$ should satisfy that each entry of $\mathbf{A} \cdot \mathbf{g}'$ and $\mathbf{A} \cdot \mathbf{g}^*$ is in $[0, 1]$. This problem can be solved using linear programming.

When obtaining a solution $\mathbf{g}^*$, let $\mathbf{z}$ denote $\mathbf{b} - \mathbf{A}\mathbf{g}^*$. Then we can run the remaining strategy of the l_1-polynomial regression to generate a consistent-hypothesis as well as a learned one. In the rest of this section we will formalize these procedures.

3.2 Finding Consistent-Hypotheses

Recall that $(x^1, b_1), \cdots, (x^m, b_m)$ denote m pairs sampled from D independently, $1 \leq k \leq m$, and f is the function in $\mathcal{H}_{n,\mathcal{S}}$ which achieves opt-error with respect to D. Refer to Section 3.1 for the definitions of notations $\mathbf{A}, \mathbf{b}, \mathbf{g}^*, \mathbf{e}, \mathbf{z}$.

Algorithm 1: The consistent-hypothesis-finder.

```
Input:
  - $m$ pairs of form $(x, b)$ drawn from $D$ independently.
  - $\epsilon, \delta$ and the knowledge of $\mathcal{S}$.
Output: a hypothesis $f_0$.
1. Run a $l_1$-polynomial regression algorithm to find a solution $\mathbf{g}^*$ such that

       $\|\mathbf{A} \cdot \mathbf{g}^* - \mathbf{b}\|_1 = \inf_{\mathbf{g}'} \|\mathbf{A} \cdot \mathbf{g}' - \mathbf{b}\|_1$

   where $\mathbf{g}', \mathbf{g}^*$ satisfy that each entry of $\mathbf{A} \cdot \mathbf{g}', \mathbf{A} \cdot \mathbf{g}^*$ is in $[0, 1]$.
   Assume that $\mathbf{g}^*$ consists of all $g_S^*$'s. Let $p(x) = \sum_{S \in \mathcal{S}} g_S^* \prod_{j \in S} x_j$. (Thus $p(x^k) \in$
   $[0, 1]$ for $1 \le k \le m$.)
2. Uniformly sample $t \in (0, 1)$ $O(1 + 1/\epsilon) \ln(\frac{1}{\delta})$ times. Select one $t$ satisfying
   $f_0(x) = \mathsf{Sign}(p(x) - t)$ achieves the minimal empirical error on the $m$ examples
   and finally output $f_0$.
End Algorithm
```

Let I denote the set of the indices $k \in [m]$ on which $e_k \neq 0$. Let $\mu = |I|/m$. (It can be seen that $\mu \approx \text{opt}$.)

First it can be seen that since $\mathbf{e} = \mathbf{b} - \mathbf{A}\mathbf{g}$ and $\mathbf{g}^*$ achieves the minimal $\|\mathbf{b} - \mathbf{A}\mathbf{g}^*\|_1$ among all $\mathbf{g}'$, $\|\mathbf{z}\|_1 \le \|\mathbf{e}\|_1 = |I|$. Then we follow the method of [11] to construct a consistent hypothesis as shown in Algorithm 1, in which we instantiate the second step for determining t.

For distribution D, let $\text{er}_D(h)$ denote $\Pr[h(x) \neq b]$ for $(x, b) \leftarrow D$. Let Z denote pairs $(x^1, b_1), \cdots, (x^m, b_m)$. Then let $\widehat{\text{er}}_Z(h)$ denote $\frac{1}{m}|\{k : h(x^k) \neq b_k\}|$.

▶ **Proposition 3.** *With probability* $1 - \delta$*, the hypothesis* $f_0(x)$ *in Algorithm 1 is such that* $\widehat{\text{er}}_Z(f_0) \le \mu + \mu\epsilon < \mu + \epsilon$.

Proof. Let h denote $\mathsf{Sign}(p(x) - t)$ for uniform t. First using the argument of [11] (the proof of Theorem 5), we have the following claim.

$$\mathbf{E}_t[\widehat{\text{er}}_Z(h)] \le \frac{1}{m} \sum_{k=1}^{m} |p(x^k) - b^k|$$

To see this, $\mathbf{E}_t[\widehat{\text{er}}_Z(h)]$ equals the average sum of the probabilities of all events $h(x^k) \neq b^k$. Thus for each (x^k, b^k), $f_0(x^k) \neq b^k$ if t lies between $p(x^k)$ and b^k. Note that $p(x^k) \in [0, 1]$ and $b^k \in \{0, 1\}$. Hence, for uniform $u \in (0, 1)$, for any k, the probability that t lies in between the two numbers is $|p(x^k) - b^k|$. So the above inequality holds.

Then notice that

$$\frac{1}{m} \sum_{k=1}^{m} |p(x^k) - b^k)| = \frac{1}{m} \sum_{k=1}^{m} |z_k| = \frac{1}{m} \cdot \|\mathbf{z}\|_1 \le \frac{1}{m} \cdot \|\mathbf{e}\|_1 = \frac{|I|}{m} = \mu$$

So $\mathbf{E}_t[\widehat{\text{er}}_Z(h)] \le \mu$. Furthermore, by Markov's inequality, $\Pr[\widehat{\text{er}}_Z(h) > (1+\epsilon)\mu] \le \frac{\mu}{(1+\epsilon)\mu} = \frac{1}{1+\epsilon} = 1 - \frac{\epsilon}{1+\epsilon}$. Thus

$$\Pr[\widehat{\text{er}}_Z(h) \le (1 + \epsilon)\mu] > \frac{\epsilon}{1 + \epsilon}$$

So for $O(1+1/\epsilon) \ln(\frac{1}{\delta})$ times sampling of u, with probability $1-(1-\frac{\epsilon}{1+\epsilon})^{O(1+1/\epsilon) \ln \frac{1}{\delta}} > 1-\delta$, there is at least one u such that $\widehat{\text{er}}_Z(h) \le (1+\epsilon)\mu < \mu+\epsilon$. Then f_0 is this h. The proposition holds. ◀

We remark that Proposition 3 can be extended to any concept class $\mathcal{C}$ that can be l_1- (or l_2) approximated by $\mathcal{H}_{n,\mathcal{S}}$ in expectation as shown [11].

In the following we show that $t = \frac{1}{2}$ is a universal constant such that for any distribution D, letting $f_0 = \mathsf{Sign}(p(x) - \frac{1}{2})$ in Algorithm 1 (ignoring (ϵ, δ) and omitting the second step), the following result holds.

▶ **Proposition 4.** *The hypothesis $f_0(x) = \mathsf{Sign}(p(x) - \frac{1}{2})$ is such that $\widehat{er}_Z(f_0) \leq 2\mu$.*

Proof. Notice that $\mathbf{A} \cdot \mathbf{g}^* = \mathbf{b} - \mathbf{z}$. First since $\|\mathbf{z}\|_1 \leq \|\mathbf{e}\|_1 = \mu m$, there is at most 2μ fraction of $k \in [1, m]$ such that $|z_k| \geq \frac{1}{2}$. That is, there is at least $1 - 2\mu$ fraction of all k's satisfying $|z_k| < \frac{1}{2}$. This means that for this $1 - 2\mu$ fraction of all k's, $p(x^k)$ differs from b^k by a quantity less than $\frac{1}{2}$. This also means that $\lfloor p(x^k) \rceil$ equals b^k for this fraction. We now show this rounding to the closest integers is identical to the sign operation to $p(x^k) - \frac{1}{2}$ for this fraction. It can be seen that if $b^k = 1$, $p(x^k)$ is more than $\frac{1}{2}$. Thus $\lfloor p(x^k) \rceil$ will output 1. In this case $\mathsf{Sign}(p(x^k) - \frac{1}{2})$ outputs 1 either. If $b^k = 0$, $p(x^k)$ is less than $\frac{1}{2}$. Thus $\lfloor p(x^k) \rceil$ will output 0. In this case $\mathsf{Sign}(p(x^k) - \frac{1}{2})$ outputs 0 either. The proposition holds. ◀

3.3 The Learning Result

In the rest of this section we present the required sample complexity and state the learning result. Let $\mathcal{F}_{n,\mathcal{S}}$ denote the boolean function class, in which each one on input $x \in \{0,1\}^n$ first computes $\prod_{j \in S} x_j$ for all $S \in \mathcal{S}$ and compute a halfspace of all $\prod_{j \in S} x_j$. Thus it can be seen that $\mathcal{H}_{n,\mathcal{S}}$ and the output hypotheses of Algorithm 1 are in $\mathcal{F}_{n,\mathcal{S}}$. In the following let us estimate the VC-dimension of $\mathcal{F}_{n,\mathcal{S}}$.

▶ **Proposition 5.** *$\mathcal{F}_{n,\mathcal{S}}$ is contained in the class of 2-level threshold circuits of $|\mathcal{S}| \cdot (n+1)$ weights and thresholds and $|\mathcal{S} + 1|$ computation gates which is of VC-dimension $O(n \cdot |\mathcal{S}| \cdot \log |\mathcal{S}|)$.*

Proof. First each monomial of form $\prod_{j \in S} x_j$ can be computed by an AND gate of $j \leq n$ inputs and each AND gate of n inputs can be computed by a threshold gate of the n inputs and $n+1$ weights and threshold. Thus f can be computed by a 2-level threshold circuits in which the first level computes $\prod_{j \in S} x_j$ for all $S \in \mathcal{S}$ and the second computes the threshold gate above. It can be seen that this circuit is of $O(|\mathcal{S}| \cdot n)$ weights and thresholds and $|\mathcal{S}| + 1$ gates in total. Thus due to [4], the VC dimension of all such circuits is $O(n \cdot |\mathcal{S}| \cdot \log |\mathcal{S}|)$. ◀

Then recall the following result.

▶ **Theorem 6.** *([19]) Let D be any distribution over $\{0,1\}^n \times \{0,1\}$. Let Z denote m pairs independently sampled from D. For $0 < \epsilon < 1$, it holds that for all $h \in \mathcal{F}_{n,\mathcal{S}}$,*

$$\Pr[|er_D(h) - \widehat{er}_Z(h)| \geq \epsilon] \leq \delta, \quad \textit{if } m \geq \frac{64}{\epsilon^2}(2\,VCdim(\mathcal{F}_{n,\mathcal{S}}) \ln(\frac{12}{\epsilon}) + \ln(\frac{4}{\delta}))$$

Suppose that when given Z, $f_0 \in \mathcal{F}_{n,\mathcal{S}}$ is a hypothesis such that $\mathrm{er}_Z(f_0) \leq c \cdot \mathrm{opt} + \epsilon_0$ for some constant c ($c = 1$ in Proposition 3 and $c = 2$ in Proposition 4). Then we have the following result.

▶ **Claim 7.** *When $m \geq \frac{64}{\epsilon^2}(2\,VCdim(\mathcal{F}_{n,\mathcal{S}}) \ln(\frac{12}{\epsilon}) + \ln(\frac{4}{\delta}))$ and let Z, D, f_0 be defined as above, with probability $1 - \delta$, $er_D(f_0) < c \cdot opt + \epsilon_0 + \epsilon$.*

Proof. Given the condition of m, by Theorem 6, we have that with probability $1 - \delta$, $|\mathrm{er}_D(h) - \widehat{\mathrm{er}}_Z(h)| \leq \epsilon$ for all $h \in \mathcal{F}_{n,\mathcal{S}}$. Thus for $f_0 \in \mathcal{F}_{n,\mathcal{S}}$, we have

$$\mathrm{er}_D(f_0) \leq \widehat{\mathrm{er}}_Z(f_0) + \epsilon \leq c \cdot \mathrm{opt} + \epsilon_0 + \epsilon$$

The claim holds. ◀

Combining Proposition 5 and Claim 7, we have the following proposition.

▶ **Proposition 8.** *Choosing* $m \geq O(\frac{1}{\epsilon^2}(n|\mathcal{S}|\log|\mathcal{S}|\ln(\frac{12}{\epsilon}) + \ln(\frac{4}{\delta})))$ *and letting* $f_0 \in \mathcal{F}_{n,\mathcal{S}}$ *be such that* $\widehat{er}_Z(f_0) < c \cdot opt + \epsilon_0$ *where* Z *denotes* m *pairs sampled from* D*, with probability* $1-\delta$*,* $er_D(f_0) < c \cdot opt + \epsilon_0 + \epsilon$*.*

Then we estimate $|I|$ as follows, which will be used in the proof of Proposition 10.

▶ **Claim 9.** *For any* $0 < \delta < 1$*, with probability* $1-\delta$*,* $|I| \leq (opt \cdot m + \sqrt{3\ln\frac{1}{\delta}\cdot opt \cdot m})$*.*

Proof. Let $\xi_k = 1$ if $e_k \neq 0$ and $\xi_k = 0$ if $e_k = 0$ for $1 \leq k \leq m$. Let $X = \sum_{k=1}^{m}\xi_k$. Then $\mathbf{E}[X] = \text{opt}\cdot m$. Due to the Chernoff bound, for any $0 < \lambda < 1$,

$$\Pr[X < (1+\lambda)\mathbf{E}[X]] > 1 - e^{-\lambda^2\mathbf{E}[X]/3}$$

So set $\lambda = \sqrt{3\ln\frac{1}{\delta}}\cdot\frac{1}{\sqrt{\text{opt}\cdot m}}$. Then the above probability formula is simplified to

$$\Pr[X < (\text{opt}\cdot m + \sqrt{3\log\frac{1}{\delta}\cdot\text{opt}\cdot m})] > 1-\delta$$

The claim holds. ◀

Lastly, replace ϵ, δ in Algorithm 1 by $\frac{\epsilon}{3}, \frac{\delta}{3}$. We have the following result.

▶ **Proposition 10.** *Algorithm 1 can with probability at least* $1-\delta$ *output a hypothesis, denoted* f_0 *in time poly*$(|\mathcal{S}|, n, \frac{1}{\epsilon}, \log\frac{1}{\delta})$ *satisfying* $er_D(f_0) \leq c\cdot opt + \epsilon$*, where* $opt = \min_{h\in\mathcal{H}_{n,\mathcal{S}}}(er_D(h))$ *(*$c = 1$ *when using Proposition 3 or* $c = 2$ *when using Proposition 4).*

Proof. By Claim 9, except for probability $\frac{\delta}{3}$, $\mu = |I|/m \leq \text{opt} + \sqrt{3\ln\frac{3}{\delta}\cdot\text{opt}}\cdot m^{-\frac{1}{2}}$. By Proposition 3 (or Proposition 4), except for another $\delta/3$ probability, $\widehat{\text{er}}_Z(f_0) \leq c\mu + \epsilon/3 = c\cdot\text{opt} + O(m^{-1/2}) + \epsilon/3$, where Z denotes the sample consisting of the m pairs. So by Proposition 8, $\text{er}_D(f_0) \leq c\cdot\text{opt} + O(m^{-1/2}) + 2\epsilon/3 < c\cdot\text{opt} + \epsilon$ (where $O(m^{-1/2}) < \epsilon/3$), and the total failure probability is at most δ.

Moreover, $(\mathbf{A}, \mathbf{b})$ can be generated in time polynomial in $(|\mathcal{S}|, m)$, and the l_1-polynomial regression algorithm runs in time polynomial in its input. Thus the time complexity holds. ◀

4 Learning DNF Formulae

In this section we present an agnostic learning result for DNF formulae, as an application of the general result in [11] for all concept classes admitting l_1-approximation with low-degree polynomials in expectation. Recall that s-term DNF formulae can be PAC learned in time $n^{O(n^{1/3}\cdot\log s)}$ [13], and [6] combined with [16] presents a query algorithm to agnostically learn DNFs in time $n^{O(\log(1/\epsilon)\log\log n)}$ under the uniform distribution. We will present an agnostically learning algorithm for s-term DNF formulae ($s < \sqrt{n}$) by showing that such DNF formulae have uniform approximation with low-degree polynomials. First, let us recall the general result in [11] as follows.

▶ **Theorem 11.** *([11]) Let* $\mathcal{C}$ *denote a concept class,* D *be any distribution over* $\{-1,1\}^n \times \{-1,1\}$*. Assume for any hypothesis* $h \in \mathcal{C}$*, there is a polynomial* p *of degree* d *such that* $\mathbf{E}_D[|h(x) - p(x)|] < \epsilon$*. Then there is an algorithm that on input parameters* (ϵ, δ) *and* d*, sufficiently many pairs sampled from* D *independently can with probability* $1-\delta$ *output a hypothesis* f *such that* $er_D(f) \leq opt+\epsilon$ *in time poly*$(n^d, \frac{1}{\epsilon}, \log\frac{1}{\delta})$ *where* $opt = \min_{h\in\mathcal{C}}(er_D(h))$*.*

Let f be a boolean function mapping $\{-1,1\}^n \to \{-1,1\}$. Let p be a degree-$d_{\epsilon'}$ n-variate polynomial mapping $\mathbf{R}^n \to \mathbf{R}$. We say that $p(x)$ ϵ'-uniformly approximates $f(x)$ if $|f(x)-p(x)| \leq \epsilon'$ for any $x \in \{-1,1\}^n$.

In the following we show that each s-term DNF formula f can be $2\epsilon'$-uniformly approximated by a polynomial p of degree $O(\sqrt{n}\cdot s\cdot \log s \log^2(1/\epsilon'))$ for any ϵ'. This implies $\mathbf{E}_D[|f-p|] \leq 2\epsilon'$. Thus by Theorem 11 we obtain the result of agnostically learning s-term DNFs.

Now consider f as a DNF formula which is the OR of s conjunctions $f_1,\cdots,f_s$. W.l.o.g., assume each f_i is the AND of at most n literals in $\{x_1,\cdots,x_n,\overline{x}_1,\cdots,\overline{x}_n\}$. (If it connects more than n literals, then there is an j such that $x_j,\overline{x}_j$ appear in it simultaneously, which means it is always equal to false and thus can be got rid of from f.) In the following we show that f admits a uniform approximation.

▶ **Proposition 12.** *Each s-term DNF formula f can be $2\epsilon'$-uniformly approximated by a polynomial p of degree $O(\sqrt{n}\cdot s\cdot\log s\log^2(1/\epsilon'))$ for any ϵ'.*

Proof. By [17], for each AND of n variables, for any ϵ_0, there is a multi-variate real polynomial that can ϵ_0-uniformly approximate it. That is, for each f_i, there is a $p_i(x)$ of degree $O(\sqrt{n}\log(1/\epsilon_0))$ satisfying $|p_i(x)-f_i(x)| \leq \epsilon_0$ for all $x \in \{-1,1\}^n$. It can be seen that $f_i(x)/p_i(x) \in [\frac{1}{1+\epsilon_0},\frac{1}{1-\epsilon_0}]$ for $f_i(x)=\pm 1$ and for each i.

Notice that $\frac{1}{1+\epsilon_0} > 1-\epsilon_0$. Since $(1-\epsilon_0)(1+2\epsilon_0) = 1+\epsilon_0-2\epsilon_0^2$, choosing $\epsilon_0 < \frac{1}{n^2}$, we have that

$$\frac{1}{1-\epsilon_0} = \frac{1+2\epsilon_0}{1+\epsilon_0-2\epsilon_0^2} < 1+2\epsilon_0$$

So $f_i(x)/p_i(x) \in (1-\epsilon_0, 1+2\epsilon_0)$ for all i's. Let $f_i(x)/p_i(x) = 1+\Delta_i(x)$. Then $\Delta_i \in (-\epsilon_0, 2\epsilon_0)$.

Since f is OR of $f_1,\cdots,f_s$, using [17] again, we have that there exists an s-variate multi-linear polynomial $P(f_1,\cdots,f_s)$ of degree $O(\sqrt{s}\log(1/\epsilon'))$ such that $|f(f_1,\cdots,f_s) - P(f_1,\cdots,f_s)| \leq \epsilon'$ for any $f_1,\cdots,f_s$. Denote the Fourier expansion of $P(f_1,\cdots,f_s)$ by $\sum_{|S|\leq O(\sqrt{s}\log(1/\epsilon'))}\beta_S\prod_{j\in S} f_j$, where each $S \subset [n]$ and β_S's are coefficients each of which is less than a constant. Thus we have

$$\begin{aligned}
P(f_1,\cdots,f_s) &= \sum_{|S|\leq O(\sqrt{s}\log(1/\epsilon'))}\beta_S\prod_{j\in S} f_j = \sum_{|S|\leq O(\sqrt{s}\log(1/\epsilon'))}\beta_S\prod_{j\in S}(p_j\cdot(1+\Delta_j)) \\
&= \sum_{|S|\leq O(\sqrt{s}\log(1/\epsilon'))}\beta_S\prod_{j\in S} p_j\cdot\prod_{j\in S}(1+\Delta_j) \\
&= \sum_{|S|\leq O(\sqrt{s}\log(1/\epsilon'))}\beta_S\prod_{j\in S} p_j\cdot(1+\sum_{j=1}^{|S|}\Delta_j + O(\max_j(\Delta_j))) \\
&= P(p_1,\cdots,p_s) + \sum_{|S|\leq O(\sqrt{s}\log(1/\epsilon'))}\beta_S\prod_{j\in S} p_j(\sum_{j=1}^{|S|}\Delta_j + O(\max_j(\Delta_j)))
\end{aligned}$$

When $\epsilon_0\cdot n\cdot\binom{s}{O(\sqrt{s}\log(1/\epsilon'))} < \epsilon'/n$, the second addend in the right side of the last equality is less than ϵ'. Thus in the beginning, we would choose

$$\epsilon_0 < \frac{\epsilon'}{n^2}\cdot s^{-O(\sqrt{s})\log(1/\epsilon')}$$

Then each $p_i(x)$ is of degree $O(\sqrt{n}\log(1/\epsilon_0)) = O(\sqrt{n}\cdot(\log n + \sqrt{s}\log s\log(1/\epsilon'))) = O(\sqrt{ns}\log s\log(1/\epsilon'))$ (when $\sqrt{s} > \log n$).

More importantly, we have $|P(f_1, \cdots, f_s) - P(p_1, \cdots, p_s)| < \epsilon'$, which shows that $|f(f_1(x), \cdots, f_s(x)) - P(p_1(x), \cdots, p_s(x))| < 2\epsilon'$ for any $x \in \{-1, 1\}^n$.

Notice that $P(p_1(x), \cdots, p_s(x))$ is actually a multi-linear polynomial on x of degree $O(\sqrt{ns} \log s \log(1/\epsilon')) \cdot O(\sqrt{s} \log(1/\epsilon')) = O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon'))$. The proposition holds. ◀

Thus we have the following learning result.

▶ **Proposition 13.** *For each s, for any (ϵ, δ), all s-term DNF formulae can be agnostically learned to error $opt + \epsilon$ and confidence δ in time poly$(n^d, \frac{1}{\epsilon}, \log \frac{1}{\delta})$, where $d = O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon))$.*

Proof. By Proposition 12, $\mathbf{E}_D[|f(x) - P(p_1(x), \cdots, p_s(x))|] \leq 2\epsilon'$ for any $\epsilon' > 0$ where $P(p_1(x), \cdots, p_s(x)$ is of degree $O(\sqrt{n} \cdot s \cdot \log s \log^2(1/\epsilon'))$. Thus, choosing $\epsilon = 2\epsilon'$, by Theorem 11, the proposition holds. ◀

Acknowledgements. The author is grateful to the reviewers of ISAAC 2017 for their useful comments.

References

1 Pranjal Awasthi, Maria-Florina Balcan, and Philip M. Long. The power of localization for efficiently learning linear separators with noise. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 449–458. ACM, 2014. doi:10.1145/2591796.2591839.

2 Richard Beigel. When do extra majority gates help? polylog(n) majority gates are equivalent to one. *Computational Complexity*, 4:314–324, 1994.

3 Nader H. Bshouty and Christino Tamon. On the fourier spectrum of monotone functions. *J. ACM*, 43(4):747–770, 1996. doi:10.1145/234533.234564.

4 T. M. Cover. Capacity problems for linear machines. *Patten Recognition*, pages 283–289, 1968.

5 Amit Daniely. A PTAS for agnostically learning halfspaces. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 484–502. JMLR.org, 2015. URL: http://jmlr.org/proceedings/papers/v40/Daniely15.html.

6 Parikshit Gopalan, Adam Tauman Kalai, and Adam R. Klivans. Agnostically learning decision trees. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 527–536. ACM, 2008. doi:10.1145/1374376.1374451.

7 Parikshit Gopalan and Rocco A. Servedio. Learning and lower bounds for ac^0 with threshold gates. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 588–601. Springer, 2010.

8 David Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150, 1992.

9 Lisa Hellerstein and Rocco A. Servedio. On PAC learning algorithms for rich boolean function classes. *Theor. Comput. Sci.*, 384(1):66–76, 2007. doi:10.1016/j.tcs.2007.05.018.

10 Jeffrey C. Jackson, Adam Klivans, and Rocco A. Servedio. Learnability beyond ac0. In *IEEE Conference on Computational Complexity*, page 26. IEEE Computer Society, 2002.

11 Adam Tauman Kalai, Adam R. Klivans, Yishay Mansour, and Rocco A. Servedio. Agnostically learning halfspaces. *SIAM J. Comput.*, 37(6):1777–1805, 2008. doi:10.1137/060649057.

12 Michael J. Kearns, Robert E. Schapire, and Linda Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

13 Adam R. Klivans, Ryan O'Donnell, and Rocco A. Servedio. Learning intersections and thresholds of halfspaces. *J. Comput. Syst. Sci.*, 68(4):808–840, 2004. doi:10.1016/j.jcss.2003.11.002.

14 Adam R. Klivans and Rocco A. Servedio. Learning DNF in time $2^{\tilde{o}(n^{1/3})}$. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 258–265. ACM, 2001. doi:10.1145/380752.380809.

15 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

16 Yishay Mansour. An o(nˆ(log log n)) learning algorithm for DNT under the uniform distribution. *J. Comput. Syst. Sci.*, 50(3):543–550, 1995. doi:10.1006/jcss.1995.1043.

17 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994. doi:10.1007/BF01263419.

18 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

19 V.N.Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

Succinct Color Searching in One Dimension*

Hicham El-Zein[1], J. Ian Munro[2], and Yakov Nekrich[3]

1 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
helzein@uwaterloo.ca
2 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
imunro@uwaterloo.ca
3 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
ynekrich@uwaterloo.ca

Abstract

In this paper we study succinct data structures for one-dimensional color reporting and color counting problems. We are given a set of n points with integer coordinates in the range $[1, m]$ and every point is assigned a color from the set $\{1, \ldots, \sigma\}$. A color reporting query asks for the list of distinct colors that occur in a query interval $[a, b]$ and a color counting query asks for the number of distinct colors in $[a, b]$.

We describe a succinct data structure that answers approximate color counting queries in $O(1)$ time and uses $\mathcal{B}(n, m) + O(n) + o(\mathcal{B}(n, m))$ bits, where $\mathcal{B}(n, m)$ is the minimum number of bits required to represent an arbitrary set of size n from a universe of m elements. Thus we show, somewhat counterintuitively, that it is not necessary to store colors of points in order to answer approximate color counting queries. In the special case when points are in the rank space (i.e., when $n = m$), our data structure needs only $O(n)$ bits. Also, we show that $\Omega(n)$ bits are necessary in that case.

Then we turn to succinct data structures for color reporting. We describe a data structure that uses $\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m)) + o(n \lg \sigma)$ bits and answers queries in $O(k+1)$ time, where k is the number of colors in the answer, and $nH_d(S)$ ($d = \log_\sigma n$) is the d-th order empirical entropy of the color sequence. Finally, we consider succinct color reporting under restricted updates. Our dynamic data structure uses $nH_d(S) + o(n \lg \sigma)$ bits and supports queries in $O(k + 1)$ time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Succinct Data Structures, Range Searching, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.30

1 Introduction and Motivation

Range search problems are problems where a point set is preprocessed so that certain information about a query region can be efficiently computed. These problems are of fundamental importance in computational geometry, both in the study of their optimality with respect to space and query time, and as tools employed to provide efficient solutions to various geometric problems. In this paper we focus on the following two problems. One dimensional color range reporting (counting): Given a set of colored points $\mathcal{P}$, preprocess $\mathcal{P}$ into an efficient data structure so that for any range $\mathcal{Q} = [a, b]$ the distinct colors contained in $\mathcal{P} \cap \mathcal{Q}$ can be reported (counted).

We study both problems in the context of succinctness, where the goal is to achieve the optimal space requirement plus a lower order term, while maintaining fast query time.

* This work was sponsored by the NSERC of Canada and the Canada Research Chairs Program.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 30; pp. 30:1–30:11
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Designing succinct data structures is an area of interest in theory and practice motivated by the need of storing large amount of data using the smallest space possible. In recent years there has been a surge of interest in succinct data structures for computational geometry [5, 8, 15]. For further reading and more in-depth coverage of succinct data structures we refer the reader to the survey by Munro and Rao [16].

Previous Work. If the input points are in the rank space, one-dimensional color reporting queries can be answered in $O(k+1)$ time using $nH_d(S)+o(n)\lg\sigma+O(n\lg\lg\sigma)$ bits [2, 4, 6], where $d=o(\log_\sigma n)$ and $H_d(S)$ is the *d-th* order empirical entropy of the given sequence of colors S. In the general case, one-dimensional color reporting queries can be answered in $O(\lg n+k)$ time in the static and dynamic scenarios as shown by Janardan and Lopez [17] and Gupta et al. [26]. Muthukrishnan [20] later described a static $O(n)$ space data structure that answers queries in $O(k+1)$ time when all point coordinates are bounded by n. His result implies an $O(n)$-words data structure that answer queries in $O(\min(\lg\lg m, \sqrt{\lg n/\lg\lg n})+k)$ time using the reduction-to-rank-space technique, where $O(\min(\lg\lg m, \sqrt{\lg n/\lg\lg n}))$ is the time needed to answer a predecessor query [27, 10]. A dynamic data structure of Mortensen [18] supports queries and updates in $O(\lg\lg n+k)$ and $O(\lg\lg n)$ time respectively if the values of all elements are bounded by n. Finally, Nekrich and Vitter [22] presented an $O(n)$-words static data structure that answers queries in $O(k+1)$ time; their result is valid even in the case when point are not in the rank space. They also presented a dynamic version of their structure that uses the same space and achieves the same query time while handling updates in $O(\lg^\varepsilon n)$ time.

One-dimensional color counting in the rank space was studied by Gagie et al. [11]. They gave a data structure that answers queries in $O(\lg^{1+\varepsilon} n)$ time for any constant $\varepsilon>0$ and uses $nH_0(S)+O(n)+o(nH_0(S))$ bits. Nekrich [21] described a data structure that uses $O(n\lg n)$ bits and answers color counting queries in $O(\lg k/\lg\lg n)$ time, where k is the number of colors. A lower bound that follows from the predecessor problem [1, 3] holds for exact one-dimensional color counting, and does not permit constant query time for a data structure with space bounded by a polynomial function of n. We circumvent this lower bound by focusing on approximate color counting. If we combine a reduction of one-dimensional color counting to point counting in 2D with the result of Chan and Wilkinson [7], we obtain a data structure that uses $O(n\lg n)$ bits and answer approximate color counting queries in $O(\lg^\varepsilon n)$ time. The data structure of Nekrich [7] also uses $O(n\lg n)$ bits but answers approximate color counting queries in $O(1)$ time. In both [21] and [7] it is assumed that points are in the rank space. In the general case, Saladi [24] presented a data structure that uses $O(n)$ words and answers queries in $O(\lg\lg U)$ time.

Our Results. We focus on studying one-dimensional color reporting and counting in the succinct scenario. In Section 2 we solve an open problem from [24] by presenting a data structure that answers approximate color counting queries in optimal $O(1)$ time. Our data structure uses $\mathcal{B}(n,m)+O(n)+o(\mathcal{B}(n,m))$ bits, where $\mathcal{B}(n,m)=n\lg(m/n)$ is the minimum number of bits required to store a set of size n from a universe of m elements. Thus, we demonstrate that is not necessary to store the colors of points in order to answer approximate color counting queries. If points are in the rank space, our data structure needs only $O(n)$ bits and does not require access to the original data set. That is, similar to data structures for answering range minimum queries [9] that can answer queries without storing the original data set, we can construct a data structure for a colored set of points S and discard the set S.

Using our data structure, we are still able to obtain a constant factor approximation on the number of colors in $S \cap [a, b]$ for an arbitrary query interval $[a, b]$.

Then we turn to the problem of reporting colors using succinct space. We describe a data structure that answers color reporting queries in $O(k + 1)$ time while using $\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m) + n \lg \sigma)$ bits in Section 4. This result is a succinct counterpart of the data structure from [22] that also achieves optimal query time but uses $O(n \lg n)$ bits.

Finally we consider dynamic succinct color reporting in the rank space. We present a succinct data structure that answers color reporting queries in optimal $O(k + 1)$ time and updates in $O(\lg n)$ time while using $nH_d(S) + o(n \lg \sigma)$ bits. Our data structure supports an update operation that changes the color of a point in $O(\lg n)$ time.

Applications. Color reporting and counting queries are related to problems that arise in string processing and databases. Color searching queries are helpful when we are interested in (the number of) distinct object categories in a query range or look for distinct documents that contain a query substring. One prominent example is the document counting queries on a collection of documents. We keep documents (strings) d_1, . . ., d_D in a data structure so that for any query string P the number of documents that contain P can be calculated. This problem can be solved by answering color counting queries on the so called document array; see [20, 12] for a detailed description. The document array, however, needs $O(n \lg D)$ bits of space in the worst case. If the number of documents is large and the alphabet size is small, the space usage of the document array can be significantly larger than the space needed to store the document collection. Using the result of Theorem 4, we can answer approximate document counting queries using $O(n)$ additional bits.

In this paper we assume that the reader is familiar with basic concepts of succinct data structures and range reporting.

2 Approximate Color Range Counting

In this section we present a data structure that uses $\mathcal{B}(n, m) + O(n) + o(\mathcal{B}(n, m))$ bits of space and answers approximate color counting queries in constant time. A color range counting query for an interval returns the number of distinct colors contained within the interval. For any constant $\varepsilon > 0$, our color range counting data structure returns in constant time an approximate answer of at most $(1 + \varepsilon)$ of the correct answer.

2.1 Approximate Color Range Counting in Rank Space

We begin by describing a data structure for the problem in the special case when the input points are in the rank space. The input consists of a sequence $S = s_1, \ldots, s_n$ of n colors. A query is a range $[a, b]$ where $a, b \in [n]$, and the answer is a $(1 + \varepsilon)$-approximation of the number of distinct colors found in $s_a, \ldots, s_b$.

2.1.1 Space Inefficient Solution

First we describe a space inefficient solution that requires $O(n \lg^3 n)$ bits of space and answers one-dimensional approximate color counting queries in constant time.

Consider the balanced binary tree $\mathcal{T}$, where every leaf of $\mathcal{T}$ corresponds to an element of S, and every internal node has two children. Given a node $u \in \mathcal{T}$, $u_l(u_r)$ denotes the left(right) child of u, $S(u)$ denotes the set of all elements stored in the leaf descendants of u, and $a_u(b_u)$ denotes the rightmost(leftmost) element in $S(u_l)(S(u_r))$.

Let $\delta = 1 + \varepsilon$. For each node $u \in \mathcal{T}$ we store the unique values $l_1, \ldots, l_{\log_\delta n}$ in a fusion tree [10], where l_i ($1 \leq i \leq \log_\delta n$) is the maximum value satisfying the condition that $s_{l_i}, \ldots, s_{a_u}$ contains δ^i unique colors. Also, for each node $u \in \mathcal{T}$ and each i ($1 \leq i \leq \log_\delta n$) we store the unique values $r_{i1}, \ldots, r_{i \log_\delta n}$ in a fusion tree [10], where r_{ij} ($1 \leq j \leq \log_\delta n$) is the minimum value satisfying the condition that $s_{b_u}, \ldots, s_{r_j}$ contains δ^j unique colors that are not present in $s_{l_i}, \ldots, s_{a_u}$.

Query. Given a query $[a, b]$ we find the lowest common ancestor u of a and b in $\mathcal{T}$. We query the fusion tree stored on $l_1, \ldots, l_{\log_\delta n}$ to find the predecessor l_i of a, then we query the fusion tree stored on $r_{i1}, \ldots, r_{i \log_\delta n}$ and find the successor r_{ij} of b. Finally we return $\delta^i + \delta^j$ as an estimate for the number of distinct colors in $[a, b]$.

▶ **Lemma 1.** *The algorithm described above returns a $(1 + \varepsilon)$-approximation of the number of distinct colors in $s_a, \ldots, s_b$.*

Proof. Denote by x the number of distinct colors in $s_a, \ldots, s_{a_u}$ and y the number of distinct colors in $s_{b_u}, \ldots, s_b$ that are not found in $s_a, \ldots, s_{a_u}$. Let y' denote the number of colors in s_{b_u}, …, s_b that do not occur in $l_i, \ldots, s_{a_u}$. By the definition of l_i and r_{ij}, $x \leq \delta^i \leq \delta \cdot x$ and $y' \leq \delta^j \leq \delta \cdot y'$. Since $y' \leq y$, $\delta^j \leq \delta \cdot y$. Hence $\delta^i + \delta^j \leq \delta(x + y)$. There are at most $\delta^i - x$ colors that occur in $l_i, \ldots, s_{a_u}$, but do not occur in $s_a, \ldots, s_{a_u}$. Hence $y - (\delta^i - x) \leq y'$ and $y - (\delta^i - x) \leq \delta^j$. If we add δ^i to both parts of the latter inequality, we obtain $y + x \leq \delta^j + \delta^i$. Summing up

$$x + y \leq \delta^i + \delta^j \leq \delta(x + y)$$

which completes the proof. ◀

▶ **Theorem 2.** *There exists an $O(n \lg^3 n)$-bit data structure that supports one-dimensional $(1 + \varepsilon)$-approximate color range counting queries in constant time when the input points are in the rank space.*

2.1.2 Lower Bound

Next, we show using a simple proof that $\Omega(n)$ bits are required for any data structure that answers one-dimensional $(1 + \varepsilon)$-approximate color range counting queries in the rank space.

We assume without loss of generality that $\sigma > \lfloor 1 + \varepsilon \rfloor$, otherwise no data structure is needed since returning σ for any query would be a correct $(1 + \varepsilon)$-approximation of the exact answer. Moreover, denote by $c_1, c_2, \ldots, c_k$ the first $k = \lfloor 1 + \varepsilon \rfloor + 1$ colors. Divide a sequence S of size n to n/k blocks each of size k. We say that S satisfies property $(*)$ if for each block b in S one of the following two conditions hold:

- either b consists of the color c_1 repeated k times,
- or $b = c_1, c_2, \ldots, c_k$.

Clearly, the number of sequences that satisfy $(*)$ is $2^{(n/k)}$ since there exist n/k blocks in a sequence of size n and each block can have one of two different values. Moreover for any two distinct sequences S_1 and S_2 satisfying $*$ differing at block b, there exist at least one $(1 + \varepsilon)$-approximate range counting query, namely the query that asks for the number of different colors in b, that will return different values. Thus, the information theoretic lower bound for storing a one-dimensional $(1 + \varepsilon)$-approximate range counting data structure is $\Omega(\lg 2^{(n/k)}) = \Omega(n/k) = \Omega(n/\epsilon)$ bits.

▶ **Theorem 3.** *Any one-dimensional $(1 + \varepsilon)$-approximate range counting data structure requires $\Omega(n/\epsilon)$ bits.*

2.1.3 Compact Data Structure

In this subsection we show how to make the data structure of Theorem 2 compact by bootstrapping. Let $\delta = 1 + \varepsilon$. We define the functions $f(n) = \lg^4 n$, $f^{(h)}(n) = f^{(h-1)}(f(n))$. The function $f^*(n)$ is defined as $f^*(n) = 1 + f^*(f(n))$ for $n > 2^{16}$ and $f^*(n) = 1$ otherwise. We start by modifying the tree $\mathcal{T}$ so that each leaf of $\mathcal{T}$ corresponds to a block of $f(n)$ consecutive elements of S (instead of a single element of S). Then, we define the family of trees $\mathcal{T}_{ij}$ where $1 \leq i \leq f^*(n)$ and $1 \leq j \leq n/f^{(i)}(n)$ as follows. Tree $\mathcal{T}_{ij}$ spans the i^{th} block of S of size $f^{(i)}(n)$ (i.e. $s_{((i-1)f^{(i)}(n)+1)}, \ldots, s_{(if^{(i)}(n))}$) and each leaf of $\mathcal{T}_{ij}$ correspond to a block of $f^{(i+1)}(n)$ consecutive elements. For each node $u \in \mathcal{T}_{ij}$ we store in separate fusion trees the sets of values: $\{l_p | 1 \leq p \leq \log_\delta f^{(i)}(n)\}$, and for each $1 \leq p \leq \log_\delta f^i(n)$ the set $\{r_{pq}|, 1 \leq q \leq \log_\delta f^{(i)}(n)\}$ as defined in Section 2.1.1. Finally, for every two indices a and b satisfying $1 \leq a \leq b \leq f(n)$ we store in a table B the index i such that a and b are in the same block of size $f^i(n)$ but in different blocks of size $f^{i+1}(n)$. In other words, i must satisfy the following conditions $\lfloor a/f^{(i)}(n) \rfloor = \lfloor b/f^{(i)}(n) \rfloor$ and $\lfloor a/f^{(i+1)}(n) \rfloor \neq \lfloor b/f^{(i+1)}(n) \rfloor$

Space Analysis. The number of nodes in $\mathcal{T}$ is reduced to $n/f(n)$ and the space used by $\mathcal{T}$ and fusion trees stored in its nodes is $O(n/\lg n)$ bits. The number of nodes in $\mathcal{T}_{ij}$ is $f^{(i)}(n)/f^{(i+1)}(n)$ and the space used by $\mathcal{T}_{ij}$ and fusion trees stored in its nodes is $O(f^{(i)}(n)/\lg(f^{(i)}(n)))$ bits. Thus, the total space used by all such trees is:

$$\begin{aligned}\sum_{i=1}^{f^*(n)} \left(\sum_{j=1}^{n/f^{(i)}(n)} O\Big(f^{(i)}(n)/\lg(f^{(i)}(n))\Big) \right) &= \sum_{i=1}^{f^*(n)} \Big(n/f^{(i)}(n) \cdot O\Big(f^{(i)}(n)/\lg(f^{(i)}(n))\Big)\Big) \\ &= \sum_{i=1}^{f^*(n)} O\Big(n/\lg(f^{(i)}(n))\Big) \\ &= n \sum_{i=1}^{f^*(n)} O\Big(1/\lg(f^{(i)}(n))\Big) \\ &= O(n)\end{aligned}$$

Finally, the table B uses $o(n)$ bits. Thus, the total space used is $O(n)$ bits.

Query. Given a query $[a, b]$, if a and b are in two different blocks of size $f(n)$, we can answer queries using $\mathcal{T}$ in the same way as described in Subsection 2.1.1. Otherwise, we query B on values ($a \mod f(n)$) and ($b \mod f(n)$) to find the index i satisfying the condition that a and b are in the same block of size $f^{(i)}(n)$ but in different blocks of size $f^{(i+1)}(n)$. Finally, we query $\mathcal{T}_{i\lfloor a/f^{(i)}n \rfloor}$ as we query $\mathcal{T}$.

▶ **Theorem 4.** *There exists a compact $O(n)$-bit data structure that supports one-dimensional $(1+\varepsilon)$-approximate color range counting queries in constant time when the input points are in the rank space.*

3 General Approximate Range Counting

In this section, we present a data structure that uses $\mathcal{B}(n,m) + O(n) + o(\mathcal{B}(n,m))$ bits of space and answers $(1+\varepsilon)$-approximate color counting queries in constant time.

Let $\delta = 1 + \varepsilon$ and let $x_1, \ldots, x_n$ be the coordinates of the n given colored points $\mathcal{P}$ in sorted order. Denote by $\mathcal{P}_{\lceil \lg^3 n \rceil}$ the set of points whose x-coordinate rank is a multiple of

$\lceil \lg^3 n \rceil$. For each point $p \in \mathcal{P}$ denote by $\mathrm{L}(p)$ the set of points to the left of p, and by $\mathrm{R}(p)$ the set of points to the right of p.

For each point $p \in \mathcal{P}_{\lceil \lg^3 n \rceil}$ we store in a fusion tree [10] the unique values $l_1, \ldots, l_{\log_\delta n}$ where l_i ($i \in [\log_\delta n]$) is the maximum value satisfying the condition that $s_{l_i}, \ldots, s_p$ contains δ^i unique colors. Also, for each point $p \in \mathcal{P}_{\lceil \lg^3 n \rceil}$ and each $i \in [\log_\delta n]$ we store in a fusion tree [10] the unique values $r_{i1}, \ldots, r_{i \log_\delta n}$ where r_{ij} ($j \in [\log_\delta n]$) is the minimum value satisfying the condition that $s_{p+1}, \ldots, s_{r_j}$ contains δ^j unique colors not present in $s_{l_i}, \ldots, s_p$. We also store a succinct point reporting structure [13] on $\mathcal{P}_{\lceil \lg^3 n \rceil}$.

Next, we divide $x_1, \ldots, x_n$ into $n/\lceil \lg^3 n \rceil$ blocks each of size $\lceil \lg^3 n \rceil$, except for the last one. Using $O(n \lg^{4/5} m)$ bits [23] we store predecessor and successor data structures for each block independently. Since the size of each block is at most $\lceil \lg^3 n \rceil$, answering predecessor and successor queries within a block takes constant time. Finally, we store in $O(n)$ bits the compact data structure from Theorem 4 for answering queries in the rank space.

Query. Given a query $[a, b]$ we check if a point $p \in \mathcal{P}_{\lceil \lg^3 n \rceil}$ is in $[a, b]$. If so, we query the fusion tree stored on $l_1, \ldots, l_{\log_{1+\varepsilon} n}$ to find l_i the predecessor of a, then we query the fusion tree stored on $r_{i1}, \ldots, r_{i \log_{1+\varepsilon} n}$ to find r_{ij} the successor of b, afterwards we return $(1+\varepsilon)^i + (1+\varepsilon)^j$.

If such a point p does not exist, then both a and b are in one of the blocks whose size is $\lceil \lg^3 n \rceil$. Using the reporting data structure stored on $\mathcal{P}$ we get the rank of an arbitrary point in $[a, b]$ then determine which block does a and b belong to. Afterwards, using the predecessor and successor structures, we determine the rank of a and b. Since the query is now reduced to the rank space, we can answer it in constant time.

▶ **Theorem 5.** *There exists an $(\mathcal{B}(n, m) + O(n) + O(n \lg^{4/5} m))$-bit data structure that supports one-dimensional $(1+\varepsilon)$-approximate color range counting queries in constant time.*

Next, we describe how to reduce the space of the predecessor and successor data structures. We use a well known trick and split the universe $[m]$ into n subranges $r_1, \ldots, r_n$ each of size m/n. We also use succinct rank and select data structures that store a bit vector of size n using $n + o(n)$ bits and answers rank and select queries in constant time [19]. For each non-empty subrange r_i we store a predecessor and successor structure for every block of $\lg^2 n$ consecutive elements and a point reporting structure P_i on all the points within r_i. These structures are stored consecutively in an array A. To locate the data structures for any range r_i within A, we count the number of points in the ranges r_j for $j < i$ then scale that number. For that purpose, we construct a bit vector B of size $2n$ bits, with rank and select queries, that stores a zero for each range r_i followed by n_i ones, where n_i is the number of points in the range r_i. To count the number of points preceding r_i, we use a select query to get the position k of the i^{th} zero in B, then with a rank query we count the number of ones before position k.

Given a non-empty query range $[a, b]$ such that there exist at most $\lg^3 n$ points between a and b, a belongs to r_i where $i = \lfloor a/(m/n) \rfloor$ and b belongs to r_j where $j = \lfloor b/(m/n) \rfloor$, we find the rank of a in the following manner. First, we map a to $a' = a - im/n$ and b to $b' = b - jm/n$. If the range $[a', m/n]$ is empty in P_i, we use rank and select queries to get s the number of ones before the $(i+1)^{\text{th}}$ zero in B, the rank of a will be $s + 1$. Otherwise, we find a point p in P_i within the range $[a', m/n]$ if i and j are different or within the range $[a', b']$ if i and j are the same. If p's rank within r_i is k, we query the $\lfloor k/\lg^3 n \rfloor$ successor data structure to find the rank of a' in r_i. Then, we add the number of points occurring in each range r_l where $l < i$ to this rank to get the rank of a. We obtain the rank of b in a similar manner.

The extra space used is $o(\mathcal{B}(n, m))$ bits for the point reporting structures stored on the ranges $r_1, \ldots, r_n$, $O(n \lg^{4/5}(m/n)) = o(\mathcal{B}(n, m))$ bits for the predecessor and successor data structures, and $O(n)$ bits for the bit vector B.

▶ **Theorem 6.** *There exists an $(\mathcal{B}(n, m) + O(n) + o(\mathcal{B}(n, m))$-bit data structure that supports one-dimensional $(1 + \varepsilon)$-approximate color range counting queries in constant time.*

4 1D Color Range Reporting

Using similar techniques to those used in the previous section, we present in this section a succinct data structure that uses $\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m) + n \lg \sigma)$ bits of space and answers color reporting queries in optimal $O(k + 1)$ time.

If the input points are in the rank space (i.e. the x-coordinates of the input points are 1,...,n and the input consists of a sequence $S = s_1, \ldots, s_n$ of n colors, a query is a range $[a, b]$ where $a, b \in [n]$, and the answer is the distinct colors found in $s_a, \ldots, s_b$), one-dimensional color range reporting can be solved in $O(k + 1)$ time using $nH_d(S) + o(n) \lg \sigma + O(n \lg \lg \sigma)$ bits [2, 4, 6].

This solution can be extended to general one-dimensional range reporting by storing the x-coordinates of the points in sorted order in an indexable dictionary that supports select queries in constant time using $\mathcal{B}(n, m) + o(\mathcal{B}(n, m))$ bits [25] in addition to the data structure described in [2, 4, 6]. We can find the predecessor or successor of any x-coordinate in $O(\lg n)$ time by answering $O(\lg n)$ select queries. Hence, we can reduce any query $[a, b]$ to the rank space in $O(\lg n)$ additional time.

▶ **Theorem 7.** *There exists an $(\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m) + n \lg \sigma))$-space data structure that supports one-dimensional color range reporting queries in $O(\lg n + k)$time.*

4.1 Improved Data Structure

Next, we show how to improve the query time obtained from Theorem 7 to $O(k + 1)$, while using the same amount of space.

Let $x_1, \ldots, x_n$ be the coordinates in sorted order of the n given colored points $\mathcal{P}$. We denote by $\mathcal{P}_{\lceil \lg^2 n \rceil}$ the set of points whose x-coordinate rank is a multiple of $\lceil \lg^2 n \rceil$. For each point $p \in \mathcal{P}$ we denote by $\mathrm{L}(p)$ the set of points to the left of p, and by $\mathrm{R}(p)$ the set of points to the right of p. For every color z the set $\mathrm{Min}(p)$ contains the minimal element $e \in \mathrm{L}(p)$ of color z, and the set $\mathrm{Max}(p)$ contains the maximal element $e \in \mathrm{R}(p)$ of color z.

Data Structure. For each point $p \in \mathcal{P}_{\lceil \lg^2 n \rceil}$, we store the smallest $\lceil \lg n \rceil$ elements of $\mathrm{Min}(p)$ and the largest $\lceil \lg n \rceil$ elements of $\mathrm{Max}(p)$. We also store two succinct one-dimensional point reporting data structures [13], one on every point in $\mathcal{P}$, and the other on every point in $\mathcal{P}_{\lceil \lg^2 n \rceil}$. Next, we store a data structure similar to the one used in subsection 3 that can find in constant time the ranks of a query $[a, b]$ if $[a, b]$ is not empty, and a and b belong to the same block of size $\lg^2 n$, Finally, we store the data structure from Theorem 7.

Answering Queries. We report all colors in a query range $[a, b]$ as follows. Using the reporting data structure stored on $\mathcal{P}_{\lceil \lg^2 n \rceil}$, we search for some $p \in \mathcal{P}_{\lceil \lg^2 n \rceil} \cap [a, b]$.

If such a point p exist, we traverse the list $\mathrm{L}(p)$ until an element $p' > b$ is found or the end of $\mathrm{L}(p)$ is reached. We also traverse the list $\mathrm{R}(p)$ until an element $p' < a$ is found or the end of $\mathrm{R}(p)$ is reached. If we reach neither the end of $\mathrm{L}(p)$ nor the end of $\mathrm{R}(p)$, then all distinct colors in $[a, b]$ are reported. Otherwise, the range $[a, b]$ contains more than $\lg n$ distinct colors. In that case we use the data structure from Theorem 7.

If a and b belong to a continuous block of $\lg^2 n$ points, we find their ranks in a similar manner to subsection 3, then solve the problem in the rank space as described in the previous subsection.

▶ **Theorem 8.** *There exists a $(\mathcal{B}(n, m) + nH_d(S) + O(n) + o(\mathcal{B}(n, m) + n \lg \sigma))$-bit data structure that supports one-dimensional color range reporting queries in $O(k + 1)$ time.*

Note that $n = o(n \lg \sigma)$ as long as σ is not a constant. If σ is a constant, we solve the problem using a different approach. We store a separate succinct range emptiness data structure [13] for every subset of points with a given color. To answer a query $[a, b]$, for each color c we query the range emptiness data structure associated with c to check if a point with color c occurs in the range $[a, b]$, if so we report c. The query runtime is a constant since the number of colors is constant and range emptiness queries take constant time. Hence, we obtain the following theorem.

▶ **Theorem 9.** *There exists an $(\mathcal{B}(n, m) + nH_d(S) + o(\mathcal{B}(n, m) + n \lg \sigma))$-space data structure that supports one-dimensional color range reporting queries in $O(k + 1)$ time.*

5 Dynamic Color Reporting in Rank Space

Finally, we describe a succinct data structure that uses $nH_d(S) + o(n \lg \sigma)$ bits of space and answers color reporting queries in optimal $O(k + 1)$ time when the input points are in the rank space, while supporting the following update operation in $O(\lg n)$ time: given an index i and a color c, set the color of the i^{th} element to c.

▶ **Theorem 10.** *There exists an $(nH_d(S) + o(n \lg \sigma) + O(n))$-bit data structure that supports one-dimensional color range reporting queries in $O(k + 1)$ time and updates in $O(\lg n)$ time when points are in the rank space.*

Proof. Let the input sequence be $S = s_1, \ldots, s_n$, and $\mathcal{T}$ be the complete balanced binary tree where every leaf of $\mathcal{T}$ corresponds to an element of S and every internal node has two children. For any node $u \in \mathcal{T}$, $S(u)$ denotes the set of all elements stored in the leaf descendants of u. For $i \in \{1, \ldots, n\}$ denote by $l_i(r_i)$ the height of the highest ancestor u of the node corresponding to i such that i is the leftmost(rightmost) element in $S(u)$ with color s_i.

We store S in a dynamic data structure using $nH_d(S) + o(n \lg \sigma)$ bits that supports access in $O(1)$ time and Update, Rank, and Select in $O(\lg n/ \lg \lg n)$ time [14]. We divide S into blocks of $\lg n$ elements each, then we subdivide each block to subblocks of size $\lg \lg n$ elements. For each subblock b_{ij} $(0 \le i < n/ \lg n$ and $0 \le j < \lg n/ \lg \lg n)$ in block b_i we store:

- The maximum value m^l_{ij} of the sequence $l_{i \lg n + j \lg \lg n}, \ldots, l_{i \lg n + (j+1) \lg \lg n}$ and a succinct range maximum data structure [9] T^l_{ij} to answer range maximum queries on it.
- The maximum value m^r_{ij} of the sequence $r_{i \lg n + j \lg \lg n}, \ldots, r_{i \lg n + (j+1) \lg \lg n}$ and a succinct range maximum data structure [9] T^r_{ij} to answer range maximum queries on it.

The space used is $O(\lg \lg n)$ bits per subblock, which sums to $O(n)$ bits. For each block b_i we store:

- The sequence $m^l_{i0}, \ldots, m^l_{i \lg \lg n}$, its maximum value m^l_i, and a succinct range maximum data structure [9] T^l_i to answer range maximum queries on it.

- The sequence $m^r_{i0}, \ldots, m^r_{i\lg\lg n}$, its maximum value m^r_i, and a succinct range maximum data structure [9] T^r_i to answer range maximum queries on it.

The space used is $O(\lg n/\lg\lg n)$ bits per block, which sums to $O(n/\lg\lg n)$ bits. Finally, using Lemma 1 from a result by Nekrich et al. [22] we store using $O(n)$ bits two-dimensional point reporting structures T^l and T^r containing the set of points (i, m^l_i) and (i, m^r_i) where $1 \le i \le n/\lg n$. These structures support queries in $O(k+1)$ time and updates in $O(\lg^{\varepsilon} n)$ time.

Answering Queries: Given a query $[a,b]$, we find the lowest common ancestor u of a and b. Let $u_l(u_r)$ be the left(right) child of u, c be the rightmost child of u_l, and let h denote the height of u_l and u_r.

To get all distinct colors in $[a,c] = [a,b] \cap S(u_l)$, it is sufficient to report all colors s_i in that range with $r_i \ge h$. We maintain the invariant that each color is reported on its right most occurrence.

If $[a,c]$ was contained in a single subblock b_{ij}, we query T^r_{ij} for all the distinct colors as follows. We get the largest element r_d in $r_a, \ldots, r_c$, if s_d was previously reported we return, otherwise we report s_d and recurse on the interval $[d,c]$ followed by $[a,d]$. Note that it is important to recurse on $[d,c]$ before $[a,d]$ to maintain the invariant mentioned above, which guarantees that $r_d = \min(r_a, \ldots, r_c)$ will be smaller than h if the color s_d was previously reported.

Otherwise, if $[a,c]$ spans several subblocks but is contained in a single block b_i we proceed as follows. We first query the rightmost subblock partially spanned by $[a,c]$. Then, we query T^r_i to get all the subblocks b_{ij} spanned by $[a,c]$ satisfying the condition that $m^r_{ij} \ge h$ in order from right to left. We query each one of them in that order, then we query the leftmost subblock that is partially spanned by $[a,c]$.

Finally, if $[a,c]$ spans several blocks we first query the rightmost block partially spanned by $[a,c]$. Then, we query T^r to get all the blocks i spanned by $[a,c]$ satisfying the condition that $m^r_i \ge h$ in order from right to left. We query each one of them in that order, then we query the leftmost block that is partially spanned by $[a,c]$.

Similarly, to report all the distinct colors in $[c+1,b] = [a,b] \cap S(u_r)$ it is sufficient to report all colors s_i in that range with $l_i \ge h$. We do this in a similar way to the method used to query $[a,c]$, while maintaining the invariant that each color is reported on its left most occurrence.

Updating the Sequence: If the color of position i was updated from c to c' the following values could get modified: r_i, r_a where a is the first index before i with color c, r_b where b is the first index after i with color c', l_i, l_d where d is the first index after i with color c, and l_e where e is the first index before i with color c'.

We can find the value r_i of any index i in $O(\lg n/\lg\lg n)$ time by using Rank and Select queries to get the first index j before i with the same color as index i, then computing the lowest common ancestor of i and j. Similarly, to get the value l_i, we use Rank and Select queries to get the first index j after i with the same color as index i, then we compute the lowest common ancestor of i and j.

Since we don't store the values $r_1, \ldots, r_n$ and $l_1, \ldots, l_n$ explicitly, once one of them changes (say r_a where a is in subblock b_{ij}) we recompute all values r_j where $j \in b_{ij}$ and reconstruct T^r_{ij}. Recomputing all values r_j where $j \in b_{ij}$ takes $O(\lg\lg n \cdot \lg n/\lg\lg n) = O(\lg n)$ time and reconstructing T^r_{ij} takes $O(\lg\lg n)$ time. If m^r_{ij} changed, we rebuild T^r_i in $O(\lg n)$ time. Finally, if m_i changed we update its value in T^r in $O(\lg^{\varepsilon} n)$ time. Since only a constant number of values get updated, the runtime is $O(\lg n)$. ◀

If σ is a constant then the $O(n)$ additional bits stored by the data structure are no longer a lower order term, so we handle this case separately. We divide S into blocks of size $\lg n/2\lg\sigma$. We store a lookup table using $O(\sqrt{n}\lg^2 n)$ bits to answer color range queries over every possible block of this size. Also, we store the data structure from Theorem 10 on the sequence $S' = s'_1, \ldots, s'_{2\lg\sigma n/\lg n}$ with alphabet $\sigma' = 2^\sigma$, where s'_i denotes the subset of colors found on the i^{th} block of S. The total space used is $nH_d(S) + o(n\lg\sigma) + O(n/\lg n)$ bits. To answer a query $\mathcal{Q}$, we use the lookup table to get the colors in the (two) blocks which are not completely spanned by $\mathcal{Q}$, then we use the data structure from Theorem 10 to get the colors in the blocks that are fully spanned by $\mathcal{Q}$. Each color will be reported at most a constant number of times. The query time is $O(k+1) = O(1)$ and update time is $O(\lg n)$.

▶ **Theorem 11.** *There exists an $(nH_d(S) + o(n\lg\sigma))$-bit data structure that supports one-dimensional color range reporting queries in $O(k+1)$ time and updates in $O(\lg n)$ time when points are in the rank space.*

Acknowledgment. The first author gratefully acknowledges the discussions he had with Rahul Saladi on the problem.

References

1. Miklós Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.
2. Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.
3. Paul Beame and Faith E Fich. Optimal bounds for the predecessor problem. In *Proceedings of the 31st annual ACM symposium on Theory of computing*, pages 295–304. ACM, 1999.
4. Philip Bille, Gad M Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 373–389. Society for Industrial and Applied Mathematics, 2011.
5. Prosenjit Bose, Eric Y. Chen, Meng He, Anil Maheshwari, and Pat Morin. Succinct geometric indexes supporting point location queries. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 635–644, 2009.
6. Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming*, pages 464–474. Springer, 1995.
7. Timothy M Chan and Bryan T Wilkinson. Adaptive and approximate orthogonal range counting. *ACM Transactions on Algorithms (TALG)*, 12(4):45, 2016.
8. Arash Farzan, J. Ian Munro, and Rajeev Raman. Succinct indices for range queries with applications to orthogonal range maxima. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming, Part I*, pages 327–338, 2012.
9. Johannes Fischer. Optimal succinctness for range minimum queries. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics*, pages 158–169. Springer, 2010.
10. Michael L Fredman and Dan E Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.
11. Travis Gagie and Juha Kärkkäinen. Counting colours in compressed strings. In *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching, CPM 2011*, pages 197–207, 2011.

12 Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. Colored range queries and document retrieval. *Theoretical Computer Science*, 483:36–50, 2013.

13 Mayank Goswami, Allan Grønlund Jørgensen, Kasper Green Larsen, and Rasmus Pagh. Approximate range emptiness in constant time and optimal space. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 769–775, 2015.

14 Roberto Grossi, Rajeev Raman, Srinivasa Rao Satti, and Rossano Venturini. Dynamic compressed strings with random access. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, Part I*, pages 504–515, 2013.

15 Meng He. Succinct and implicit data structures for computational geometry. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 216–235, 2013.

16 J Ian Munro and S Srinivasa Rao. Succinct representation of data structures. In *Handbook of Data Structures and Applications*, chapter 37. Chapman and Hall/CRC, 2004.

17 Ravi Janardan and Mario Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3(01):39–69, 1993.

18 Christian W Mortensen. Generalized static orthogonal range searching in less space. Technical report, Citeseer, 2003.

19 J Ian Munro. Tables. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 37–42. Springer, 1996.

20 S Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 657–666. Society for Industrial and Applied Mathematics, 2002.

21 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9:1–9:21, 2014.

22 Yakov Nekrich and Jeffrey Scott Vitter. Optimal color range reporting in one dimension. In *Proceedings of the 21st European Symposium on Algorithms*, pages 743–754. Springer, 2013.

23 Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 232–240, 2006.

24 Saladi Rahul. Approximate range counting revisited. *arXiv preprint arXiv:1512.01713*, 2015.

25 Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–242, 2002.

26 Michiel Smid and Prosenjit Gupta. Further results on generalized intersection searching problems: counting, reporting, and dynamization. Technical report, 1992.

27 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proccedings of the 16th Annual Symposium on Foundations of Computer Science*, pages 75–84. IEEE, 1975.

Conflict-Free Coloring of Intersection Graphs*†

Sándor P. Fekete[1] and Phillip Keldenich[2]

1 **Algorithms Group, TU Braunschweig, Germany**
s.fekete@tu-bs.de

2 **Algorithms Group, TU Braunschweig, Germany**
p.keldenich@tu-bs.de

Abstract

A *conflict-free k-coloring* of a graph $G = (V, E)$ assigns one of k different colors to some of the vertices such that, for every vertex v, there is a color that is assigned to exactly one vertex among v and v's neighbors. Such colorings have applications in wireless networking, robotics, and geometry, and are well studied in graph theory. Here we study the conflict-free coloring of geometric intersection graphs. We demonstrate that the intersection graph of n geometric objects without fatness properties and size restrictions may have conflict-free chromatic number in $\Omega(\log n/\log\log n)$ and in $\Omega(\sqrt{\log n})$ for disks or squares of different sizes; it is known for general graphs that the worst case is in $\Theta(\log^2 n)$. For unit-disk intersection graphs, we prove that it is NP-complete to decide the existence of a conflict-free coloring with one color; we also show that six colors always suffice, using an algorithm that colors unit disk graphs of restricted height with two colors. We conjecture that four colors are sufficient, which we prove for unit squares instead of unit disks. For interval graphs, we establish a tight worst-case bound of two.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases conflict-free coloring, intersection graphs, unit disk graphs, complexity, worst-case bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.31

1 Introduction

Coloring the vertices of a graph is one of the fundamental problems in graph theory, both scientifically and historically. The notion of proper graph coloring can be generalized to hypergraphs in several ways. One natural generalization is *conflict-free coloring*, which asks to color the vertices of a hypergraph such that every hyperedge has at least one uniquely colored vertex. This problem has applications in wireless communication, where "colors" correspond to different frequencies.

The notion of conflict-free coloring can be brought back to simple graphs, e.g., by considering as hyperedges the neighborhoods of the vertices of G. The resulting problem arises in certain variants of frequency assignment problems if one is not interested in achieving signal coverage for all points in a region, but only at certain points of interest. For an illustration, consider a scenario in which one has a given set of nodes in the plane and wants to establish a communication network between them. Moreover, assume that constructing nodes at new locations is either very expensive or forbidden, and one can only "upgrade" any existing node to a wireless base station.

* This work was partially supported by the DFG Research Unit "Controlling Concurrent Change", funding number FOR 1800, project FE407/17-2, "Conflict Resolution and Optimization".

† A full version of the paper is available at [10], https://arxiv.org/abs/1709.03876.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 31; pp. 31:1–31:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conflict-free coloring also plays a role in robot navigation, where different beacons are used for providing direction. To this end, it is vital that in any given location, a robot is adjacent to a beacon with a frequency that is unique among the ones that can be received.

Both in the frequency assignment setting and in the robot navigation setting, one typically wants to avoid placing unnecessary base stations or beacons. Abstractly speaking, this corresponds to leaving some vertices uncolored, yielding the following formalization of *conflict-free coloring* of graphs. For any vertex $v \in V$ of a simple graph $G = (V, E)$, the neighborhood $N[v]$ consists of all vertices adjacent to v and v itself. A *conflict-free k-coloring* of G assigns one of k colors to a (possibly proper) subset $S \subseteq V$ of vertices such that every vertex $v \in V$ has a uniquely colored neighbor. The *conflict-free chromatic number* $\chi_{CF}(G)$ of G is the smallest k for which a conflict-free coloring exists. Depending on the situation it may also be more natural to consider *open neighborhood conflict-free coloring*, where each vertex v must have a uniquely colored neighbor in its *open neighborhood* $N(v)$ not including v.

Conflict-free coloring has received an increasing amount of attention. Because of the motivation arising from frequency assignment, it is natural to investigate the conflict-free coloring of intersection graphs, in particular, of simple shapes such as disks or squares. In addition, previous work has considered either general graphs and hypergraphs (e.g., see [21]) or other geometric scenarios in the presence of obstacles (e.g., see [14]); we give a more detailed overview further down. This adds to the relevance of conflict-free coloring of intersection graphs, which lie in the intersection of general graphs and geometry.

There is a spectrum of different scientific challenges when studying conflict-free coloring. What are worst-case bounds on the necessary number of colors? When is it NP-hard to determine the existence of a conflict-free k-coloring? We address these questions for the case of intersection graphs.

Our contribution. We present the following results.

- We demonstrate that n geometric objects without fatness properties and size restrictions may induce intersection graphs with conflict-free chromatic number in $\Omega(\log n / \log\log n)$.
- We prove that non-unit square and disk graphs may require $\Omega(\sqrt{\log n})$ colors. Deciding conflict-free k-colorability is NP-hard for any k for these graph classes.
- It is NP-complete for unit-disk intersection graphs to decide the existence of a conflict-free coloring with one color. The same holds for intersection graphs of unit squares and other shapes.
- Six colors are always sufficient for conflict-free coloring of unit disks. This uses an algorithm that colors unit disk graphs contained in a strip of restricted height with two colors.
- Using a similar argument, we prove that four colors are always sufficient for conflict-free coloring of unit squares.
- As a corollary, we get a tight worst-case bound of two on the conflict-free chromatic number of interval graphs.

Related work. In the geometric context, motivated by frequency assignment problems, the study of conflict-free coloring of hypergraphs was initiated by Even et al. [8] and Smorodinsky [22]. For disk intersection hypergraphs, Even et al. [8] prove that $\mathcal{O}(\log n)$ colors suffice. For disk intersection hypergraphs with degree at most k, Alon and Smorodinsky [4] show that $\mathcal{O}(\log^3 k)$ colors are sufficient. If every edge of a disk intersection hypergraph must have k distinct unique colors, Horev et al. [15] prove that $\mathcal{O}(k \log n)$ suffice. Moreover, for unit disks, Lev-Tov and Peleg [18] present an $\mathcal{O}(1)$-approximation algorithm for the conflict-free chromatic number. Abam et al. [1] consider the problem of making a conflict-free coloring robust against removal of a certain number of vertices, and prove worst-case bounds for the number of colors required.

The dual problem in which one has to color a given set of points such that each region contains a uniquely colored point has also received some attention. Har-Peled and Smorodinsky [13] prove that for families of pseudo-disks, every set of points can be colored using $\mathcal{O}(\log n)$ colors. For rectangles, Ajwani et al. [3] show that $\mathcal{O}(n^{0.382})$ colors suffice, whereas Elbassioni and Mustafa [7] show that it is possible to add a sublinear number of points such that sublinearly many colors suffice. For coloring points on a line with respect to intervals, Cheilaris et al. [6] present a 2-approximation algorithm, and a $\left(5-\frac{2}{k}\right)$-approximation algorithm when every interval must contain k uniquely colored points.

Conflict-free coloring also arises in the context of the conflict-free variant of the chromatic Art Gallery Problem, which asks to guard a polygon using colored guards such that each point sees a uniquely colored guard. Fekete et al. [9] prove that computing the chromatic number is NP-hard in this context. On the positive side, Hoffman et al. [14] prove $\Theta(\log \log n)$ colors are sometimes necessary and always sufficient for the conflict-free chromatic art gallery problem under rectangular visibility in orthogonal polygons. For straight-line visibility, Bärtschi et al. [5] prove that $\mathcal{O}(\log n)$ colors are sufficient.

There also has been work regarding the scenario where the hypergraph is induced by the neighborhoods of vertices of a simple graph. Except for the need to color all vertices, this corresponds to the scenario considered in this work. This does not change the asymptotic number of colors required, since it suffices to insert one additional color to color all vertices that would otherwise remain uncolored. In this situation, Pach and Tardos [21] prove that the conflict-free chromatic number of an n-vertex graph is in $\mathcal{O}(\log^2 n)$. Glebov et al. [12] extend this result by proving that almost all $G(n, \omega(1/n))$-graphs have conflict-free chromatic number $\mathcal{O}(\log n)$. Moreover, they show that the upper bound of Pach and Tardos [21] is tight by giving a randomized construction for graphs having conflict-free chromatic number $\Theta(\log^2 n)$. In more recent work, Gargano and Rescigno [11] show that finding the conflict-free chromatic number for general graphs is NP-complete, and prove that the problem is FPT w.r.t. vertex cover or neighborhood diversity number. In our work with Abel et al. [2], we consider conflict-free coloring of general and planar graphs and proved a conflict-free variant of Hadwiger's conjecture, which implies that planar graphs have conflict-free chromatic number at most three. Most recently, Keller and Smorodinsky [17] consider conflict-free coloring on intersection graphs of geometric objects, in a scenario very similar to ours. Among other results, they prove that $\mathcal{O}(\log n)$ colors suffices to color a family $\mathcal{F}$ of pseudodisks in a conflict-free manner. With respect to *open* neighborhoods (also known as *pointed neighborhoods*), they prove that this is tight; for closed neighborhoods as studied in this paper, the tightness of this bound is not proven and remains open. They also consider the list coloring variant of the problem.

Conflict-free coloring is not the only type of coloring for which unit disk graphs have been found to require a bounded number of colors. In their recent work, McDiarmid et al. [19] consider clique coloring of unit disk graphs, in particular with regard to the asymptotic behavior of the clique chromatic number of random unit disk graphs. They also prove that every unit disk graph in the plane can be colored with nine colors, while three colors are sometimes necessary. Similar to the present paper, they prove this by cutting the plane into strips of height $\sqrt{3}$; for each of these strips it is then proven that three colors suffice.

For more details on related works, refer to the full version [10].

2 Preliminaries

In the following, $G = (V, E)$ denotes a graph on $n := |V|$ vertices. For a vertex v, $N(v)$ denotes its *open neighborhood* and $N[v] = N(v) \cup \{v\}$ denotes its *closed neighborhood.* A *conflict-free k-coloring* of a graph $G = (V, E)$ is a coloring $\chi : V' \to \{1, \dots, k\}$ of a subset $V' \subseteq V$ of the vertices of G, such that each vertex v has at least one *conflict-free neighbor* $u \in N[v]$, i.e., a neighbor u whose color $\chi(u)$ occurs only once in $N[v]$. The *conflict-free chromatic number* $\chi_{CF}(G)$ is the minimum number of colors required for a conflict-free coloring of G.

A graph G is a *disk graph* iff G is the intersection graph of disks in the plane. G is a *unit disk graph* iff G is the intersection graph of disks with fixed radius $r = 1$ in the plane, and a *unit square graph* iff G is the intersection graph of axis-aligned squares with side length 1 in the plane. A unit disk (square) graph is of *height h* iff G can be modeled by the intersection of unit disks (squares) with center points in $(-\infty, \infty) \times [0, h]$. In the following, when dealing with intersection graphs, we assume that we are given a geometric model. In the case of unit disk and unit square graphs, we identify the vertices of the graph with the center points of the corresponding geometric objects in this model.

3 General Objects

Intersection graphs of geometric objects can generally contain cliques of arbitrary size, so their chromatic number may be unbounded. However, cliques do not require a large number of colors in a conflict-free coloring, so it is not immediately clear whether the intersection graphs for a family of geometric objects have bounded conflict-free chromatic number.

If the intersecting objects can be scaled down arbitrarily, i.e., if every representable graph can be represented using arbitrarily small area, we can make use of the following lemma to prove lower bounds on the number of colors required.

▶ **Lemma 1.** *Let G_k be a graph with $\chi_{CF}(G_k) \geq k$, and let G be a graph containing two disjoint copies J_k^1 and J_k^2 of G_k. Let $v_1, \dots, v_l$ be vertices of G, not contained in J_k^1 or J_k^2, and let each vertex v_i be adjacent to every vertex of J_k^1 and J_k^2. Moreover, let these vertices be the only neighbors of J_k^1 and J_k^2. Then in every conflict-free k-coloring of G, one of the vertices $v_1, \dots, v_l$ has a color that appears only once in $v_1, \dots, v_l$.*

Proof. Assume there was a conflict-free k-coloring χ of G such that none of the vertices $v_1, \dots, v_l$ has a unique color. Therefore, each vertex in J_k^1 has a conflict-free neighbor in J_k^1, and restricting χ to $V(J_k^1)$ yields a conflict-free k-coloring of J_k^1. As $\chi_{CF}(G_k) \geq k$, each color is used on $V(J_k^1)$ at least once. The same holds for J_k^2. Therefore, each vertex $v_1, \dots, v_l$ has at least two occurrences of each color in its neighborhood; this contradicts the fact that χ is a conflict-free coloring of G. ◀

For general objects like freely scalable ellipses or rectangles, it is possible to model a complete graph K_n of arbitrary size n, such that the following conditions hold: (1) For every object v, there is some non-empty area of v not intersecting any other objects. (2) For every pair of objects v, w, there is a non-empty area common to these objects not intersecting any other objects. This can be seen by choosing n intersecting lines such that no three lines intersect in a common point. These lines can then be approximated using sufficiently thin objects to achieve the desired configuration.

In this case, the conflict-free chromatic number is unbounded, because we can inductively build a family G_n of intersection graphs with $\chi_C(G_n) = n$ as follows. Starting with

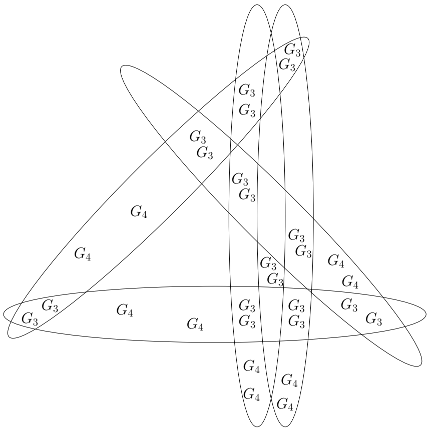


Figure 1 The graph G_5, shown as an intersection graph of ellipses, requires 5 colors.

$G_1 = (\{v\}, \emptyset)$ and $G_2 = C_4$ (a four-vertex cycle), we construct G_n by starting with a K_n modeled according to conditions (1) and (2). For every object v, we place two scaled-down non-intersecting copies of G_{n-1} into an area covered only by v; Figure 1 depicts the construction of G_5 for ellipses. According to Lemma 1, these gadgets enforce that every vertex of the underlying K_n is colored. For every pair of objects v, w, we place two scaled-down non-intersecting copies of G_{n-2} into an area covered only by v and w. Using an argument similar to that used in the proof of Lemma 1, these gadgets enforce that v and w have to receive different colors. Thus the resulting graph requires n colors.

The number of vertices used by this construction satisfies the recurrence

$$|G_1| = 1, \quad |G_2| = 4, \quad |G_n| = n + 2n|G_{n-1}| + n(n-1)|G_{n-2}|.$$

To estimate the growth of $|G_n|$, let $\bar{G}_n = |G_n|$ for $n \leq 2$ and $\bar{G}_n = 3n\bar{G}_{n-1} + n(n-1)\bar{G}_{n-2}$; clearly, $\bar{G}_n \geq |G_n|$ for all n. The recurrence $\bar{G}_n$ has the closed-form solution

$$\bar{G}_n = \frac{n!}{13 \cdot 2^{n+1}} \cdot \left((5\sqrt{13}-13)(3+\sqrt{13})^n - (13+5\sqrt{13})(3-\sqrt{13})^n\right) = \mathcal{O}\left(n! \left(\frac{3+\sqrt{13}}{2}\right)^n\right),$$

implying that the number of colors required in geometric intersection graphs on n vertices may be $\Omega(\frac{\log n}{\log \log n})$.

We summarize.

▶ **Theorem 2.** *The intersection graph of n convex objects in the plane may have conflict-free chromatic number in $\Omega(\log n / \log \log n)$.*

The best upper bound on the number of colors required in this scenario that we are aware of is $\mathcal{O}(\log^2 n)$, which holds for general graphs and is due to Pach and Tardos [21].

4 Different-Sized Squares and Disks

Due to their fatness, squares and disks do not allow us to construct an arbitrarily big clique K_n such that condition (2) of Section 3 holds. However, we can still prove that there is no constant bound on their conflict-free chromatic number. The proof is based on Lemma 1, which enables us to reduce the conflict-free coloring problem on intersection hypergraphs to our problem.

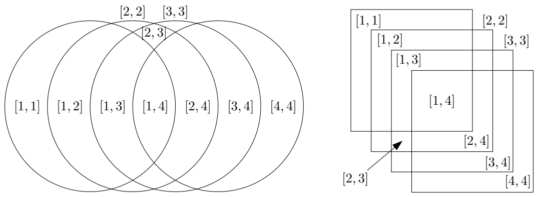


Figure 2 A chain of length 4 using either disks or squares, requiring 3 colors in every conflict-free hypergraph coloring. Adding two copies of D_2 to every interval yields a disk intersection graph that requires 3 colors in any conflict-free coloring.

▶ **Theorem 3.** *The conflict-free chromatic number of disk intersection graphs and square intersection graphs can be* $\Omega(\sqrt{\log n})$.

Proof. We begin our proof by inductively constructing, for any number of colors k, a disk intersection graph D_k with conflict-free chromatic number $\chi_{CF}(D_k) = k$ and $\mathcal{O}(2^{2k^2})$ vertices. The first level of the construction is D_1, consisting of an isolated vertex. The remainder of the construction is based on a lower-bound example due to Even et al. [8], requiring $\Omega(\log n)$ colors when each point in the union of all disks must lie in a uniquely colored disk. This lower-bound example consists of *chain disks* $1, \dots, 2^{k-1}$ on a horizontal line segment, placed such that all disks overlap in the center. For each interval $[i, j]$, $1 \le i \le j \le 2^{k-1}$, there is one region with non-zero area in which exactly the disks from this interval overlap. This situation is depicted in Figure 2.

To construct D_k, for each such interval $[i, j]$, we choose one such region and place two scaled-down disjoint copies of D_{k-1} in it. We prove that D_k requires k colors by induction on k. That D_1 requires one color is clear. Given that D_{k-1} requires $k-1$ colors for some $k \ge 2$, we can prove that D_k requires k colors as follows. Assume there was a conflict-free $(k-1)$-coloring χ of D_k. Due to Lemma 1, we know that, for every interval $[i, j]$, $1 \le i \le j \le 2^{k-1}$, at least one of the chain disks in $[i, j]$ has a unique color. We now prove using induction that any color assignment with this property has at least k colors. For a chain of length 2^0, one color is required for the interval $[1, 1]$. For a chain of length 2^l, we require one unique color for the interval $[1, 2^l]$. Let i be the chain disk colored using this color. At least one of the intervals $[1, i-1], [i+1, 2^l]$ has length at least 2^{l-1}. By induction, this interval requires l colors. These colors must all be distinct from the color used for i, therefore forcing us to use $l+1$ colors in total. This contradicts the fact that χ uses only $k-1$ colors; therefore, $\chi_{CF}(D_k) \ge k$. The number of vertices used by D_k satisfies the recurrence

$$|D_1| = 1, \quad |D_k| = 2^{k-1} + \frac{2^{k-1}(2^{k-1}+1)}{2}|D_{k-1}| = 2^{k-1} + (2^{2k-3} + 2^{k-2})|D_{k-1}|,$$

which is in $\mathcal{O}(2^{2k^2})$. All our arguments can also be applied to squares instead of disks. ◀

In [8], Even et al. prove that $\Theta(\log n)$ colors are always sufficient and sometimes necessary to color a disk intersection hypergraph in a conflict-free manner. This implies that $\mathcal{O}(\log n)$ colors are sufficient in our case, leaving a gap of $\mathcal{O}(\sqrt{\log n})$.

▶ **Theorem 4.** *For any fixed number of colors k, deciding whether a disk (or square) intersection graph is conflict-free k-colorable is NP-complete.*

Proof. The proof works inductively by reducing conflict-free $(k-1)$-colorability to conflict-free k-colorability; conflict-free 1-colorability is NP-hard by Theorem 5. For details, refer to the full version [10]. ◀

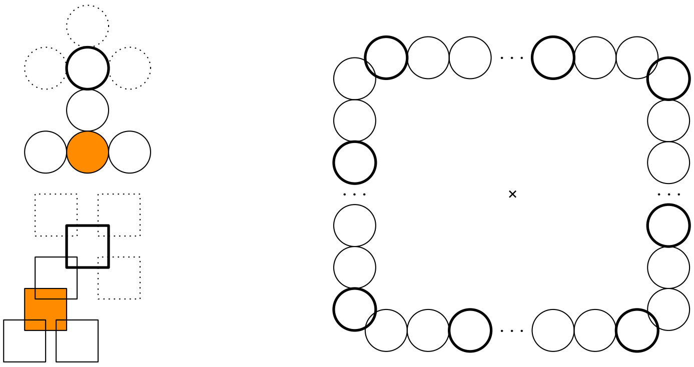


Figure 3 (Left) Clause gadget represented using unit disks and unit squares. The *clause vertices* that are attached to the remainder of $G(\phi)$ are drawn with bold outline. Dashed objects depict where the connections to the variables attach to the clause vertex. Orange vertices must be colored in any conflict-free 1-coloring; therefore, the clause vertex must remain uncolored. **(Right)** A variable gadget. *True vertices*, i.e., vertices that are colored if the variable is set to *true* are drawn with bold outline. In a conflict-free coloring of a variable gadget, every third vertex along the cycle must be colored. This implies that we must color either all *true* vertices or none of them.

5 Unit-Disk Graphs

The construction used in the previous section hinges on high aspect ratios of the intersecting shapes. In the setting of frequency assignment for radio transmitters, it is natural to only consider *fat* objects with bounded aspect ratio, such as unit disks and unit squares. As it turns out, their intersection graphs have conflict-free chromatic number bounded by a small constant; on the other hand, even deciding the existence of a conflict-free coloring with a single color is NP-complete.

5.1 Complexity: One Color

While it is trivial to decide whether a graph has a regular chromatic number of 1 and straightforward to check a chromatic number of 2, it is already NP-complete to decide whether a conflict-free coloring with a single color exists, even for unit-disk intersection graphs. This is a refinement of Theorem 4.1 in Abel et al. [2], which shows the same results for general planar graphs.

▶ **Theorem 5.** *It is NP-complete to decide whether a unit-disk intersection graph $G = (V, E)$ has a conflict-free coloring with one color.*

Proof. Due to space constraints, we only sketch the proof of NP-hardness; for a detailed proof, refer to the full version [10]. We prove NP-hardness by reduction from Positive Planar 1-in-3-SAT, see Mulzer and Rote [20]. Given a Boolean formula ϕ in 3-CNF with only positive literals and planar clause-variable incidence graph, we construct a unit disk intersection graph $G(\phi)$ that has a conflict-free 1-coloring iff ϕ is 1-in-3-satisfiable. In $G(\phi)$, variables are represented using variable gadgets (sufficiently long cycles with length divisible by 3) and clauses are represented using clause gadgets; see Figure 3. The variable gadgets have *true* and *false* vertices; coloring the *true* vertices corresponds to setting the variable to true. The *clause vertex* of the clause gadget is connected to a *true* vertex of each variable occurring in the clause by a path of length divisible by 3; see Figure 4. This enforces that in any conflict-free 1-coloring of $G(\phi)$, each clause vertex is connected to exactly one

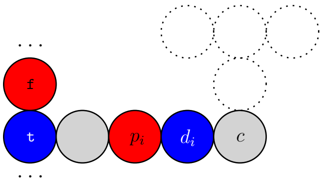


Figure 4 Path connecting a variable to a clause vertex c (with dashed clause gadget). The vertices marked t and f are *true* and *false* vertices of a variable. Blue vertices are colored if the variable is set to *true*, red vertices are colored if the variable is set to *false*. Gray vertices cannot be colored. For c, this is enforced directly by the clause gadget; for the other vertices along the path, it follows from the fact that the clause vertex cannot be colored.

variable gadget where the *true* vertices are colored. Therefore a conflict-free 1-coloring of $G(\phi)$ induces a 1-in-3-satisfying assignment and vice versa. ◀

5.2 A Worst-Case Upper Bound: Six Colors

On the positive side, we show that the conflict-free chromatic number of unit disk graphs is bounded by six. We do not believe this result to be tight. In particular, we conjecture that the number is bounded by four; in fact, we do not even know an example for which two colors are insufficient. One of the major obstacles towards obtaining tighter bounds is the fact that a simple graph-theoretic characterization of unit disk graphs is not available, as recognizing unit disk graphs is complete for the existential theory of the reals [16]. This makes it hard to find unit disk graphs with high conflict-free chromatic number, especially considering the size that such a graph would require: The smallest graph with conflict-free chromatic number three we know has 30 vertices, and by enumerating all graphs on 12 vertices one can show that at least 13 vertices are necessary, even without the restriction to unit disk graphs.

One approach to conflict-free coloring of unit disk graphs is by subdividing the plane into strips, coloring each strip independently. We conjecture the following.

▶ **Conjecture 6.** *Unit disk graphs of height* 2 *are conflict-free* 2*-colorable.*

If this conjecture holds, every unit disk graph is conflict-free 4-colorable. In this case, one can subdivide the plane into strips of height 2, and then color the subgraphs in all even strips using colors $\{1, 2\}$ and the subgraphs in odd strips using colors $\{3, 4\}$. Instead of Conjecture 6, we prove the following weaker result.

▶ **Theorem 7.** *Unit disk graphs G of height $\sqrt{3}$ are conflict-free 2-colorable.*

Proof. Given a realization of G consisting of unit disks with center points with y-coordinate in $[0, \sqrt{3}]$, we compute a conflict-free 2-coloring of G using the following simple greedy approach. In an order corresponding to the lexicographic order of the points in $\mathbb{R}^2$ (denoted by $\leq$), we build a set C of colored vertices to which we alternatingly assign colors 1 and 2. In each step, we add a new point to C until all points are covered, i.e., they are either colored or have a colored neighbor. In order to select the next colored point, we find the lexicographically maximal point c such that every point $c' < c$ is already colored or has a colored neighbor in $C \cup \{c\}$. We observe that this point c may already have a colored neighbor, but then there must be an uncovered point between c and previously colored point.

In this procedure, every point v is assigned a colored neighbor $w \in N[v]$. It remains to exclude the following three cases. (1) A colored point v is adjacent to another point w of the

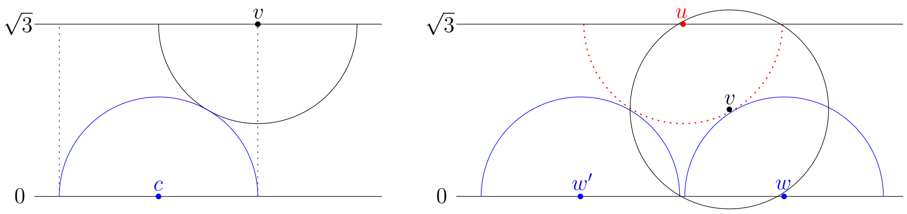


Figure 5 **(Left)** Every colored point c induces a vertical strip of width 2 (dashed lines); all points v within this strip are adjacent to c. **(Right)** The configuration in case (2); there must be a point u of color 2 adjacent to v.

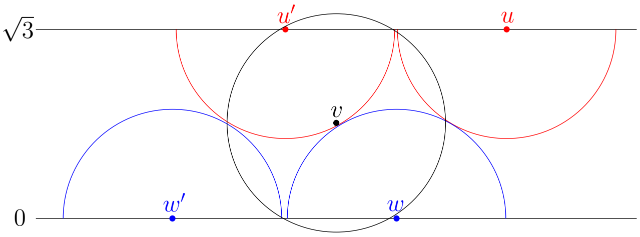


Figure 6 The configuration in case (3); the algorithm would have chosen v instead of u'.

same color, (2) an uncolored point is adjacent to two or more points of one color and none of the other color, (3) an uncolored point is adjacent to two or more points of both colors.

To this end, we use the following observation. Each colored point c induces a closed vertical *strip* of width 2 centered around c. As shown in Figure 5, every point v in this strip is adjacent to c. Thus, the horizontal distance between two colored points must be greater than 1. For case (1), assume there was a point v of color 1 adjacent to a point $w > v$ of color 1. This cannot occur, because between v and w, there must be a point x of color 2; therefore, the horizontal distance between v and w must be greater than 2, a contradiction.

Regarding case (2), assume there was an uncolored point v adjacent to two points $w' < v < w$ of color 1; see Figure 5. Between points w' and w, there must be a point u of color 2, and v must not be adjacent to u. There are two possible orderings: $w' < v < u < w$ and $w' < u < v < w$. W.l.o.g., let $u < v$; the other case is symmetric. In this situation, the x-coordinates of the points have to satisfy $x(u) < x(v) - 1$, $x(w') < x(u) - 1$, and thus $x(w') < x(v) - 2$ in contradiction to the assumption that v and w' are adjacent.

Regarding case (3), assume there was an uncolored point v adjacent to two points $w' < v < w$ of color 1 and two points $u' < v < u$ of color 2. W.l.o.g., assume $w' < u' < v < w < u$ as depicted in Figure 6; the case $u' < w'$ is symmetric. Because w' and v are adjacent, the vertical strip induced by v intersects the strip induced by w'. Thus, there cannot be a point y with $w' < y < v$ not adjacent to w' or v. This is a contradiction to the choice of u': The algorithm would have chosen v, or a larger point, instead of u'. ◀

The following corollary follows by subdividing the plane into strips of height $\sqrt{3}$.

▶ **Corollary 8.** *Unit disk graphs are conflict-free* 6*-colorable.*

Unfortunately, the proof of Theorem 7 does not appear to have a straightforward generalization to strips of larger height. Further reducing the height to find strips that are colorable with one color is also impossible, see Section 6.3.

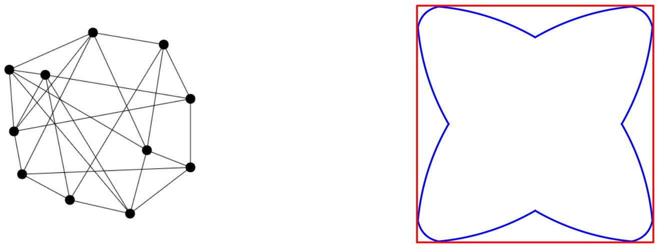


Figure 7 **(Left)** A vertex-minimal graph satisfying (1) and (2). **(Right)** In any unit disk graph G embeddable in a 2×2-square with domination number $\gamma(G) = 3$, no points lie in the depicted area.

5.3 Unit-Disk Graphs of Bounded Area

Proving Conjecture 6 is non-trivial, even when all center points lie in a 2×2-square. In this setting, a circle packing argument can be used to establish the sufficiency of three colors. If a unit disk graph with conflict-free chromatic number 3 can be embedded into a 2×2-square, the following are necessary. (1) Every minimum dominating set D has size 3, and every pair of dominating vertices must have a common neighbor not shared with the third dominating vertex. Thus, every minimum dominating set lies on a 6-cycle without chords connecting a vertex with the opposite vertex. (2) G has diameter 2; otherwise, one could assign the same color to two vertices at distance 3.

Using the domination number, one can further restrict the position of the points in the 2×2-square: There is an area in the center of the square, depicted in Figure 7, that cannot contain the center of any disk because this would yield a dominating set of size 2.

The smallest graph satisfying constraints (1) and (2) has 11 vertices and is depicted in Figure 7. It is not a unit disk graph and it is still conflict-free 2-colorable, but every coloring requires at least four colored vertices, proving that coloring a minimum dominating set can be insufficient. This implies that a simple algorithm like the one used in the proof of Theorem 7 will most likely be insufficient for strips of greater height. We are not aware of any unit disk graph satisfying these constraints.

6 Unit-Square and Interval Graphs

The constructions of the previous section can also be applied to the case of squares; for interval graphs, we get a tight worst-case bound.

6.1 Complexity: One Color

It is straightforward to see that the construction of Theorem 5 can be applied for unit square instead of unit disks.

▶ **Corollary 9.** *It is NP-complete to decide whether a unit square graph $G = (V, E)$ has a conflict-free coloring with one color.*

6.2 A Worst-Case Upper Bound: Four Colors

The proof of Theorem 7 can be applied to unit square graphs of height 2 instead of unit disk graphs of height $\sqrt{3}$; see Figure 8.

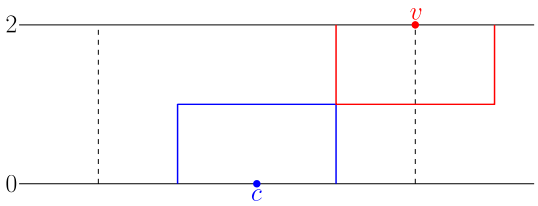


Figure 8 For unit square graphs of height 2, we have a similar situation to that depicted in Figure 5: Centered around each colored vertex c, there is a vertical strip of width 4 such that all vertices v with center points in this strip are adjacent to c. The remainder of the proof of Theorem 7 applies to unit square graphs analogously.

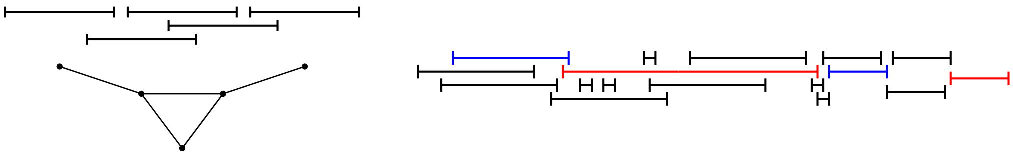


Figure 9 **(Left)** Realizing the Bull Graph as a unit interval graph. Conflict-free coloring requiring two colors: there is no dominating vertex, the only pair of vertices at distance 3 is no dominating set. **(Right)** A conflict-free 2-coloring of an interval graph as computed by the greedy coloring algorithm sketched above.

▶ **Corollary 10.** *Unit square graphs of height* 2 *are conflict-free* 2*-colorable. Unit square graphs are conflict-free* 4*-colorable.*

6.3 Interval Graphs: Two Colors

Unit interval graphs correspond to unit disk or unit square graphs with all centers lying on a line. Even then, two colors in a conflict-free coloring may be required; the Bull Graph is such an example, see Figure 9.

In this case, the bound of 2 is tight: By Theorem 7, unit interval graphs are conflict-free 2-colorable. By adapting the algorithm used in the proof to always choose the interval extending as far as possible to the right without leaving a previous interval uncovered, this can be extended to interval graphs with non-unit intervals. For an example of this procedure, refer to Figure 9.

7 Conclusion

There are various directions for future work. In addition to closing the worst-case gap for unit disks (and proving Conjecture 6), the worst-case conflict-free chromatic number for unit square graphs also remains open. Other questions include a tight bound for disk (or square) intersection graphs, and a necessary criterion for a family of geometric objects to have intersection graphs with unbounded conflict-free chromatic number.

References

1 Mohammad Ali Abam, Mark de Berg, and Sheung-Hung Poon. Fault-tolerant conflict-free colorings. In *Proc. 20th Canadian Conference on Computational Geometry (CCCG)*, pages 13–16, 2008.

2 Zachary Abel, Victor Alvarez, Erik D. Demaine, Sándor P. Fekete, Aman Gour, Adam Hesterberg, Phillip Keldenich, and Christian Scheffer. Three colors suffice: Conflict-free coloring of planar graphs. In *Proc. 28th Symp. Discrete Algorithms*, pages 1951–1963, 2017.

3 Deepak Ajwani, Khaled Elbassioni, Sathish Govindarajan, and Saurabh Ray. Conflict-free coloring for rectangle ranges using $O(n^{.382})$ colors. In *Proc. 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 181–187, 2007.

4 Noga Alon and Shakhar Smorodinsky. Conflict-free colorings of shallow discs. In *Proc. 22nd Symposium on Computational Geometry (SoCG)*, pages 41–43, 2006.

5 Andreas Bärtschi, Subir Kumar Ghosh, Matúš Mihalák, Thomas Tschager, and Peter Widmayer. Improved bounds for the conflict-free chromatic art gallery problem. In *Proc. 30th Symposium on Computational Geometry (SoCG)*, pages 144–153, 2014.

6 Panagiotis Cheilaris, Luisa Gargano, Adele A. Rescigno, and Shakhar Smorodinsky. Strong conflict-free coloring for intervals. *Algorithmica*, 70(4):732–749, 2014.

7 Khaled Elbassioni and Nabil H. Mustafa. Conflict-free colorings of rectangles ranges. In *23rd Proc. Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 254–263, 2006.

8 Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003.

9 Sándor P. Fekete, Stephan Friedrichs, Michael Hemmer, Joseph B. M. Mitchell, and Christiane Schmidt. On the chromatic art gallery problem. In *Proc. 26th Canadian Conference on Computational Geometry (CCCG)*, pages 1–6, paper 11, 2014.

10 Sándor P. Fekete and Phillip Keldenich. Conflict-free coloring of intersection graphs. *arXiv preprint arXiv:1709.03876*, 2017.

11 Luisa Gargano and Adele A. Rescigno. Complexity of conflict-free colorings of graphs. *Theoretical Computer Science*, 566:39–49, 2015.

12 Roman Glebov, Tibor Szabó, and Gábor Tardos. Conflict-free coloring of graphs. *Combinatorics, Probability and Computing*, 23:434–448, 2014.

13 Sariel Har-Peled and Shakhar Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry*, 34(1):47–70, 2005.

14 Frank Hoffmann, Klaus Kriegel, Subhash Suri, Kevin Verbeek, and Max Willert. Tight bounds for conflict-free chromatic guarding of orthogonal art galleries. In *Proc. 31st Symposium on Computational Geometry (SoCG)*, pages 421–435, 2015.

15 Elad Horev, Roi Krakovski, and Shakhar Smorodinsky. Conflict-free coloring made stronger. In *Proc. 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 105–117, 2010.

16 Ross J. Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discrete & Computational Geometry*, 47(3):548–568, 2012.

17 Chaya Keller and Shakhar Smorodinsky. Conflict-free coloring of intersection graphs of geometric objects. *arXiv preprint arXiv:1704.02018*, 2017.

18 Nissan Lev-Tov and David Peleg. Conflict-free coloring of unit disks. *Discrete Applied Mathematics*, 157(7):1521–1532, 2009.

19 Colin McDiarmid, Dieter Mitsche, and Pawel Pralat. Clique colourings of geometric graphs. *arXiv preprint arXiv:1701.02693*, 2017.

20 Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2):11, 2008.

21 János Pach and Gábor Tardos. Conflict-free colourings of graphs and hypergraphs. *Combinatorics, Probability and Computing*, 18(05):819–834, September 2009.

22 Shakhar Smorodinsky. *Combinatorial Problems in Computational Geometry.* PhD thesis, School of Computer Science, Tel-Aviv University, 2003.

On Using Toeplitz and Circulant Matrices for Johnson-Lindenstrauss Transforms*†

Casper Benjamin Freksen[1] and Kasper Green Larsen[2]

1 **Department of Computer Science, Aarhus University, Aarhus, Denmark**
cfreksen@cs.au.dk

2 **Department of Computer Science, Aarhus University, Aarhus, Denmark**
larsen@cs.au.dk

Abstract

The Johnson-Lindenstrauss lemma is one of the corner stone results in dimensionality reduction. It says that given N, for any set of N, vectors $X \subset \mathbb{R}^n$, there exists a mapping $f : X \to \mathbb{R}^m$ such that $f(X)$ preserves all pairwise distances between vectors in X to within $(1 \pm \varepsilon)$ if $m = \mathcal{O}(\varepsilon^{-2} \lg N)$. Much effort has gone into developing fast embedding algorithms, with the Fast Johnson-Lindenstrauss transform of Ailon and Chazelle being one of the most well-known techniques. The current fastest algorithm that yields the optimal $m = \mathcal{O}(\varepsilon^{-2} \lg N)$ dimensions has an embedding time of $\mathcal{O}(n \lg n + \varepsilon^{-2} \lg^3 N)$. An exciting approach towards improving this, due to Hinrichs and Vybíral, is to use a random $m \times n$ Toeplitz matrix for the embedding. Using Fast Fourier Transform, the embedding of a vector can then be computed in $\mathcal{O}(n \lg m)$ time. The big question is of course whether $m = \mathcal{O}(\varepsilon^{-2} \lg N)$ dimensions suffice for this technique. If so, this would end a decades long quest to obtain faster and faster Johnson-Lindenstrauss transforms. The current best analysis of the embedding of Hinrichs and Vybíral shows that $m = \mathcal{O}(\varepsilon^{-2} \lg^2 N)$ dimensions suffice. The main result of this paper, is a proof that this analysis unfortunately cannot be tightened any further, i.e., there exists a set of N vectors requiring $m = \Omega(\varepsilon^{-2} \lg^2 N)$ for the Toeplitz approach to work.

1998 ACM Subject Classification F.0 Theory of Computation

Keywords and phrases dimensionality reduction, Johnson-Lindenstrauss, Toeplitz matrices

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.32

1 Introduction

The performance of many geometric algorithms depends heavily on the dimension of the input data. A widely used technique to combat this “curse of dimensionality”, is to preprocess the input via *dimensionality reduction* while approximately preserving important geometric properties. Running the algorithm on the lower dimensional data then uses less resources (time, space, etc.) and an approximate result for the high dimensional data can be derived from the low dimensional result.

Dimensionality reduction approximately preserving pairwise Euclidean distances has found uses in a wide variety of applications, including: Nearest-neighbour search [2, 13], clustering [6, 8], linear programming [23], streaming algorithms [20], compressed sensing

[*] This research is supported by a Villum Young Investigator Grant, an AUFF Starting Grant and MADALGO, Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84.

[†] The full version of this paper is [12], https://arxiv.org/abs/1706.10110.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 32; pp. 32:1–32:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

[7, 11], numerical linear algebra [26], graph sparsification [21], and differential privacy [5]. See more applications in [22, 15]. The most fundamental result in this regime is the Johnson-Lindenstrauss (JL) lemma [16], which says the following:

▶ **Theorem 1** (Johnson-Lindenstrauss lemma). *Let $X \subset \mathbb{R}^n$ be a set of N vectors, then for any $0 < \varepsilon < 1/2$, there exists a map $f : X \to \mathbb{R}^m$ for some $m = \mathcal{O}(\varepsilon^{-2} \lg N)$ such that*

$$\forall x, y \in X, (1 - \varepsilon)\|x - y\|_2^2 \leq \|f(x) - f(y)\|_2^2 \leq (1 + \varepsilon)\|x - y\|_2^2.$$

This result dates back to 1984 and says that to preserve pairwise Euclidean distances amongst a set of N points/vectors in $\mathbb{R}^n$ to within a factor $(1 \pm \varepsilon)$, it suffices to use just $m = \mathcal{O}(\varepsilon^{-2} \lg N)$ dimensions. The bound on m was very recently proven optimal [19].

The standard technique for constructing a map with the properties of Theorem 1 is the following: Let A be an $m \times n$ matrix with entries independently sampled as either $\mathcal{N}(0, 1)$ random variables (as in [10]) or Rademacher (uniform among $\{-1, +1\}$) random variables (as in [1]). Once such entries have been drawn, let $f : \mathbb{R}^n \to \mathbb{R}^m$ be defined as:

$$f(x) = \frac{1}{\sqrt{m}} Ax.$$

To prove that the map f satisfies the guarantees in Theorem 1, it is first shown that for any vector x, the probability that $\|f(x)\|_2^2$ is not within $(1 \pm \varepsilon)\|x\|_2^2$ is less than $1/N^2$. This probability is called the error probability and denoted δ. Using linearity of f and a union bound over all pairs $x, y \in X$, the probability that all pairwise distances (i.e. the norm of the vector $x - y$) are preserved can be shown to be at least $1/2$.

1.1 Time Complexity

Examining the classic Johnson-Lindenstrauss reduction above, we see that to embed a vector, we need to multiply with a dense matrix and the embedding time becomes $\mathcal{O}(nm)$ (or equivalently $\mathcal{O}(n\varepsilon^{-2} \lg N)$). This may be prohibitively large for many applications (recall one prime usage of dimensionality reduction is to speed up algorithms), and much research has been devoted to obtaining faster embedding time.

Fast Johnson-Lindenstrauss Transform

Ailon and Chazelle [2] were the first to address the question of faster Johnson-Lindenstrauss transforms. In their seminal paper, they introduced the so-called Fast Johnson-Lindenstrauss transform for speeding up dimensionality reduction. The basic idea in their paper is to first “precondition” the input data by multiplying with a diagonal matrix with random signs, followed by multiplying with a Hadamard matrix. This has the effect of “spreading” out the mass of the input vectors, allowing for the dense matrix A above to be replaced with a sparse matrix. Since we can multiply with a Hadamard matrix using Fast Fourier Transform, this gives an embedding time of $\mathcal{O}(n \lg n + \varepsilon^{-2} \lg^3 N)$ for embedding into the optimal $m = \mathcal{O}(\varepsilon^{-2} \lg N)$ dimensions. For $m = \varepsilon^{-2} \lg N \leq n^{1/2-\gamma}$ for any constant $\gamma > 0$, the embedding complexity was improved even further down to $\mathcal{O}(n \lg m)$ in [3].

Another approach to achieve the $\mathcal{O}(n \lg m)$ embedding time, but without the restriction on $\varepsilon^{-2} \lg N \leq n^{1/2-\gamma}$, is to sacrifice the target dimension. This was done in [4] and later improved in [18], where the embedding complexity was $\mathcal{O}(n \lg m)$ at the cost of an increased target dimension $m = \mathcal{O}(\varepsilon^{-2} \lg N \lg^4 n)$.

Sparse Vectors

Another approach to improve the performance of JL transforms, is to assume the input data is sparse, i.e. has few non-zero coordinates. Designing an algorithm based on the work in [25], Dasgupta et al. [9] achieved an embedding complexity of $\mathcal{O}(\|x\|_0\varepsilon^{-1}\lg^2(mN)\lg N)$, where $\|x\|_0 = |\{i \mid x_i \neq 0\}|$. This was later improved to $\mathcal{O}(\|x\|_0\varepsilon^{-1}\lg N)$ in [17].

Toeplitz Matrices

Finally, another very exciting approach is to use Toeplitz matrices or partial circulant matrices for the embedding. We first introduce the terminology.

An $m \times n$ Toeplitz matrix is an $m \times n$ matrix, where every entry on a diagonal has the same value:

$$\begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_{n-1} \\ t_{-1} & t_0 & t_1 & \cdots & t_{n-2} \\ t_{-2} & t_{-1} & t_0 & \cdots & t_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{-(m-1)} & t_{-(m-2)} & t_{-(m-3)} & \cdots & t_{n-m} \end{pmatrix}$$

A partial circulant matrix is a special kind of Toeplitz matrix, where every row, except the first, is the previous row rotated once:

$$\begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_{n-1} \\ t_{n-1} & t_0 & t_1 & \cdots & t_{n-2} \\ t_{n-2} & t_{n-1} & t_0 & \cdots & t_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n-(m-1)} & t_{n-(m-2)} & t_{n-(m-3)} & \cdots & t_{n-m} \end{pmatrix}$$

Hinrichs and Vybíral [14] proposed the following algorithm for generating a JL embedding based on a Toeplitz matrix[1]: Let $t_{-(m-1)}, t_{-(m-2)}, \ldots, t_{n-1}$ and $d_1, \ldots, d_n$ be i.i.d. Rademacher[2] random variables, and T be a Toeplitz matrix defined from $t_{-(m-1)}, t_{-(m-2)}, \ldots, t_{n-1}$ such that entry (i, j) takes values t_{j-i} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Let D be an $n \times n$ diagonal matrix with the random variable d_i giving the i'th diagonal entry. Define the map f as

$$f(x) = \frac{1}{\sqrt{m}} TDx.$$

Multiplying with a Toeplitz matrix corresponds to computing a convolution and can be done using Fast Fourier Transform. By appropriately blocking the input coordinates, the complexity of embedding a vector x is just $\mathcal{O}(n \lg m)$ for any target dimension m. The big question is of course, how low can the target dimension m be, while preserving the distances between vectors up to a factor of $1 \pm \varepsilon$?

In the original paper [14], the authors proved that setting the target dimension to $m = \mathcal{O}(\varepsilon^{-2}\lg^3(1/\delta))$, the norm of any vector would be preserved to within $(1 \pm \varepsilon)$ with probability at least $1-\delta$. Setting $\delta = 1/N^2$, a union bound over all pairwise difference vectors (as in the classic construction) shows that dimension $m = \mathcal{O}(\varepsilon^{-2}\lg^3 N)$ suffices. Later, the

[1] [14] uses a partial circulant matrix but notes that a Toeplitz matrix could be used as well.
[2] Note that in [14, 24] these variables are erroneously referred to as Bernoulli variables.

Table 1 Comparison of the performances of various Johnson-Lindenstrauss transform algorithms. N is the number of input vectors, n is the dimension of the input vectors, m is the dimension of the output vectors, ε is the distortion.

Type	Embedding time	Target dimension (m)	Ref.	Notes
Random projection	$\mathcal{O}(nm)$	$\mathcal{O}(\varepsilon^{-2}\lg N)$	[10]	
Sparse	$\mathcal{O}(\|x\|_0\varepsilon^{-1}\lg^2(mN)\lg N)$	$\mathcal{O}(\varepsilon^{-2}\lg N)$	[9]	
Sparse	$\mathcal{O}(\|x\|_0\varepsilon^{-1}\lg N)$	$\mathcal{O}(\varepsilon^{-2}\lg N)$	[17]	
FFT	$\mathcal{O}(n\lg n + m\lg^3 N)$	$\mathcal{O}(\varepsilon^{-2}\lg N)$	[2]	
FFT	$\mathcal{O}(n\lg m)$	$\mathcal{O}(\varepsilon^{-2}\lg N)$	[3]	$m \leq n^{1/2-\gamma}$
FFT	$\mathcal{O}(n\lg m)$	$\mathcal{O}(\varepsilon^{-2}\lg N\lg^4 n)$	[18]	
Toeplitz	$\mathcal{O}(n\lg m)$	$\mathcal{O}(\varepsilon^{-2}\lg^3 N)$	[14]	
Toeplitz	$\mathcal{O}(n\lg m)$	$\mathcal{O}(\varepsilon^{-2}\lg^2 N)$	[24]	

analysis was refined in [24], which lowered the target dimension to $m = \mathcal{O}(\varepsilon^{-2}\lg^2(1/\delta))$ for preserving norms to within $(1 \pm \varepsilon)$ with probability $1-\delta$. Again, setting $\delta = 1/N^2$, this gives $m = \mathcal{O}(\varepsilon^{-2}\lg^2 N)$ target dimension. Now if the analysis could be tightened even further to give the optimal $m = \mathcal{O}(\varepsilon^{-2}\lg N)$ dimensions, this would end the decades long quest for faster and faster embedding algorithms!

Our Contribution

Our main result unfortunately shows that the analysis of Vybíral [24] cannot be tightened to give an even lower target dimensionality. More specifically, we prove that the upper bound given in [24] is optimal:

▶ **Theorem 2.** *Let T and D be the $m \times n$ Toeplitz and $n \times n$ diagonal matrix in the embedding proposed by [14]. For all $0 < \varepsilon < C$, where C is a universal constant, and any desired error probability $\delta > 0$, if the following holds for every unit vector $x \in \mathbb{R}^n$:*

$$\Pr\left[\left|\left\|\frac{1}{\sqrt{m}}TDx\right\|_2^2 - 1\right| < \varepsilon\right] > 1-\delta,$$

then it must be the case that $m = \Omega(\varepsilon^{-2}\lg^2(1/\delta))$.

While Theorem 2 already shows that one cannot tighten the analysis of Vybíral for preserving the norm of just one vector, Theorem 2 does leave open the possibility that one would not need to union bound over all N^2 pairs of difference vectors when trying to preserve all pairwise distances amongst a set of N vectors. It could still be the case that there somehow was a strong positive correlation between distances being preserved (though this seems extremely unlikely, and would be something not seen in any previous approach to JL). To complete the picture, we indeed show in the full version of this paper [12] that this is not the case, at least for N somewhat smaller than the dimension n:

▶ **Theorem 3.** *Let T and D be the $m \times n$ Toeplitz and $n \times n$ diagonal matrix in the embedding proposed by [14]. For all $0 < \varepsilon < C$, where C is a universal constant, if the following holds for every set of N vectors $X \subset \mathbb{R}^n$:*

$$\Pr\left[\forall x, y \in X : \left|\left\|\frac{1}{\sqrt{m}}TDx - \frac{1}{\sqrt{m}}TDy\right\|_2^2 - \|x-y\|_2^2\right| \leq \varepsilon\|x-y\|_2^2\right] = \Omega(1),$$

then it must be the case that either $m = \Omega(\varepsilon^{-2} \lg^2 N)$ *or* $m = \Omega(n/N)$*.*

We remark that our proofs also work if we replace T be a partial circulant matrix (which was also proposed in [14]). Furthermore, we expect that minor technical manipulations to our proof would also show the above theorems when the entries of T and D are $\mathcal{N}(0,1)$ distributed rather than Rademacher (this was also proposed in [14]).

2 Lower Bound for One Vector

Let T be $m \times n$ Toeplitz matrix defined from random variables $t_{-(m-1)}, t_{-(m-2)}, \dots, t_{n-1}$ such that entry (i,j) takes values t_{j-i} for $i = 1, \dots, m$ and $j = 1, \dots, n$. Let D be an $n \times n$ diagonal matrix with the random variable d_i giving the i'th diagonal entry. This section shows the following:

▶ **Theorem 4.** *Let* T *be* $m \times n$ *Toeplitz and* D $n \times n$ *diagonal. If* $t_{-(m-1)}, t_{-(m-2)}, \dots, t_{n-1}$ *and* $d_1, \dots, d_n$ *are independently distributed Rademacher random variables for* $i = -(m-1), \dots, n-1$ *and* $j = 1, \dots, n$*, then for all* $0 < \varepsilon < C$*, where* C *is a universal constant, there exists a unit vector* $x \in \mathbb{R}^n$ *such that*

$$\Pr\left[\left|\left\|\frac{1}{\sqrt{m}}TDx\right\|_2^2 - 1\right| > \varepsilon\right] \geq 2^{-\mathcal{O}(\varepsilon\sqrt{m})}.$$

and furthermore, all but the first $O(\sqrt{m})$ *coordinates of* x *are* 0*.*

It follows from Theorem 4 that if we want to have probability at least $1 - \delta$ of preserving the norm of *any* unit vector x to within $(1 \pm \varepsilon)$, it must be the case that $\varepsilon\sqrt{m} = \Omega(\lg(1/\delta))$, i.e. $m = \Omega(\varepsilon^{-2} \lg^2(1/\delta))$. This is precisely the statement of Theorem 2. Thus we set out to prove Theorem 4.

To prove Theorem 4, we wish to invoke the Paley-Zygmund inequality, which states, that if X is a non-negative random variable with finite variance and $0 \leq \theta \leq 1$, then

$$\Pr[X > \theta\mathbb{E}[X]] \geq (1-\theta)^2 \frac{\mathbb{E}^2[X]}{\mathbb{E}[X^2]}.$$

We carefully choose a unit vector x, and define the random variable for Paley-Zygmund to be the k'th moment of the difference between the norm of x transformed and 1.

Proof. Let k be an even positive integer less than $m/4$ and define $s := 4k$. Note that $s \leq m$. Let x be an arbitrary n-dimensional unit vector such that the first s coordinates are in $\{-1/\sqrt{s}, +1/\sqrt{s}\}$, while the remaining $n - s$ coordinates are 0. Define the random variable parameterized by k

$$Z_k := \left(\left\|\frac{1}{\sqrt{m}}TDx\right\|_2^2 - 1\right)^k.$$

Since k is even, the random variable Z_k is non-negative.

We wish to lower bound $\mathbb{E}[Z_k]$ and upper bound $\mathbb{E}[Z_k^2]$ in order to invoke Paley-Zygmund. The bounds we prove are as follows:

▶ **Lemma 5.** *If* $k \leq \sqrt{m}$*, then the random variable* Z_k *satisfies:*

$$\mathbb{E}[Z_k] \geq m^{-k/2} k^k 2^{-\mathcal{O}(k)}$$

and

$$\mathbb{E}[Z_k^2] \leq m^{-k} k^{2k} 2^{\mathcal{O}(k)}.$$

Before proving Lemma 5 we show how to use it together with Paley-Zygmund to complete the proof of Theorem 4.

We start by invoking Paley-Zygmund and then rewriting the expectations according to Lemma 5,

$$\begin{aligned}
\Pr[Z_k > \mathbb{E}[Z_k]/2] \geq (1/4)\frac{\mathbb{E}^2[Z_k]}{\mathbb{E}[Z_k^2]} &\implies \\
\Pr[Z_k^{1/k} > (\mathbb{E}[Z_k]/2)^{1/k}] \geq (1/4)\frac{\mathbb{E}^2[Z_k]}{\mathbb{E}[Z_k^2]} &\implies \\
\Pr\left[\left|\left\|\frac{1}{\sqrt{m}}TDx\right\|_2^2 - 1\right| > \frac{k}{C_0\sqrt{m}}\right] \geq 2^{-\mathcal{O}(k)}.
\end{aligned}$$

Here C_0 is some constant greater than 0. For any $0 < \varepsilon < 1/C_0$, we can now set k such that $k/(C_0\sqrt{m}) = \varepsilon$, i.e. we choose $k = \varepsilon C_0\sqrt{m}$. This choice of k satisfies $k \leq \sqrt{m}$ as required by Lemma 5. We have thus shown that:

$$\Pr\left[\left|\left\|\frac{1}{\sqrt{m}}TDx\right\|_2^2 - 1\right| > \varepsilon\right] \geq 2^{-\mathcal{O}(\varepsilon\sqrt{m})}.$$

◀

▶ Remark. Theorem 4 can easily be extended to partial circulant matrices. The difference between partial circulant and Toeplitz matrices is the dependence between the values in the first m and last m columns. However, as only the first $s = 4k \leq 4\sqrt{m}$ entries in x are nonzero, the last m columns are ignored, and so partial circulant and Toeplitz matrices behave identically in our proof.

Proof of Lemma 5. Before we prove the two bounds in Lemma 5 individually, we rewrite $\mathbb{E}[Z_k]$, as this benefits both proofs.

$$\begin{aligned}
\mathbb{E}[Z_k] &= \mathbb{E}\left[\left(\left\|\frac{1}{\sqrt{m}}TDx\right\|_2^2 - 1\right)^k\right] \\
&= \mathbb{E}\left[\left(\left(\frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{n} t_{j-i}d_jx_j\right)^2\right) - 1\right)^k\right] \\
&= \mathbb{E}\left[\left(\left(\frac{1}{m}\sum_{i=1}^{m}\left(\left(\sum_{j=1}^{n} t_{j-i}^2d_j^2x_j^2\right) + \left(\sum_{j=1}^{n}\sum_{h\in\{1,\ldots,n\}\setminus\{j\}} t_{j-i}t_{h-i}d_jd_hx_jx_h\right)\right)\right) - 1\right)^k\right] \\
&= \mathbb{E}\left[\left(\frac{1}{m}\sum_{i=1}^{m}\left(\left(\sum_{j=1}^{n} t_{j-i}^2d_j^2x_j^2 - x_j^2\right) + \left(\sum_{j=1}^{n}\sum_{h\in\{1,\ldots,n\}\setminus\{j\}} t_{j-i}t_{h-i}d_jd_hx_jx_h\right)\right)\right)^k\right] \\
&= \mathbb{E}\left[\left(\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{h\in\{1,\ldots,n\}\setminus\{j\}} t_{j-i}t_{h-i}d_jd_hx_jx_h\right)^k\right] \\
&= \frac{1}{m^k}\sum_{S\in([m]\times[n]\times[n])^k\,|\,\forall(i,j,h)\in S:h\neq j} \mathbb{E}\left[\prod_{(i,j,h)\in S} t_{j-i}t_{h-i}d_jd_hx_jx_h\right]
\end{aligned}$$

Observe that for $j > s$ or $h > s$ the product becomes 0, as either x_j or x_h is 0. By removing all these terms, we simplify the sum to

$$\mathbb{E}[Z_k] = \frac{1}{m^k} \sum_{S \in ([m]\times[s]\times[s])^k \mid \forall (i,j,h)\in S: h\neq j} \mathbb{E}\left[\prod_{(i,j,h)\in S} t_{j-i}t_{h-i}d_j d_h x_j x_h\right]$$

Observe for an $S \in ([m] \times [s] \times [s])^k$, that the value $\mathbb{E}\left[\prod_{(i,j,h)\in S} t_{j-i}t_{h-i}d_j d_h x_j x_h\right]$ is 0 if one of the following two things are true:

- A d_j occurs an odd number of times in the product.
- A variable t_a occurs an odd number of times in the product.

To see this, note that by the independence of the random variables, we can write the expectation of the product, as a product of expectations where each term in the product has all the occurrences of the same random variable. Since the d_j's and t_a's are Rademachers, the expectation of any odd power of one of these random variables is 0. Thus if just a single random variable amongst the d_j's and t_a's occurs an odd number of times, we have $\mathbb{E}\left[\prod_{(i,j,h)\in S} t_{j-i}t_{h-i}d_j d_h x_j x_h\right] = 0$. Similarly, we observe that if every random variable occurs an even number of times, then the expectation of the product is exactly $1/s^k$ since each x_j also occurs an even number of times. If Γ_k denotes the number of tuples $S \in ([m]\times[s]\times[s])^k$ such that $\forall (i,j,h) \in S$ we have $h \neq j$ and furthermore:

- For all columns $a \in [s]$, $|\{(i,j,h) \in S \mid j = a \vee h = a\}| \mod 2 = 0$.
- For all diagonals $a \in \{-(m-1),\dots,s-1\}$, $|\{(i,j,h) \in S \mid j-i = a \vee h-i = a\}| \mod 2 = 0$.

Then we conclude

$$\mathbb{E}[Z_k] = \frac{\Gamma_k}{s^k m^k}. \tag{1}$$

Note that $Z_k^2 = Z_{2k}$. Therefore,

$$\mathbb{E}[Z_k^2] = \mathbb{E}[Z_{2k}] = \frac{\Gamma_{2k}}{s^{2k}m^{2k}}. \tag{2}$$

To complete the proof of Lemma 5 we need lower and upper bounds for Γ_k and Γ_{2k}. The bounds we prove are

▶ **Lemma 6.** *If $k \leq \sqrt{m}$, then Γ_k and Γ_{2k} satisfy:*

$$\Gamma_k = m^{k/2}s^k k^k 2^{-\mathcal{O}(k)}$$

and

$$\Gamma_{2k} = m^k s^{2k} k^{2k} 2^{\mathcal{O}(k)}.$$

The proof of the lower bound is in Section 2.1, while the proof of the upper bound is in the full version of this paper [12].

Substituting the bounds from Lemma 6 in (1) and (2) we get

$$\mathbb{E}[Z_k] = m^{-k/2}k^k 2^{-\mathcal{O}(k)}$$
$$\mathbb{E}[Z_k^2] = m^{-k}k^{2k}2^{-\mathcal{O}(k)},$$

which are the bounds we sought for Lemma 5. ◀

2.1 Lower Bounding Γ_k

We first recall that the definition of Γ_k is the number of tuples $S \in ([m] \times [s] \times [s])^k$ satisfying that $\forall (i,j,h) \in S$ we have $h \neq j$ and furthermore:

- For all columns $a \in [s]$, $|\{(i,j,h) \in S \mid j = a \vee h = a\}| \mod 2 = 0$.
- For all diagonals $a \in \{-(m-1), \ldots, s-1\}$, $|\{(i,j,h) \in S \mid j-i = a \vee h-i = a\}| \mod 2 = 0$.

We view a triple $(i,j,h) \in ([m] \times [s] \times [s])$ as two entries (i,j) and (i,h) in an $m \times s$ matrix. Furthermore, when we say that a triple *touches* a column or diagonal, a matrix entry of the triple lie on that column or diagonal, so (i,j,h) touches columns j and h and diagonals $j-i$ and $h-i$. Similarly, we say that a tuple $S \in ([m] \times [s] \times [s])^k$ touches a given column or diagonal l times, if l triples in S touches that column or diagonal.

We intent to prove a lower bound for Γ_k by constructing a big family of tuples $\mathcal{F} \subseteq ([m] \times [s] \times [s])^k$, where each tuple satisfies, that each column and diagonal touched by that tuple is touched exactly twice. As each column and diagonal is touched an even number of times, the number of tuples in the family is a lower bound for Γ_k.

Proof of $\Gamma_k = m^{k/2}s^k k^k 2^{-\mathcal{O}(k)}$. We describe how to construct a family of tuples $\mathcal{F} \subseteq ([m] \times [s] \times [s])^k$ satisfying that $\forall S \in \mathcal{F}, \forall (i,j,h) \in S$ we have $h \neq j$ and furthermore:

- For all columns $a \in [s]$, $|\{(i,j,h) \in S \mid j = a \vee h = a\}| \in \{0,2\}$.
- For all diagonals $a \in \{-(m-1), \ldots, s-1\}$, $|\{(i,j,h) \in S \mid j-i = a \vee h-i = a\}| \in \{0,2\}$.

From this and the definition of Γ_k it is clear that $|\mathcal{F}| \leq \Gamma_k$.

When constructing an $S \in \mathcal{F}$, we view S as consisting of two halves S_1 and S_2, such that S_1 touches exactly the same columns and diagonals as S_2 and both S_1 and S_2 touches each column and diagonal at most once. To capture this, we give the following definition, where $\mathbb{S}$ is meant to be the family of such halves S_1 and S_2.

▶ **Definition 7.** Let $\mathbb{S}$ be the set of all tuples $S \in ([m] \times [s] \times [s])^{k/2}$ such that

- $\forall (i,j,h) \in S, j \neq h$
- For all columns $a \in [s], |\{(i,j,h) \in S \mid j = a \vee h = a\}| \leq 1$
- For all diagonals $a \in \{-(m-1), \ldots, s-1\}, |\{(i,j,h) \in S \mid j-i = a \vee h-i = a\}| \leq 1$

Definition 7 mimics the definition of Γ_k, and the first item in Definition 7 ensures that the triples in a tuple in $\mathbb{S}$ are of the same form as in Γ_k. The final two items ensure that each column and diagonal, respectively, is touched at most once. This is exactly the properties we wanted of S_1 and S_2 individually.

We can now construct $\mathcal{F}$ as all pairs of (half) tuples $S_1, S_2 \in \mathbb{S}$, such that S_1 touches exactly the same columns and diagonals as S_2. To capture that S_1 and S_2 touch the same columns and diagonals, we introduce the notion of a signature. A signature of S_i is the set of columns and diagonals touched by S_i.

To have S_1 and S_2 touch exactly the same columns and diagonals, it is necessary and sufficient that they have the same signature.

We introduce the following notation: B denotes the number of signatures with at least one member, and by enumerating the signatures, we let b_i denote the number of (half) tuples in $\mathbb{S}$ with signature i.

We recall that a (half) tuple $S_1 \in \mathbb{S}$ touches each column and diagonal at most once, and if S_1 and S_2 share the same signature, they touch exactly the same columns and diagonals. Therefore, using $\circ$ to mean concatenation, $S = S_1 \circ S_2 \in \mathcal{F}$, as each column and diagonal touched is touched exactly twice. Therefore $|\mathcal{F}|$ is a lower bound for Γ_k. Note that for a given signature i, the number of choices of S_1 and S_2 with that signature is b_i^2. This gives the following inequality,

$$\Gamma_k \geq |\mathcal{F}| = \sum_{i=1}^{B} b_i^2.$$

We now apply the Cauchy-Schwarz inequality:

$$\sum_{i=1}^{B} b_i^2 \sum_{i=1}^{B} 1^2 \geq \Big(\sum_{i=1}^{B} b_i\Big)^2 \implies \sum_{i=1}^{B} b_i^2 \geq \frac{\big(\sum_{i=1}^{B} b_i\big)^2}{\sum_{i=1}^{B} 1^2} \implies \Gamma_k \geq \frac{|\mathbb{S}|^2}{B}. \tag{3}$$

To get a lower bound on $|\mathbb{S}|^2/B$ (and in turn Γ_k), we need a lower bound on $|\mathbb{S}|$ and an upper bound on B. These bounds are stated in the following lemmas

▶ **Lemma 8.** $|\mathbb{S}| = \Omega(m^{k/2}s^k 2^{-k})$.

▶ **Lemma 9.** $B = \mathcal{O}\left(\binom{m+s}{k/2} s^{k/2} \binom{s}{k}\right)$

Before proving any of these lemmas, we show that they together with (3) give the desired lower bound on Γ_k:

$$\Gamma_k = \frac{\Omega(m^{k/2}s^k 2^{-k})^2}{\mathcal{O}\left(\binom{m+s}{k/2} s^{k/2} \binom{s}{k}\right)} = \Omega\left(\frac{m^k s^{2k} 2^{-2k} (k/2)^{k/2} k^k}{(m+s)^{k/2} s^{k/2} s^k}\right). \tag{4}$$

Because $s = 4k$, we have $\frac{(k/2)^{k/2}}{s^{k/2}} = 2^{-\Theta(k)}$, and because $s \leq m$: $\frac{m^k}{(m+s)^{(k/2)}} = m^{k/2}2^{-\Theta(k)}$. With this we can simplify (4),

$$\Gamma_k = m^{k/2} s^k k^k 2^{-\mathcal{O}(k)}.$$

which is the lower bound we sought. ◀

Proof of Lemma 8. Recall that $\mathbb{S} \subseteq ([m] \times [s] \times [s])^{k/2}$ is the set of (half) tuples that touch each column and diagonal at most once, and, for each triple (i, j, h) in these (half) tuples, we have $j \neq h$.

We prove Lemma 8 by analysing how we can create a large number of distinct $S \in \mathbb{S}$ by choosing the triples in S iteratively.

For each triple, we choose a row and two distinct entries on this row. We choose the row among any of the m rows.

However, because $S \in \mathbb{S}$, when choosing entries on the row, we cannot choose entries that lie on columns or diagonals touched by previously chosen triples. Instead we choose the two entries among any of the other entries. Therefore, whenever we choose a triple, this triple prevents at most four row entries from being chosen for every subsequent triple, as the two diagonals and two columns touched by the chosen triple intersect with at most four entries on the rows of the subsequent triples. This leads to the following recurrence, describing a lower bound for the number of triples

$$F(r, c, t) = \begin{cases} r \cdot c \cdot (c-1) \cdot F(r, c-4, t-1) & \text{if } t > 0 \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

where r is the number of rows to choose from, c is the minimum number of choosable entries in any row, and t is the number of triples left to choose.

Inspecting (5), we can see that F can equivalently be defined as

$$F(r,c,t) = r^t \prod_{i=0}^{t-1} (c-4i)(c-1-4i). \tag{6}$$

If $t \leq \frac{c}{8}$ then the terms inside the product in (6) are greater than $\frac{c}{2}$, so we can bound F from below:

$$F(r,c,t) \geq r^t \big(\frac{c}{2}\big)^{2t} = r^t c^{2t} \frac{1}{4^t}.$$

We now insert the values for r, c and t to find a lower bound for $|\mathbb{S}|$, noting that $s = 4k$ ensures that $t \leq \frac{c}{8}$:

$$|\mathbb{S}| \geq F\big(m, s, \frac{k}{2}\big) \geq m^{k/2} s^k \frac{1}{4^{k/2}} \implies |\mathbb{S}| = \Omega(m^{k/2} s^k 2^{-k}).$$

◀

Proof of Lemma 9. Recall that for at triple $S \in \mathbb{S}$ we define the signature as the set of columns and diagonals touched by S. Furthermore, viewing a triple $(i,j,h) \in ([m] \times [s] \times [s])$ as the two entries (i,j) and (i,h) in an $m \times s$ matrix, we define the left endpoint as $(i, \min\{j,h\}$ and the right endpoint as $(i, \max\{j,h\})$.

The claim to prove is

$$B = \mathcal{O}\left(\binom{m+s}{k/2} s^{k/2} \binom{s}{k}\right).$$

This is proven by first showing an upper bound on the number of choices for the diagonals of left endpoints, then diagonals of right endpoints and finally for columns.

In an $m \times s$ matrix there are $m+s$ different diagonals and as the chosen diagonals have to be distinct, there are $\binom{m+s}{k/2}$ choices for the diagonals corresponding to left endpoints in a triple.

As the right endpoint of a triple has to be in the same row as the left endpoint, there are at most s choices for the diagonal corresponding to the right endpoint when the left endpoint has been chosen (which it has in our case). This gives a total of $s^{k/2}$ choices for diagonals corresponding to right endpoints.

Finally, there are s columns to choose from and the chosen columns have to be distinct, and so the total number of choices of columns is $\binom{s}{k}$.

The product of these number of choices gives the upper bound sought. ◀

References

1 Dimitris Achlioptas. Database-friendly random projections: Johnson–Lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, June 2003. doi:10.1016/S0022-0000(03)00025-4.

2 Nir Ailon and Bernard Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, May 2009. doi:10.1137/060673096.

3 Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual BCH codes. *Discrete & Computational Geometry*, 42(4):615–630, 2009. doi:10.1007/s00454-008-9110-x.

4 Nir Ailon and Edo Liberty. An almost optimal unrestricted fast Johnson–Lindenstrauss transform. *ACM Trans. Algorithms*, 9(3):21:1–21:12, June 2013. doi:10.1145/2483699.2483701.

5 Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The Johnson–Lindenstrauss transform itself preserves differential privacy. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS '12, pages 410–419, Washington, DC, USA, 2012. IEEE Computer Society. doi:10.1109/FOCS.2012.67.

6 C. Boutsidis, A. Zouzias, M. W. Mahoney, and P. Drineas. Randomized dimensionality reduction for k-means clustering. *IEEE Transactions on Information Theory*, 61(2):1045–1062, February 2015. doi:10.1109/TIT.2014.2375327.

7 E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, February 2006. doi:10.1109/TIT.2005.862083.

8 Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 163–172, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746569.

9 Anirban Dasgupta, Ravi Kumar, and Tamás Sarlos. A sparse Johnson–Lindenstrauss transform. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 341–350, New York, NY, USA, 2010. ACM. doi:10.1145/1806689.1806737.

10 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003. doi:10.1002/rsa.10073.

11 D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006. doi:10.1109/TIT.2006.871582.

12 Casper Benjamin Freksen and Kasper Green Larsen. On using Toeplitz and circulant matrices for Johnson–Lindenstrauss transforms. *ArXiv e-prints*, 2017. arXiv:1706.10110.

13 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(14):321–350, 2012. doi:10.4086/toc.2012.v008a014.

14 Aicke Hinrichs and Jan Vybíral. Johnson–Lindenstrauss lemma for circulant matrices. *Random Structures & Algorithms*, 39(3):391–398, 2011. doi:10.1002/rsa.20360.

15 Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 10–33, Washington, DC, USA, 2001. IEEE Computer Society. doi:10.1109/SFCS.2001.959878.

16 William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26:189–206, 1984. doi:10.1090/conm/026/737400.

17 Daniel M. Kane and Jelani Nelson. Sparser Johnson–Lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, January 2014. doi:10.1145/2559902.

18 Felix Krahmer and Rachel Ward. New and improved Johnson–Lindenstrauss embeddings via the Restricted Isometry Property. *SIAM J. Math. Anal.*, 43(3):1269–1281, 2011.

19 Kasper Green Larsen and Jelani Nelson. Optimality of the Johnson–Lindenstrauss lemma. In *Proceedings of the 2017 IEEE 58th Annual Symposium on Foundations of Computer Science*, FOCS '17, Washington, DC, USA, October 2017. IEEE Computer Society.

20 S. Muthukrishnan. *Data Streams: Algorithms and Applications*, volume 1(2) of *Foundations and Trends™ in Theoretical Computer Science.* now Publishers Inc., Hanover, MA, USA, January 2005. doi:10.1561/0400000002.

21 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, December 2011. doi:10.1137/080734029.

22 Santosh S. Vempala. *The random projection method*, volume 65 of *DIMACS - Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA, September 2004. doi:10.1007/978-1-4615-0013-1_16.

23 Ky Vu, Pierre-Louis Poirion, and Leo Liberti. Using the Johnson–Lindenstrauss lemma in linear and integer programming. *ArXiv e-prints*, July 2015. arXiv:1507.00990.

24 Jan Vybíral. A variant of the Johnson–Lindenstrauss lemma for circulant matrices. *Journal of Functional Analysis*, 260(4):1096–1105, 2011. doi:10.1016/j.jfa.2010.11.014.

25 Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009. ACM. doi:10.1145/1553374.1553516.

26 David P. Woodruff. *Sketching as a Tool for Numerical Linear Algebra*, volume 10(1–2) of *Foundations and Trends™ in Theoretical Computer Science*. now Publishers Inc., Hanover, MA, USA, 2014. doi:10.1561/0400000060.

Almost Linear Time Computation of Maximal Repetitions in Run Length Encoded Strings

Yuta Fujishige[1], Yuto Nakashima[2], Shunsuke Inenaga[3], Hideo Bannai[4], and Masayuki Takeda[5]

1 **Department of Informatics, Kyushu University, Japan**
`yuta.fujishige@inf.kyushu-u.ac.jp`

2 **Department of Informatics, Kyushu University, Japan**
`yuto.nakashima@inf.kyushu-u.ac.jp`

3 **Department of Informatics, Kyushu University, Japan**
`inenaga@inf.kyushu-u.ac.jp`

4 **Department of Informatics, Kyushu University, Japan**
`bannai@inf.kyushu-u.ac.jp`

5 **Department of Informatics, Kyushu University, Japan**
`takeda@inf.kyushu-u.ac.jp`

Abstract

We consider the problem of computing all maximal repetitions contained in a string that is given in run-length encoding. Given a run-length encoding of a string, we show that the maximum number of maximal repetitions contained in the string is at most $m+k-1$, where m is the size of the run-length encoding, and k is the number of run-length factors whose exponent is at least 2. We also show an algorithm for computing all maximal repetitions in $O(m\alpha(m))$ time and $O(m)$ space, where α denotes the inverse Ackermann function.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms

Keywords and phrases maximal repetitions, run length encoding

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.33

1 Introduction

Periodicity and repetitions in strings are one of the most important characteristic features in strings. They have been one of the first objects of study in the field of combinatorics on words [25] and have many theoretical, as well as practical applications, e.g., in bioinformatics [14].

Maximal repetitions are periodically maximal substrings of a string where the smallest period is at most half the length of the substring, i.e., there are at least two consecutive occurrences of the same substring. An $O(n \log n)$ time algorithm for computing all of the maximal repetitions contained in a string of length n, was shown by Main and Lorentz [24], which is optimal for general unordered alphabets, i.e., when only equality comparisons between the letters are allowed. Kolpakov and Kucherov [15] further showed that the number of maximal repetitions is actually $O(n)$, and gave a linear time algorithm for ordered constant size alphabets (and essentially for integer alphabets), to compute all of them. The algorithm was a modification of the algorithm by Main [23], which in turn relies on the Lempel-Ziv 77 (LZ77) factorization [27] of the string, which can be computed in linear time for ordered constant size or integer alphabets [8], but requires $\Omega(n \log \sigma)$ time for general ordered alphabets [16], where σ is the size of the alphabet. Recently, a new characterization

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 33; pp. 33:1–33:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of maximal repetitions using Lyndon words was proposed by Bannai et al. [2, 3], which lead to a very simple proof to what was known as the "runs" conjecture, i.e., that the number of maximal repetitions in a given string of length n is less than n [3]. The characterization also lead to a new linear time algorithm for computing maximal repetitions on ordered constant size and integer alphabets, which does not require the LZ77 factorization, but only on a linear number of *longest common extension queries*. Furthermore, based on this algorithm, the running time for computing all maximal repetitions for general ordered alphabets were subsequently improved to $O(n \log^{2/3} n)$ by Kosolobov [17], $O(n \log \log n)$ by Gawrychowski et al. [12], and $O(n\alpha(n))$ by Crochemore et al. [9], where α denotes the inverse Ackermann function.

In this paper, we consider the problem of computing all maximal repetitions contained in a string when given the *run-length encoding* (RLE) of the string, which is a well known compressed representation where each maximal substring of the same character is encoded as a pair consisting of the letter and the length of the substring. For example, the run-length encoding of the string `aaaabbbaaacc` is $(\mathtt{a}, 4)(\mathtt{b}, 3)(\mathtt{a}, 3)(\mathtt{c}, 2)$. The main contributions of the paper are:

1. an upper bound $m + k - 1$ on the number of maximal repetitions contained in a string, where m is the size of its run-length encoding and k is the number of run-length factors whose exponent is at least 2, and
2. an $O(m\alpha(m))$ time and $O(m)$ space algorithm to compute all maximal repetitions in a string.

Our algorithm is at least as efficient as the non-RLE algorithms for general ordered alphabets. Furthermore, when the input string is compressible via RLE, our algorithm can be faster and more space efficient compared to the non-RLE algorithms. Although our algorithm mimics those for non-RLE strings and is conceptually simple, its correctness is based on new non-trivial observations on the occurrence of specific Lyndon words in run-length encoded strings.

Efficient algorithm for string problems when the input is given in RLE has been considered in various contexts, for example, edit distance [6], various Longest Common Subsequence problems [20, 18], palindrome retrieval [7], computing Lempel Ziv factorization [26], etc. We shall repeat below a claim made in [18] concerning the significance of RLE-based solutions:

> "A common criticism against RLE based solutions is a claim that, although they are theoretically interesting, since most strings "in the real world" are not compressible by RLE, their applicability is limited and they are only useful in extreme artificial cases. We believe that this is not entirely true. There can be cases where RLE is a natural encoding of the data, for example, in music, a melody can be expressed as a string of pitches and their duration. Furthermore, in the data mining community, there exist popular preprocessing schemes for analyzing various types of time series data, which convert the time series to strings over a fairly small alphabet as an approximation of the original data, after which various analyses are conducted (e.g. SAX (Symbolic Aggregate approXimation) [19], *clipped* bit representation [1], etc.). These conversions are likely to produce strings which are compressible by RLE (and in fact, shown to be effective in [1]), indicating that RLE based solutions may have a wider range of application than commonly perceived."

2 Preliminaries

2.1 Strings

Let Σ denote the alphabet, i.e. the set of letters (or characters). An element of Σ^* is called a string. For any strings $x, y \in \Sigma^*$, xy represents their concatenation. If $w = xyz$ for any strings $w, x, y, z \in \Sigma^*$, x, y, z are respectively called a prefix, substring, suffix of w. A prefix x and a suffix z of w are respectively called a proper prefix, and proper suffix of w, if $x \neq w$ and $z \neq w$. A string x is called a border of w, if it is a proper suffix as well as a prefix of w. The length of a string w is denoted as $|w|$. The empty string is a string of length 0 and will be denoted by ε. For any $1 \leq i \leq j \leq |w|$, $w[i]$ denotes the ith letter of w, and $w[i..j] = w[i] \cdots w[j]$. For convenience, let $w[i..j] = \varepsilon$ when $i > j$. For any integer $k \geq 0$ and string $x \in \Sigma^*$, $x^0 = \varepsilon$, and $x^k = x^{k-1}x$.

We assume a general ordered alphabet, where a total order $\prec$ is defined on Σ, and the order between two letters in the alphabet can be computed in constant time. A total order $\prec$ on the alphabet induces a total order on the set of strings called the *lexicographic order*, which we also denote by $\prec$, i.e., for any $x, y \in \Sigma^*$, $x \prec y \iff x$ is a proper prefix of y, or, there exists $1 \leq i \leq \min\{|x|, |y|\}$ s.t. $x[1..i-1] = y[1..i-1]$ and $x[i] \prec y[i]$.

All previous linear time algorithms either assume a constant size ordered alphabet or an integer alphabet, i.e., $\Sigma = \{1, \ldots, n^c\}$ for some constant c. We will later see that this assumption does not help in our case.

2.2 Maximal Repetitions

For any string $w \in \Sigma^*$, an integer $1 \leq p < |w|$ is called a *period* of w if $w[i] = w[i+p]$ for all $1 \leq i \leq |w| - p$. A string whose smallest period is at most half its length is called a *repetition*. We are interested in occurrences of repetitions as a substring of a given string which are periodically maximal. Specifically, a triplet $r = (i, j, p)$ is called a *maximal repetition* of w, if and only if all the following hold:

1. p is the smallest period of $w[i..j]$ and $|w[i..j]| \geq 2p$ (repetition),
2. $i = 1$ or $w[i-1] \neq w[i-1+p]$ (left maximal), and
3. $j = |w|$ or $w[j+1] \neq w[j+1-p]$ (right maximal).

For any string w, we denote the set of maximal repetitions as $MReps(w)$. Although maximal repetitions are commonly referred to as "runs" in the literature, we use the term "maximal repetitions" so as not to confuse it with "run" in "run-length encoding".

For example, the string $w =$ `abaababaabaab` contains seven maximal repetitions, i.e., $MReps(w) = \{(3,4,1), (8,9,1), (11,12,1), (4,8,2), (1,6,3), (6,13,3), (1,11,5)\}$.

2.3 Run Length Encoding

Let $\mathcal{N}$ denote the set of positive integers. For any string $w \in \Sigma^*$, let $a_i \in \Sigma$ and $e_i \in \mathcal{N}$, for $1 \leq i \leq m$, be such that $w = a_1^{e_1} \cdots a_m^{e_m}$ and $a_i \neq a_{i+1}$ for all $1 \leq i < m$. The *run-length encoding* $RLE(w)$ of string w is a string over the alphabet $\Sigma \times \mathcal{N}$, and is defined as $RLE(w) = (a_1, e_1) \cdots (a_m, e_m)$. For any $1 \leq i \leq m$, each letter $RLE(w)[i] = (a_i, e_i)$ and its corresponding substring $a_i^{e_i}$ in w is called a run-length factor, and e_i is called its exponent.

The set of starting (resp. ending) positions of run-length factors of w is denoted by S_w (resp. E_w), i.e., $S_w = \{1 + \sum_{k=1}^{i-1} e_k : 1 \leq i \leq m\}$ and $E_w = \{\sum_{k=1}^{i} e_k : 1 \leq i \leq m\}$. We will also write $S_w[i] = 1 + \sum_{k=1}^{i-1} e_k$ and $E_w[i] = \sum_{k=1}^{i} e_k$ for any $1 \leq i \leq m$.

2.4 Lyndon Words

A string w is a *Lyndon word* [22] with respect to lexicographic order $\prec$, if and only if $w \prec w[i..|w|]$ for any $1 < i \leq |w|$, i.e., w is lexicographically smaller than any of its proper suffixes with respect to $\prec$. It is easy to see that a Lyndon word w cannot have a non-empty border, since a border would be a proper suffix of w that is lexicographically smaller than w, since it is also a prefix of w. An equivalent definition for a Lyndon word, is a word which is lexicographically smaller than any of its proper cyclic rotations.

For example, if $\mathtt{a} \prec \mathtt{b}$, then, the string $\mathtt{abaabb}$, $\mathtt{baa}$, $\mathtt{abab}$ are not Lyndon words with respect to $\prec$, while $\mathtt{aabab}$ is. The following is also well known.

▶ **Lemma 1** (Proposition 1.3 [10]). *For any Lyndon words u and v, uv is a Lyndon word iff $u \prec v$.*

2.5 Longest Common Extension

For any string w of length n, the *longest common extension query* is, given two positions $1 \leq i, j \leq n$, to answer

$$LCE_w(i,j) = \max\{k \mid w[i..i+k-1] = w[j..j+k-1], i+k-1, j+k-1 \leq n\}.$$

We also define the longest common extension in the reverse direction, i.e.,

$$LCE^R_w(i,j) = \max\{k \mid w[i-k+1..i] = w[j-k+1..j], i-k+1, j-k+1 \geq 1\}.$$

Note that if there is a way to compute $LCE_w(i,j)$ given w, there is also a way to compute $LCE^R_w(i,j)$ by considering the reversed string $w^R = w[n] \cdots w[1]$, since $LCE^R_w(i,j) = LCE_{w^R}(n-i+1, n-j+1)$.

3 The Maximum Number of Maximal Repetitions by RLE

The goal of this section is to prove the following Theorem.

▶ **Theorem 2.** *For any string w, let m be the size of its run-length encoding, and k the number of run-length factors of w whose exponent is at least 2. Then, $|MReps(w)| \leq m+k-1$.*

The proof basically follows the idea of [3] for normal strings, but it is extended to deal with RLE strings.

For any maximal repetition $r = (i,j,p)$ of string w and any lexicographic order $\prec$, there exists a substring of length p in $w[i..j]$ that is a Lyndon word with respect to $\prec$. This is because the set $\{w[i'..i'+p-1] \mid i+1 \leq i' \leq i+p\}$ contains all p cyclic rotations of $w[i+1..i+p]$ which are all distinct, since p is the smallest period of w, and a lexicographically smallest rotation will always exist. Any length p subinterval $[\ell, \ell+p-1]$ of a maximal repetition $r = (i,j,p)$ such that $w[\ell..\ell+p-1]$ is a Lyndon word with respect to $\prec$, is called an *L-root* of r with respect to $\prec$.

Theorem 2 is trivial when $|\Sigma| = 1$, so we can assume $|\Sigma| \geq 2$, and thus, we are able to consider two orderings denoted by $\prec_0$ and $\prec_1$, where $\prec_0 = \prec$ and for any $a, b \in \Sigma$, $a \prec_0 b \iff b \prec_1 a$. We also use $\prec_0$ and $\prec_1$ to denote the lexicographic orders on Σ^* induced by the respective total orders. As in [3], we choose, for each maximal repetition $r = (i,j,p)$, a specific lexicographic order denoted by $\prec_r \in \{\prec_0, \prec_1\}$ so that $w[j+1] \prec_r w[j+1-p]$. We note that either order can be chosen when $j = n$. The set B_r is defined as the beginning positions of L-roots of r with respect to this order, but excludes a position if it coincides with the beginning position of the maximal repetition, i.e., for any maximal repetition $r = (i, j, p)$,

$$B_r = \{\ell \mid [\ell..\ell + p - 1] \text{ is an L-root of } r \text{ w.r.t. } \prec_r \text{ , and } \ell \neq i\}.$$

Note that $|B_r| \geq 1$ since a maximal repetition always contains an L-root that does not start at its beginning. One of the crucial results of [3] was the following lemma, which implies that the number of maximal repetitions in a string w of length n is at most $n - 1$ since $\cup_{r \in MReps(w)} B_r \subseteq [2..n]$ and thus $|MReps(w)| \leq \sum_{r \in MReps(w)} |B_r| \leq n - 1$.

▶ **Lemma 3** (Lemma 8 of [3]). *For any distinct maximal repetitions r, r' of w, $B_r \cap B_{r'} = \emptyset$.*

The following lemma is an important new observation for L-roots of maximal repetitions with respect to their run-length encoding.

▶ **Lemma 4.** *For any maximal repetition $r = (i, j, p)$ of string w with $p \geq 2$, it holds that $B_r \subset S_w$, i.e., a position in B_r must be the beginning of an RLE-factor.*

Proof. Suppose to the contrary, that there is some $\ell \in B_r$ that is not at the beginning of an RLE-factor, i.e., $\ell \notin S_w$, and let $[\ell..\ell + p - 1]$ be the corresponding L-root of r. By the assumption, $w[\ell - 1] = w[\ell]$. Furthermore, by the definition of B_r, we have that $i < \ell$ and by the periodicity of r, $w[\ell - 1] = w[\ell + p - 1]$. However, this implies that $w[\ell..\ell + p - 1]$ has a border, contradicting that it is a Lyndon word. The lemma holds, since $1 \in S_w$ but $1 \notin B_r$. ◀

Of course, a run-length factor can be a maximal repetition of period 1, and can be stated as follows.

▶ **Lemma 5.** *For any string w, let $RLE(w) = (a_1, e_1) \cdots (a_m, e_m)$. For any $1 \leq i \leq m$, $(S_w[i], E_w[i], 1)$ is a maximal repetition of period 1 if and only if $e_i \geq 2$.*

We are now ready to prove Theorem 2.

Proof of Theorem 2. Recall that k is the number of run-length factors of w whose exponent is at least 2. Due to Lemma 5, the number of maximal repetitions with period 1 is equal to k. Note that for any maximal repetition r of period 1, any position $i \in B_r$ satisfies $i \notin S_w$. Let $MReps_{p \geq 2}(w)$ be the set of maximal repetitions such that the period is at least 2. From $|B_r| \geq 1$ and Lemmas 3 and 4,

$$|MReps_{p\geq 2}(w)| \leq \sum_{r \in MReps_{p\geq 2}(w)} |B_r| \leq m - 1 < m = |S_w|$$

holds. Thus, the total number of maximal repetitions is at most $m + k - 1$. ◀

If we consider the 2 cases w.r.t. m, we can get better bounds for each of 2 cases. Corollary 6 is the tight bound for smaller m. Corollary 7 is a improved bound for larger m.

▶ **Corollary 6.** *For any string w, let m be the size of its run-length encoding. If $m \leq 3$, $|MReps(w)| \leq m$.*

If $m = 3$, it is easy to see that $|MReps_{p\geq 2}(w)| = 0$. Obviously, $|MReps(w)| = k$ also holds, where k is the number of run-length factors of w whose exponent is at least 2.

▶ **Corollary 7.** *For any string w, let m be the size of its run-length encoding, and k the number of run-length factors of w whose exponent is at least 2. If $m \geq 4$, $|MReps(w)| \leq m + k - 3$.*

Proof. Since an L-root of $r \in \mathit{MReps}_{p \geq 2}(w)$ must contain at least two different characters, the beginning position $S_w[m]$ of the last run-length factor (a_m, e_m) cannot be in B_r for any $r \in \mathit{MReps}_{p \geq 2}(w)$. If $S_w[m-1] \in B_r$ for some $r \in \mathit{MReps}_{p \geq 2}(w)$, this implies that $[S_w[m-1], E_w[m]]$ is an L-root of r. Thus, $[S_w[m-2], E_w[m-1]]$ must also be an L-root with respect to the lexicographically reversed order, and $S_w[m-2] \notin B_r$. Since r ends at position $|w|$, we can choose either $S_w[m-1]$ or $S_w[m-2]$ as an element of B_r. This implies that either $S_w[m-1]$ or $S_w[m-2]$ is not in B_r for any $r \in \mathit{MReps}_{p \geq 2}(w)$. Thus $|\mathit{MReps}_{p \geq 2}(w)| \leq m - 3$ also holds. Since 1 and $S_w[m]$ are not contained in B_r, and since only one of $S_w[m-1]$ or $S_w[m-2]$ is contained in some B_r, we have that the maximum number of maximal repetitions in a string is at most $m + k - 3$. ◀

4 Computing All Maximal Repetitions on RLE strings

In this section, we propose an algorithm to compute all maximal repetitions on RLE strings. Our algorithm follows the new algorithm for normal strings proposed in [3], but is modified to handle RLE strings. We first review the algorithm for non-RLE strings.

4.1 Overview of Algorithm for Non-RLE Strings

The crucial observation made in [3] (which was also required for the proof of Lemma 3 in the previous section) is the following:

▶ **Lemma 8** (Lemma 7 of [3]). *For any maximal repetition $r = (i, j, p)$ of string w, let $[\ell, \ell + p - 1]$ be an L-root of r with respect to order $\prec_r$. Then, $w[\ell..\ell + p - 1]$ is the longest Lyndon word that is a prefix of $w[\ell..|w|]$.*

Based on this observation, the algorithm consists of two steps. Step 1: Compute all the longest Lyndon words with respect to $\prec_0$ and $\prec_1$ that start at each position of the string (the occurrences are candidates for L-roots). Step 2: For each such candidate $\lambda = w[i_\lambda..j_\lambda]$, compute $\ell_h = \mathit{LCE}_w(i_\lambda, j_\lambda + 1)$ and $\ell_g = \mathit{LCE}^R_w(i_\lambda - 1, j_\lambda)$ to see how long the period $p_\lambda = |w[i_\lambda..j_\lambda]| = j_\lambda - i_\lambda + 1$ continues to the left and to the right. We see that $[i_\lambda, j_\lambda]$ is indeed an L-root of the maximal repetition $r = (i_\lambda - \ell_g, j_\lambda + \ell_h, p_\lambda)$ if and only if $\ell_g + \ell_h \geq p_\lambda$.

Noticing that a Lyndon word can be created from any string by appending a unique smallest letter to the front of the string, we can use the *Lyndon tree* of a Lyndon word for Step 1. Given a Lyndon word w of length $n > 1$, (u, v) is the *standard factorization* [5, 21] of w, if $w = uv$ and v is the longest proper suffix of w that is a Lyndon word, or equivalently, the lexicographically smallest proper suffix of w. It is well known that for the standard factorization (u, v) of any Lyndon word w, the factors u and v are also Lyndon words (e.g.[4]). The *Lyndon tree* of w is the full binary tree defined by recursive standard factorization of w; w is the root of the Lyndon tree of w, its left child is the root of the Lyndon tree of u, and its right child is the root of the Lyndon tree of v. The longest Lyndon word that starts at each position can be obtained from the Lyndon tree, due to the following lemma.

▶ **Lemma 9** (Lemma 22 of [3]). *Let w be a Lyndon word with respect to $\prec$. $w[i..j]$ corresponds to a right node (or possibly the root) of the Lyndon tree with respect to $\prec$ if and only if $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that starts from i.*

The Lyndon tree of a normal string can be computed in $O(n\alpha(n))$ time over general ordered alphabet because of the following lemmas.

▶ **Lemma 10** (Observation 4 of [9]). *The Lyndon tree of a string of length n can be constructed by using $O(n)$ non-crossing LCE queries.*

▶ **Lemma 11** (Theorem 12 of [9]). *In a string of length n, a sequence of q non-crossing LCE queries can be answered in time $O(q + n\alpha(n))$, where α denotes the inverse Ackermann function.*

Here, a set of LCE queries is *non-crossing* if there are no two queries (i, j) and (i', j'), such that $i < i' < j < j'$ or $i' < i < j' < j$. After computing the Lyndon tree, $O(n)$ non-crossing LCE queries are computed again for each right node in Step 2 as described above. Thus the total time complexity for computing all maximal repetitions in non-RLE string is $O(n\alpha(n))$ time over general ordered alphabet.

We note that the LCE queries and thus all maximal repetitions can be computed in total $O(n)$ time for integer alphabets (using e.g. [11]).

4.2 Extending Lyndon structures for RLE

We now consider computing maximal repetitions on RLE strings. By Theorem 2, the number of maximal repetitions in an RLE string is $O(m)$, and from Lemmas 4 and 8, we can limit the candidate L-roots of maximal repetitions with period at least 2, to the longest Lyndon words that start at beginning positions of a run-length factor. We propose the *RLE-Lyndon tree* of a string which can be represented in $O(m)$ space and contains this information. In the RLE-Lyndon tree, we treat each run-length factor like a character. The idea of the extension comes from the following lemma.

▶ **Lemma 12.** *For any $1 \leq i < j \leq |w|$, if $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that is a prefix of $w[i..|w|]$, then $j \in E_w$, i.e., j is an end of a RLE-factor.*

Proof. Suppose to the contrary, that there is some $j \notin E_w$ such that $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that is a prefix of $w[i..|w|]$. Let $RLE(w)[k]$ be the run-length factor such that $S_w[k] \leq j < E_w[k]$. Since $w[i..j]$ is a Lyndon word of length at least 2 and $w[j] = a_k = w[j+1]$, $w[i..j] \prec w[j] = w[j+1]$ holds. By Lemma 1, $w[i..j+1]$ is also a Lyndon word. This contradicts that $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that is a prefix of $w[i..|w|]$. ◀

From Lemmas 4 and 12, we have that for any maximal repetition r, each L-root of r that has a starting position in B_r, starts at the beginning position of some run-length factor and ends at the ending position of some run-length factor. We note that RLE-Lyndon substring and RLE-Lyndon factorization which will be defined in this section were introduced in [13] in a different context.

▶ **Definition 13** (RLE-Lyndon substring). A string x is an *RLE-Lyndon substring* of w if x is a Lyndon word that is a concatenation of consecutive run-length factors of w, or x is a run-length factor.

▶ **Definition 14** (RLE-standard factorization). A pair of strings (u, v) is an *RLE-standard factorization* of w if $w = uv$ and v is the longest proper suffix of w that is an RLE-Lyndon substring.

▶ **Definition 15** (RLE-Lyndon tree). The *RLE-Lyndon tree* of a Lyndon word w, denoted $\mathit{LyndonTre}^2(w)$, is an ordered full binary tree defined recursively as follows:

- if $|RLE(w)| = 1$, then $\mathit{LyndonTre}^2(w)$ consists of a single node labeled by (a_1, e_1);
- if $|RLE(w)| \geq 2$, then the root of $\mathit{LyndonTre}^2(w)$, labeled by $RLE(w)$, has left child $\mathit{LyndonTre}^2(u)$ and right child $\mathit{LyndonTre}^2(v)$, where (u, v) is the RLE-standard factorization of w.

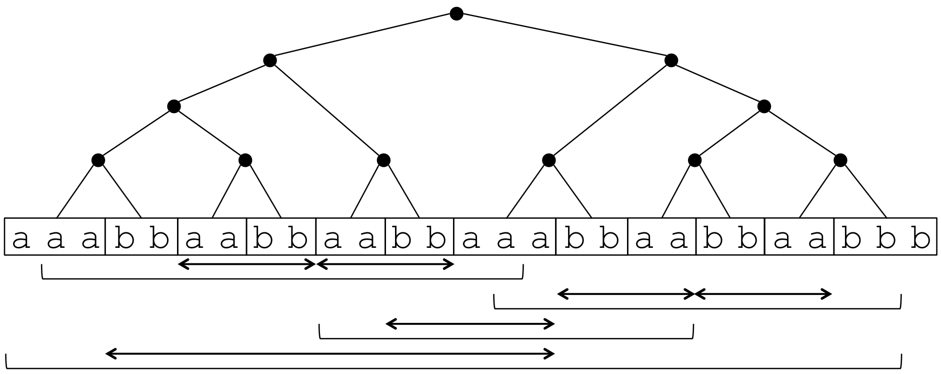


Figure 1 The RLE-Lyndon tree for the Lyndon word $\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^3$ with respect to order $\mathtt{a} \prec \mathtt{b}$. The double-headed arrows shows the L-roots that start at a position in B_r,for all 4 maximal repetitions with period at least 2.

Figure 1 shows the RLE-Lyndon tree of a string $\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^3$. Though the above structures are simply extended to RLE, it is interesting to note that these structures satisfy similar properties of the original structures. The most important property of the RLE-Lyndon tree in this paper is stated in Lemma 16, which is an analogous to Lemma 9. The lemma can be shown by similar arguments as in [3].

▶ **Lemma 16.** *Let w be a Lyndon word with respect to $\prec$. For any $i \in S_w$, $w[i..j]$ corresponds to a right node (or possibly the root) of $\mathit{LyndonTre}^2(w)$ with respect to $\prec$ if and only if $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that starts from i.*

From the above lemma, we can detect all maximal repetitions in $\mathit{MReps}_{p\geq 2}(w)$ if we have $\mathit{LyndonTre}^2(w)$ (maximal repetitions with period 1 correspond to run-length factor or leaves of $\mathit{LyndonTre}^2(w)$). In the example of Figure 1, for each maximal repetition r, the L-roots that start at a position in B_r are drawn by double-headed arrows. For example, the 2 L-roots $[S_w[3]..E_w[4]]$ and $[S_w[5]..E_w[6]]$ (corresponding to a Lyndon word $\mathtt{aabb}$) with respect to the same order $\prec$ as the Lyndon tree is represented by an internal node which is a right child. Also, it can be observed that each L-root begins at the starting position of a run-length factor and ends at the ending position of a run-length factor of w.

In Section 4.3, we show an algorithm to compute $\mathit{LyndonTre}^2(w)$. For convenience, we present the notion of *RLE-Lyndon factorizations* and show some properties of RLE-Lyndon factorizations.

▶ **Definition 17** (RLE-Lyndon factorization). A sequence $w_1, \ldots, w_s$ is the RLE-Lyndon factorization of w if each w_i is an RLE-Lyndon substring, $w_1 \succeq \ldots \succeq w_s$, and $w = w_1 \cdots w_s$.

The difference between the original Lyndon factorization [5] and the RLE-Lyndon factorization arises for Lyndon factors which are a single letter in the original Lyndon factorization. For a string $w = \mathtt{bbbabbaabbaa}$, the original Lyndon factorization of w is $\mathtt{b} \succeq \mathtt{b} \succeq \mathtt{b} \succeq \mathtt{abb} \succeq \mathtt{aabb} \succeq \mathtt{a} \succeq \mathtt{a}$, the RLE-Lyndon factorization of w is $\mathtt{b}^3 \succeq \mathtt{abb} \succeq \mathtt{aabb} \succeq \mathtt{a}^2$. Thus similar argument about the longest Lyndon word on Lyndon factorizations holds, as below.

▶ **Lemma 18.** *Let $w_1, \ldots, w_s$ be the RLE-Lyndon factorization of w. Then, w_1 is either $\mathit{RLE}(w)[1]$ or the longest Lyndon word that is a prefix of w.*

This implies that w_i is either $\mathit{RLE}(w_i \cdots w_s)[1]$ or the longest Lyndon word that is a prefix of $w_i \cdots w_s$.

4.3 Algorithms

Finally, we show how to compute $\mathit{LyndonTre}^2(w)$ in $O(m\alpha(m))$ time and $O(m)$ space. After we compute $\mathit{LyndonTre}^2(w)$, we can compute all maximal repetitions by using non-crossing LCE queries. Note that the $O(n)$ time and space solution for non-RLE strings over the integer alphabet cannot be applied to $\mathit{RLE}(w)$ to achieve an $O(m)$ time and space solution, since the alphabet for $\mathit{RLE}(w)$ cannot be assumed to be an integer alphabet in terms of its length m (m could be much smaller than n, while an exponent of a run-length factor could be as large as n). However, the solution to non-crossing LCE queries for non-RLE strings over a general ordered alphabet can be easily extended to LCE queries on an RLE string, since the algorithm is based only on character comparisons.

▶ **Corollary 19.** *For any RLE string $\mathit{RLE}(w)$ of size m, a sequence of q non-crossing LCE queries on $\mathit{RLE}(w)$ can be answered in time $O(q + m\alpha(m))$.*

We use the above corollary in order to decide the lexicographic order between RLE substrings in the construction of $\mathit{LyndonTre}^2(w)$, and to compute maximal repetitions.

▶ **Lemma 20.** *$\mathit{LyndonTre}^2(w)$ can be computed in $O(m\alpha(m))$ time and $O(m)$ space.*

Proof. Firstly, we show our algorithm. The algorithm constructs $\mathit{LyndonTre}^2(w)$ in bottom-up and from right to left. The main idea is that the right factor of RLE-standard factorization is the longest proper suffix which is an RLE-Lyndon substring. We will find such a suffix by concatenating two RLE-Lyndon substrings based on Lemma 1. Since each leaf corresponds to a single run-length factor (i.e., RLE-Lyndon substring), we know that the tree has m leaves. A stack is maintained so that at the beginning of k-th step, the stack contains the sequence of subtrees of $\mathit{LyndonTre}^2(w)$ such that the corresponding sequence of RLE-Lyndon substrings is the RLE-Lyndon factorization of the suffix $w[S_w[m-k+2]..|w|]$. In the k-th step, the algorithm pushes the leaf corresponding to $\mathit{RLE}(w)[m-k+1]$ on the stack. Let (f_b, f_e) (resp. (s_b, s_e)) be pair of positions in $\mathit{RLE}(w)$ such that the top (resp. second) subtree in the stack corresponds to the RLE-Lyndon substring $w[S_w[f_b]..E_w[f_e]]$ (resp. $w[S_w[s_b]..E_w[s_e]]$). Note that $E_w[f_e] + 1 = S_w[s_b]$ always holds. After pushing the new leaf, the algorithm does the following;

- If $w[S_w[f_b]..E_w[f_e]] \prec w[S_w[s_b]..E_w[s_e]]$, pop the two elements and push the subtree which is the concatenation of the two popped subtrees, and repeat the process.
- Otherwise, go to the next step.

We now prove that the above invariant condition of the stack holds before $k+1$-th step. We denote the RLE-Lyndon factorization of the suffix $w[S_w[m-k+2]..|w|]$ by $W_1, \ldots, W_j$. Because of the above operations, a factorization of the suffix $w[S_w[m-k+1]..|w|]$ can be represented by $W', W_i, \ldots, W_j$ for some $1 \leq i \leq j$ where $W' = \mathit{RLE}(w)[m-k+1]W_1 \cdots W_{i-1}$ (for convenience, $W_0 = \epsilon$). By the assumption, $W_i, \ldots, W_j$ is the RLE-Lyndon factorization of the suffix $W_i \cdots W_j$. By the algorithm and Lemma 1, W' is an RLE-Lyndon substring and $W' \succeq W_i$ holds. Thus $W', W_i, \ldots, W_j$ is the RLE-Lyndon factorization of the suffix $w[S_w[m-k+1]..|w|]$ since $W' \succeq W_i \succeq \ldots \succeq W_j$ holds. Since w is a Lyndon word, when all leaves are pushed on the stack and the number of elements in the stack is one, the algorithm stops and the RLE-Lyndon tree is completely constructed.

We can determine the lexicographic order by using LCE queries. More precisely, for each lexicographic comparison described above, we compute $\mathit{LCE}_{\mathit{RLE}(w)}(f_b, s_b) = k$. Then, $w[S_w[f_b]..E_w[f_e]] \prec w[S_w[s_b]..E_w[s_e]]$ if and only if $s_b + k - 1 < s_e$ and, either

1. $a_{f_b+k} \prec a_{s_b+k}$ or, $a_{f_b+k} = a_{s_b+k}$ and
2. $e_{f_b+k} < e_{s_b+k}$ and $a_{f_b+k+1} \prec a_{s_b+k}$ or
3. $e_{f_b+k} > e_{s_b+k}$ and $a_{s_b+k+1} \prec a_{f_b+k}$.

Thus, in the algorithm, we call $O(m)$ non-crossing LCE queries such that each query positions is the beginning position of some run-length factor and we can compute $LyndonTre^2(w)$ in $O(m\alpha(m))$ time.

To compute all maximal repetitions, we need to compute another $O(m)$ sets of LCE queries on w (or w^R) for each candidate L-root. The query positions are starting positions of run-length factors in w (or w^R). It is easy to see that this can also be achieved in $O(m\alpha(m))$ time by Corollary 19 since if $k = LCE_{RLE(w)}(i, j)$, then $LCE_w(S_w[i], S_w[j]) = E_w[i + k - 1] - S_w[i] + 1 + e$, where $e = \min\{e_{i+k}, e_{j+k}\}$ if $a_{i+k} = a_{j+k}$ and 0 otherwise. It is also clear that the algorithm requires $O(m)$ space. ◀

Therefore, the following theorem holds.

▶ **Theorem 21.** *Given a run-length encoding of a string w, all maximal repetitions in w can be computed in $O(m\alpha(m))$ time and $O(m)$ space.*

▶ **Corollary 22.** *For any string w, let $RLE(w) = (a_1, e_1) \cdots (a_m, e_m)$. If for all $1 \leq i \leq m$, $a_i \in \{1, \ldots, m^{c_1}\}$, and $e_i = O(m^{c_2})$ for some constants c_1 and c_2, then all maximal repetitions in w can be computed in $O(m)$ time and $O(m)$ space.*

Proof. Under the assumption, any set of $O(m)$ LCE queries on $RLE(w)$ can be answered in $O(m)$ total time using the methods for integer alphabets (i.e., $\Sigma = \{1, \ldots, m^c\}$ for some constant c), since $a_i^{e_i} = O(m^{c_1+c_2})$. ◀

References

1 Anthony Bagnall, Chotirat "Ann" Ratanamahatana, Eamonn Keogh, Stefano Lonardi, and Gareth Janacek. A bit level representation for time series data mining with shape based similarity. *Data Mining and Knowledge Discovery*, 13(1):11–40, 2006.

2 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. A new characterization of maximal repetitions by lyndon trees. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 562–571. SIAM, 2015. doi:10.1137/1.9781611973730.38.

3 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

4 Frédérique Bassino, Julien Clément, and Cyril Nicaud. The standard factorization of Lyndon words: an average point of view. *Discrete Mathematics*, 290(1):1–25, 2005.

5 K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus. iv. the quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958.

6 Kuan-Yu Chen and Kun-Mao Chao. A fully compressed algorithm for computing the edit distance of run-length encoded strings. *Algorithmica*, 65(2):354–370, 2013. doi:10.1007/s00453-011-9592-4.

7 Kuan-Yu Chen, Ping-Hui Hsu, and Kun-Mao Chao. Efficient retrieval of approximate palindromes in a run-length encoded string. *Theor. Comput. Sci.*, 432:28–37, 2012. doi:10.1016/j.tcs.2012.01.023.

8 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. doi:10.1016/j.ipl.2007.10.006.

9 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Ritu Kundu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Near-optimal computation of runs over general alphabet via non-crossing LCE queries. In *Proc. SPIRE 2016*, pages 22–34, 2016.

10 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4(4):363–381, 1983.

11 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006. doi:10.1007/11780441_5.

12 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster longest common extension queries in strings over general alphabets. In *Proc. CPM 2016*, pages 5:1–5:13, 2016.

13 Sukhpal Singh Ghuman, Emanuele Giaquinta, and Jorma Tarhio. Alternative algorithms for lyndon factorization. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2014, Prague, Czech Republic, September 1-3, 2014*, pages 169–178. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2014. URL: http://www.stringology.org/event/2014/p16.html.

14 Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps: Efficient and flexible detection of tandem repeats in DNA. *Nucleic acids research*, 31(13):3672–3678, July 2003.

15 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

16 Dmitry Kosolobov. Lempel-ziv factorization may be harder than computing all runs. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 582–593. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.582.

17 Dmitry Kosolobov. Computing runs on a general alphabet. *Inf. Process. Lett.*, 116(3):241–244, 2016.

18 Keita Kuboi, Yuta Fujishige, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster STR-IC-LCS computation via RLE. In *Combinatorial Pattern Matching, 28th Annual Symposium, CPM 2017, Warsaw, Poland, July 4-6, 2017, Proceedings*, 2017. in press. URL: http://arxiv.org/abs/1703.04954.

19 Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

20 Jia Jie Liu, Yue-Li Wang, and Yu-shan Chiu. Constrained longest common subsequences with run-length-encoded strings. *Comput. J.*, 58(5):1074–1084, 2015. doi:10.1093/comjnl/bxu012.

21 M. Lothaire. *Combinatorics on Words*. Addison-Wesley, 1983.

22 Roger C. Lyndon. On Burnside's problem. *Transactions of the American Mathematical Society*, 77(2):202–202, February 1954.

23 Michael G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989. doi:10.1016/0166-218X(89)90051-6.

24 Michael G. Main and Richard J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

25 Axel Thue. Über unendliche zeichenreihen. *Christiana Videnskabs Selskabs Skrifter, I. Math. naturv. Klasse, 7*, 1906.

26 Jun-ichi Yamamoto, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Faster compact on-line lempel-ziv factorization. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 675–686. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.675.

27 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–349, 1977.

Embedding Graphs into Embedded Graphs[*†]

Radoslav Fulek

IST, Klosterneuburg, Austria
radoslav.fulek@gmail.com

Abstract

A (possibly degenerate) drawing of a graph G in the plane is approximable by an embedding if it can be turned into an embedding by an arbitrarily small perturbation. We show that testing, whether a drawing of a planar graph G in the plane is approximable by an embedding, can be carried out in polynomial time, if a desired embedding of G belongs to a fixed isotopy class, i.e., the rotation system (or equivalently the faces) of the embedding of G and the choice of outer face are fixed. In other words, we show that c-planarity with embedded pipes is tractable for graphs with fixed embeddings.

To the best of our knowledge an analogous result was previously known essentially only when G is a cycle.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph embedding, C-planarity, Weakly simple polygons

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.34

1 Introduction

In the theory of graph visualization a drawing of a graph $G = (V, E)$ in the plane is usually assumed to be free of degeneracies, i.e., edge overlaps and edges passing through a vertex. However, in practice degenerate drawings often arise and need to be dealt with.

Recent papers [1, 7] address a certain aspect of this problem for simple polygons which can be thought of as straight-line (rectilinear) embeddings of graph cycles. Chang et al. [7] gave an $O(n^2 \log n)$-time algorithm to detect if a given polygon with n vertices can be turned into a simple (non self-intersecting) one by small perturbations of its vertices, or in other words if the polygon is **weakly simple**. We mention that there exists an earlier closely related definition of weakly simple polygons by Toussaint [6, 26], however, as pointed out in [7] this notion is not well-defined for general polygons with "spurs", see [7] for an overview of attempts at combinatorial definitions of a polygon not crossing itself.

An $O(n \log n)$ improvement on the running time of the algorithm by Chang et al. was announced very recently by Akitaya et al. [1]. The combinatorial formulation of this problem corresponds to the setting of **c-planarity with embedded pipes** introduced by Cortese et al. [10] well before the two aforementioned papers. Therein only an $O(n^3)$-time algorithm for the problem was given. Nevertheless, the algorithms in [1, 7] were built upon the ideas from [10]. Moreover, to the best of our knowledge the complexity status of the c-planarity with embedded pipes is essentially known only for cycles. Recently the problem was studied for general planar graphs by Angelini and Da Lozzo [3], but they gave only an FPT algorithm.

[*] The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no [291734]. The author gratefully acknowledges support from Austrian Science Fund (FWF): M2281-N35.

[†] Full version on arXiv https://arxiv.org/abs/1608.02087.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 34; pp. 34:1–34:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The introduction of this problem was motivated by a more general and well known problem of **c-planarity** by Feng et al. [13, 14], whose tractability status was open since 1995 even in much more restricted cases than the one that we consider. Biedl [4] gave a polynomial-time algorithm for c-planarity with two clusters. Beyond two clusters a polynomial time algorithm for c-planarity was obtained only in special cases, e.g., [9, 18, 19, 20, 21], and most recently in [5, 8, 15].

There is, however, another tightly related line of research on approximability or realizations of maps pioneered by Sieklucki [24], Minc [22] and M. Skopenkov [25] that is completely independent from the aforementioned developments, and that is also a major source of inspiration for our work. It can be easily seen that the result [25, Theorem 1.5] implies that c-planarity is tractable for flat instances with three clusters or cyclic clustered graphs [17, Section 6] with a fixed isotopy class of a desired embedding. An algorithm with a better running time was given by the author in [15].

The aim of the present work is to show that c-planarity with embedded pipes is tractable for planar graphs with a fixed isotopy class of embeddings, which extends results of [2, 3, 15]. Our work also implies the tractability of deciding whether a drawing is approximable by an embedding in a fixed isotopy class, which extends results of [1, 7]. This also answers in the affirmative a question posed in [7, Section 8.2] if the isotopy class of an embedding of G is fixed.

Roughly, we are to decide if in the given isotopy of G an embedding approximating a given (possibly degenerate) drawing of G class exists. The degenerate drawing of G is viewed as a plane graph H and the degeneracies (if any) are captured by a simplicial map between G and H. Let G and H be a pair of graphs such that H contains neither loops nor multiple edges, i.e., H is **simple**. A map $\gamma : V(G) \rightarrow V(H)$ is **simplicial** if for every edge $uv \in E(G)$ either $\gamma(u) = \gamma(v)$ or $\gamma(u)\gamma(v)$ is an edge of H. We partition $V(G)$ into **clusters** V_ν so that $\gamma(v) = \nu$ if and only if $v \in V_\nu$. If it leads to no confusion, we do not distinguish between a vertex or an edge and its representation in the drawing and we use the words "vertex" and "edge" in both contexts. We are in the position to state our problem formally.

We are given an ordered triple (G, H, γ), where G is a planar graph (possibly with loops and multiple edges) given by the isotopy class of an embedding of G in the plane, H is a plane simple graph[1], and $\gamma : V(G) \rightarrow V(H)$ is a simplicial map. We assume that the drawing given by H is piece-wise linear. The **feature size** of H is the minimum of the set consisting of the Euclidean non-zero distances between the endpoints of the line segments defining the drawing of H, and the Euclidean distances between the line segments defining the drawing of H. By treating a graph as a 1-dimensional topological space we extend the definition of γ linearly to the edges of G. We want to decide if the given isotopy class of G contains an embedding $\mathcal{E}$ such that $\|\mathcal{E}(x) - \gamma(x)\|_2 \ll \varepsilon$, for all $x \in G$, where $\varepsilon := \varepsilon(H) > 0$ is smaller than half of the feature size of H. Thus, by the choice of ε a desired embedding of G lies in a small neighborhood of H preserving the facial structure of the embedding of H.

However, to view the problem from a perspective that is more combinatorial, we put further restrictions on a desired embedding of G, which lead to the equivalent problem of **c-planarity with embedded pipes**, see Figure 1. To this end we need to introduce a couple of notions. Let $\text{dist}(\mathbf{p}, \mathbf{q})$ denote the Euclidean distance between $\mathbf{p}, \mathbf{q} \in \mathbb{R}^2$. Let $\text{dist}(\mathbf{p}, S) = \min_{\mathbf{q} \in S} \text{dist}(\mathbf{p}, \mathbf{q})$, where $S \subset \mathbb{R}^2$. Let $N_\varepsilon(S)$ for $S \subset \mathbb{R}^2$ denote the ε-neighborhood of S, i.e., $N_\varepsilon(S) = \{\mathbf{p} \in \mathbb{R}^2 | \text{ dist}(\mathbf{p}, S) \le \varepsilon\}$. Let $\varepsilon' > 0$ be a small value as described later. The

[1] In other words, a (planar) graph drawn in the plane without edge crossings.

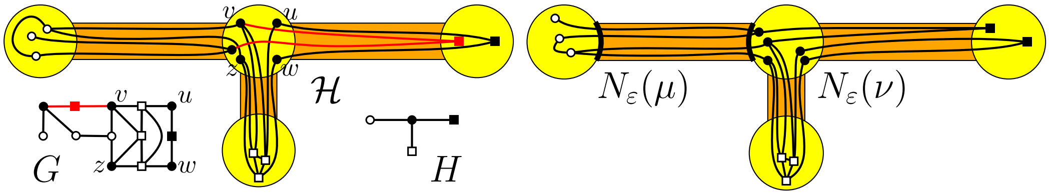


Figure 1 Instance of c-planarity with embedded pipes. The partition of the vertex set of G into clusters is encoded by the shape of vertices. An H-compatible embedding of a subgraph of G that cannot be extended to the whole G (left). An H-compatible embedding of G (right) inside $\mathcal{H}$. The valves of $\rho = \nu\mu$ at $N_\varepsilon(\nu)$ and $N_\varepsilon(\mu)$ are highlighted by bold arcs.

thickening $\mathcal{H}$ of H is the union of $N_\varepsilon(\nu)$, for all $\nu \in V(H)$ and $N_{\varepsilon'}(\rho)$, for all $\rho \in E(H)$[2]. Let the **pipe** of $\rho \in E(H)$ be the closure of $N_{\varepsilon'}(\rho) \setminus (N_\varepsilon(\nu) \cup N_\varepsilon(\mu))$, where $\rho = \nu\mu$. Let the **valve** of ρ at ν be the curve obtained as the intersection of $N_\varepsilon(\nu)$ and the pipe of ρ. We put $\varepsilon' < \varepsilon = \varepsilon(H)$, where $\varepsilon(H)$ is the same as in the previous paragraph, so that the valves are pairwise disjoint in $\mathcal{H}$.

In the combinatorial formulation of the problem, we are to decide if the given isotopy class of G contains an embedding contained in $\mathcal{H}$, where the vertices in V_ν, for every ν, are drawn in the interior of $N_\varepsilon(\nu)$ and every edge crosses the boundary of $N_\varepsilon(\nu)$, for every $\nu \in V(H)$, at most once. This does not change the problem as observed in [7]. Such an embedding of G is H**-compatible**. Let $E_{\nu\mu} = \{uv \in E(G) | \ v \in V_\nu, u \in V_\mu\}$. An H-compatible embedding of G is encoded by G, H, and a set of total orders $(E_{\nu\mu}, <_\omega)$, for every $\nu\mu \in E(H)$ and a valve ω of $\nu\mu$, where $(E_{\nu\mu}, <_\omega)$ encodes the order of crossings of ω with edges along ω. The isotopy class of G is encoded by a choice of the outer face, a set of rotations at its vertices and a containment relation of its connected components as described in Section 2. Since we are interested only in combinatorial aspects of the problem, H is also given by the isotopy class of its embedding. Throughout the paper we assume that G and H are given as above.

▶ **Theorem 1.** *There exists an $O(n^2)$-time algorithm that decides if the given isotopy class of G contains an H-compatible embedding. An H-compatible embedding of G can be also constructed in $O(n^2)$ time if it exists. In other words, c-planarity with embedded pipes is tractable, when an isotopy class of a desired embedding of G is fixed.*

As a corollary of our result we obtain that we can test in polynomial time if a piecewise linear drawing of a graph in the plane is approximable by an embedding and construct such an embedding if it exists. As we previously discussed, this extends results in [1, 7] and also [25].

▶ **Corollary 2.** *There exists an $O(n^4)$-time algorithm that decides if a piecewise linear (possibly degenerate) drawing of a graph in the plane is approximable by an embedding, and constructs such an embedding if it exists, where n is the size of the representation of the drawing.*

Extensions of our results. By [23, Theorem 3.1] and Fáry–Wagner theorem [12], our result holds also in the setting of rectilinear, i.e., straight-line, drawings of graphs. To extend it further in this setting by allowing “forks” seems to be just a little bit technical.

[2] Throughout the paper we denote vertices and edges of H by Greek letters.

In a recent manuscript [16], we verified a conjecture of M. Skopenkov [25, Conjecture 1.6] implying that our problem is tractable, when we lift the restriction on the isotopy class G. This does not imply that the problem with the restriction on the isotopy class G is tractable except when G is connected. The running time of the algorithm, that is implied by [16], is $O(|V|^{2\omega})$, where $O(n^{\omega})$ is the running time of the fastest algorithm for multiplying a pair of n by n matrices. Since $\omega > 2$ due to the matrix size, this is much worse that the running time claimed by Theorem 1. Furthermore, the algorithm is not constructive.

As noted by Chang et al. [7], the technique of Cortese et al. [10] extends directly from the plane to any closed two-dimensional surface. The same holds for our method, but since considering general two-dimensional surfaces does not bring anything substantially new to our treatment of the problem, for the sake of simplicity we consider only the planar case.

Strategy of the proof of Theorem 1. Recall that the **input** of our algorithm is a triple (G, H, γ), where the partition of the vertex set of G corresponds to the map γ from the set of vertices of G to the set of vertices of H. Hence, for $v \in V_\nu$, where $\nu \in V(H)$, we have $\gamma(v) = \nu$. The input (G, H, γ) is **positive** if there exists an H-compatible embedding of G in the given isotopy class of G, and **negative** otherwise.

Main troubles in constructing a polynomial time algorithm for our problem are caused by so called "spurs" such as the red vertex in Figure 1 (left), i.e., connected components in subgraphs of G induced by clusters, whose all adjacent vertices belong to the same cluster. Due to the presence of spurs it is hard to see that our problem is tractable even in the case, when G is a path.

The centerpiece of our method is an extension of the definition of the derivative of maps of intervals/loops (corresponding to the case, when G is a path/cycle, in our terminology) in the plane introduced by Minc [22]. We adapt this notion to the setting of c-planarity with embedded pipes. The derivative is an operator that takes (G, H, γ), and either detects that there exists no H-compatible embedding of G in the given isotopy class of G, or outputs (G', H', γ'), that is also a valid input for our algorithm, such that (G, H, γ) is positive if and only if (G', H', γ') is positive (Lemma 5). Intuitively, H' is reminiscent of the line graph of H and the subgraphs of G, that are mapped by γ to the edges of H, are turned into subgraphs of G' mapped by γ' into vertices of H'. This results in a shortening of problematic spurs, and zooming into the structure of the map γ. We show that by iterating the derivative $|E(G)|$ times we either detect that there exists no H-compatible embedding of G in the given isotopy class of G, or we arrive at an input without problematic spurs (Lemma 6). Since it is fairly easy to solve the problem for the latter inputs; the derivative at every iteration can be computed in linear time in $|V(G)|$; and by derivating the size of the input is increased only by a little, the tractability follows.

The operation of node expansion and base contraction introduced by Cortese et al. [10] resemble the derivative. The main difference is that these two operations affect only a single cluster or a pair of clusters in (G, H, γ), and therefore they are local, whereas the derivative changes the whole input. We are very positive that our method is applicable to other graph drawing problems related to c-planarity whose tractability is open. This is documented by our recent manuscript [16] in which a similar technique was applied.

The derivative is applied to an input (G, H, γ), in which every cluster V_ν induces in G an independent set. Such an input is in the **normal form**. The detailed description of the algorithm proving Theorem 1 is in Section 3. We show in Section 3.1 that an input can be assumed to be in the normal form. The definition of the derivative is given in Section 3.2, and sufficiently simplified inputs are dealt with in Section 3.3.

2 Preliminaries

Throughout the paper we tacitly use Jordan-Schönflies theorem for polygons.

Let $G = (V, E)$ denote a planar graph possibly with multiple edges and loops. For $V' \subseteq V$ we denote by $G[V']$ the sub-graph of G induced by V'. A **star** $St(v)$ of a vertex v in a graph G is the subgraph of G consisting of all the edges incident to v. Throughout the paper we use standard graph theoretical notions such as path, cycle, walk, vertex degree $deg(v)$ etc., see [11].

A **drawing** $\mathcal{D}(G)$ is a representation of G in the plane, where every vertex in V is represented by a point and every edge $e = uv$ in E is represented by a simple piecewise linear curve joining the points that represent u and v. Thus, a drawing can be thought of as a map from G understood as a topological space into the plane. In a drawing, we additionally require every pair of distinct curves representing edges to meet only in finitely many points each of which is a proper crossing or a common endpoint. In a **degenerate** drawing, we allow a pair of distinct vertices to be represented by the same point and a pair of edges to be represented by the same curve. Note that we do not allow an edge to pass through a vertex by the definition of the drawing, or in other words, we do not allow a drawing to contain **forks** [7]. A drawing in which every vertex is represented by a unique point and every edge by a unique curve is **non-degenerate**. In a non-degenerate drawing, multiple edges are mapped to distinct arcs meeting at their endpoints. In the paper we consider non-degenerate drawings, except for Corollary 2. An **embedding** is a non-degenerate drawing with no edge crossings. A graph given by an embedding in the plane is a **plane graph**. If it leads to no confusion, we do not distinguish between a vertex or an edge and its representation in the drawing and we use the words "vertex" and "edge" in both contexts.

The following lemma is well known.

▶ **Lemma 3.** *Let G be a plane graph with n vertices such that G does not contain a pair of multiple edges joining the same pair of vertices that form a face of size two, i.e., a lens, except for the outer face. The graph G has $O(n)$ edges.*

The **rotation** at a vertex in an embedding of G is the counterclockwise cyclic order of the edges, that are incident to the vertex, which is defined by the order of their end pieces at the vertex in the embedding. The rotation at a vertex is stored as a doubly linked list of edges. Furthermore, we assume that for every edge of G we store a pointer to its preceding and succeeding edge in the rotation at both of its end vertices. The **interior** and **exterior** of a cycle in an embedded graph is the bounded and unbounded, respectively, connected component of its complement in the plane. Similarly, the **interior** of an inner face and outer face in an embedded connected graph is the bounded and unbounded, respectively, connected component of the complement of its facial walk in the plane bounded by the walk. An embedding of a connected graph G is up to an isotopy described by the rotations at its vertices and the choice of its outer (unbounded) face. If G is not connected the isotopy class of its embedding is described by isotopy classes of its connected components $G_1, \ldots, G_l$ and the containment relation $G_i \subset f$, for every G_i, where f is a face of G_j, $j \neq i$, such that G_i is embedded in the interior of f.

3 Proof of Theorem 1

Let (G, H, γ) be the input of our algorithm. We naturally extend γ to edges: $\gamma(vu) = \rho = \nu\mu$, for $v \in V_\nu$ and $u \in V_\mu$, and to subgraphs G_1 of G: $\gamma(G_1) = H_1 = (V(H_1), E(H_1))$ such that

$V(H_1) = \{\nu \in V(H) | \gamma(v) = \nu,\ v \in V(G_1)\}$ and $E(H_1) = \{\rho \in E(H) |\ \gamma(e) = \rho, e \in E(G_1)\}$.

A vertex $\nu \in V(H)$ of degree two is **redundant** if $V_\nu \subseteq V(G)$ is an independent set consisting of vertices of degree two such that for every $v \in V_\nu$ we have $\gamma(vu) \neq \gamma(vw)$, where u and w are the two neighbors of v. We assume that every edge of H is used by at least one edge of G, i.e., for every $\rho \in E(H)$ there exists $e \in E(G)$ such that $\gamma(e) = \rho$.

3.1 The normal form

Similarly as in [15], the input (G, H, γ) is in the **normal form** if

1. every cluster $V_\nu \subseteq V(G)$, for $\nu \in V(H)$, is an independent set without isolated vertices; and
2. H does not contain a pair of redundant vertices joined by an edge.

We remark that (2) is required only due to the running time analysis. We do not forbid redundant vertices completely, since we do not allow H to contain multiple edges. Indeed, suppressing all vertices of degree two in a graph can lead to multiple edges. In what follows we show how to either detect that no H-compatible embedding in the given isotopy class of G exists just by considering the subgraph of G induced by a single cluster V_ν, or construct an input (G^N, H^N, γ^N) in the normal form, which is positive if and only if the input (G, H, γ) is positive. Clearly, (2) can be assumed without loss of generality. Before establishing the other condition we introduce a couple of definitions.

A **contraction** of an edge $e = uv$ in an embedding of a graph is an operation that turns e into a vertex by moving v along e towards u while dragging all the other edges incident to v along e. By a contraction we can introduce multiple edges or loops at the vertices. We will also use the following operation which can be thought of as the inverse operation of the edge contraction in an embedding of a graph. A **vertex split**, see Figure 2 Left, in an embedding of a graph G is an operation that replaces a vertex v by two vertices u and w joined by a crossing-free edge so that the neighbors of v are partitioned into two parts according to whether they are joined with u or w in the resulting drawing. The rotations at u and w are inherited from the rotation at v. When applied to G, the operations are meant to return a graph given by an isotopy class of its embedding; the same applies to vertex multisplit defined later. Note that a contraction can be carried out in $O(1)$ time, since it amounts to merging a pair of doubly linked lists, and redirecting at most four pointers. The same applies to the vertex split.

In order to satisfy (1), by a series of successive edge contractions we contract each connected component of $G[V_\nu]$, for all $\nu \in V(H)$, to a vertex. Since rotations are stored as doubly linked lists, contracting all such connected components can be carried out in linear time. We delete any created loop and isolated vertices. If a loop at a vertex from V_ν contains a vertex from a different cluster V_μ, $\nu \neq \mu$, in its interior we know that the input is negative, since for every μ all the vertices in V_μ must be contained in the outer face of $G[V_\nu]$ if the input is positive. All this can be easily checked in time linear in $|V(G)|$ by the breadth-first or depth-first search algorithm. If a loop at a vertex from V_ν does not contain a vertex from a different cluster, deleting the loop preserves the existence of an H-compatible embedding in the given isotopy class of G. Indeed, isolated vertices and deleted empty loops can be reintroduced in an H-compatible embedding of the resulting graph, and contracted edges recovered via vertex splits. Let (G^N, H^N, γ^N) denote the resulting input in the normal form. We proved the following.

▶ **Lemma 4.** *If a loop at a vertex v of G obtained during the previously described procedure contains in its interior a vertex u of G satisfying $\gamma(v) \neq \gamma(u)$, then the input (G, H, γ) is negative. Otherwise, the input (G, H, γ) is positive if and only if (G^N, H^N, γ^N) is positive.*

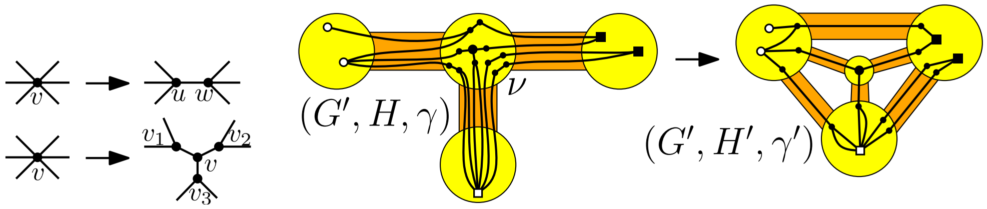


Figure 2 Left: Operation of vertex split (top) and multisplit (bottom). Right: The derivative of (G, H, γ) in the normal form. On the left the input after splitting vertices, and on the right the obtained derivative; in the example we have $H' = H'_\nu$, since every other H_μ, for $\nu \neq \mu$, is a trivial graph with one vertex.

3.2 Derivative

We present the operation of the derivative that simplifies the input, and whose iterating results in an input that is easy to deal with. Such inputs are treated in Section 3.3. Before we describe the derivative we give a couple of definitions.

A **vertex multisplit**, see Figure 2 Left, in an embedding of a graph G is an operation producing an embedding of a graph obtained from G by replacing a vertex v and its adjacent edges with a star $(\{v, v_1, \ldots, v_l\}, \{vv_1, \ldots, vv_l\})$, where $l \leq deg(v)$, so that the resulting underlying graph has vertex set $V(G) \cup \{v_1, \ldots, v_l\}$ and edge set $(E(G) \setminus \{vu_1, \ldots, vu_{deg(v)}\}) \cup \{v_{i_j}u_j | \ j = 1, \ldots, deg(v)\} \cup \{vv_1, \ldots, vv_l\}$, where $u_1, \ldots, u_{deg(v)}$ are neighbors of v in G and $1 \leq i_j \leq l$, for all j. The rotations at $v_1, \ldots, v_l$ are inherited from the rotation at v so that by contracting all the edges of $St(v)$ in the resulting graph we obtain the original embedding of G. Note that a vertex multisplit can be carried out in $O(deg(v))$ time.

The rotation of $\nu \in V(H)$ is **consistent** with the rotation of $v \in V_\nu$ if the rotation given by $(\gamma(vv_1), \ldots, \gamma(vv_{deg(v)}))$, where $(vv_1, \ldots, vv_{deg(v)})$ is the rotation at v in an embedding of G in the given isotopy class, is the rotation at $\nu \in V(H)$ in the embedding of H.

The **derivative** of (G, H, γ) is the input (G', H', γ') obtained as follows, see Figure 2 Right.

First, we construct the graph G' from G by applying the following procedure to every vertex $v \in V(G)$ such that the star $\gamma(St(v))$ has at least two edges, and thus, v is not a "spur". In fact, we construct an auxiliary input (G', H, γ), where by slightly abusing the notation we will extend γ to take values on the vertices of G'. (In the second step we use (G', H, γ) to construct (G', H', γ').) The input (G, H, γ) is clearly negative, if there exists a vertex v in G with four incident edges $vv_1, \ldots, vv_4 \in E(G)$ such that vv_1, vv_2, vv_3 and vv_4 appear in the rotation at v in the given order and $\gamma(vv_1) = \gamma(vv_3) \neq \gamma(vv_2), \gamma(vv_4)$. Otherwise, the following operations of vertex split and multisplit are applicable to G. Let the **valency** of a vertex $v \in V(G)$ be $val(v) := |E(\gamma(St(v)))|$. Thus, the valency count the size of the set of edges of H that the edges incident to v are mapped to.

If $val(v) = 2$, we apply the operation of vertex split to v thereby turning it into an edge uw as follows. Let $E(\gamma(St(v))) = \{\rho_1, \rho_2\}$. Let $v_1, \ldots, v_{deg(v)}$ be the neighbors of v. Let $\{v_1 \ldots v_l\} \cup \{v_{l+1} \ldots v_{deg(v)}\}$ be the partition of the neighbors of v such that $\gamma(vv_1) = \ldots = \gamma(vv_l) = \rho_1$ and $\gamma(vv_{l+1}) = \ldots = \gamma(vv_{deg(v)}) = \rho_2$. We put $\gamma(u), \gamma(w) := \gamma(v)$, and join u by an edge with the vertices in $\{v_1 \ldots v_l\}$ and w with the vertices in $\{v_{l+1} \ldots v_{deg(v)}\}$.

If $val(v) \geq 3$, we analogously apply the operation of vertex multisplit to v so that we replace v with a star $(\{v, v_1, \ldots, v_l\}, \{vv_1, \ldots, vv_l\})$ with $l := val(v)$ edges, in which the set of incident edges of every leaf vertex v_i is $\{vv_i\} \cup \{v_iu | \ vu \in \gamma^{-1}[\rho_i]\}$, where $E(\gamma(St(v))) = \{\rho_1, \ldots, \rho_l\}$, and for every such leaf $\gamma(v_i) := \gamma(v)$.

Let $V_{\geq 3} \subset V(G)$ denote the set of vertices in G consisting of the vertices $v \in V(G)$ with $val(v) \geq 3$. Note that $V_{\geq 3}$ can be treated also as a subset of $V(G')$. Let $E_2 \subset E(G')$ denote the set of edges in G' consisting of every edge uw obtained by splitting $v \in V(G)$ such that $val(v) = 2$. Let $\mathcal{C}$ denote the set of connected components of $G' \setminus E_2 \setminus V_{\geq 3}$. Note that every connected component of $\mathcal{C}$ is mapped to an edge of H by γ.

Second, we construct H': $V(H') := \{\rho^* | \rho \in E(H)\} \cup \{\nu_v |\ v \in V_{\geq 3}\}$, and $E(H') := \{\nu_v \rho^* |\ \rho \in E(\gamma(St(v)))\} \cup \{\gamma(C)^* \gamma(D)^* |\ C, D \in \mathcal{C}\ s.t.$ there exists $e \in E_2$ joining C with $D\}$. We put $\gamma'(v) := \gamma(C)^*$, for $v \in V(C)$ where $C \in \mathcal{C}$; and $\gamma(v) := \nu_v$, for $v \in V_{\geq 3}$.

Finally, the embedding of H', if it exists, is constructed as follows. For $\nu \in V(H)$, let C_ν be the cycle with the vertex set $\{\rho^* |\ \rho = \nu\mu \in E(H)\}$ that captures the rotation at ν, i.e., a pair of vertices ρ_0^* and ρ_1^* is joined by an edge in C_ν if ρ_0 and ρ_1 are consecutive in the rotation at ν. Let H'_ν, for $\nu \in V(H)$, denote the subgraph of H' induced by $\{\rho^* |\ \rho = \nu\mu \in E(H)\} \cup \{\nu_v | \gamma(v) = \nu\}$. Let $\hat{H}'_\nu$ be obtained from H'_ν by adding to H'_ν (1) the missing edges of the cycle C_ν; and (2) a new vertex joined by the edges exactly with all the vertices of C_ν. Note that $\hat{H}'_\nu$ is vertex three-connected, and hence, if $\hat{H}'_\nu$ is planar, then the rotations at vertices in its embedding are determined up to the choice of orientation.

Suppose that every $\hat{H}'_\nu$, for $\nu \in V(H)$, is a planar graph. Let us fix for every $\nu \in V(H)$ an embedding of H'_ν, in which the cycle C_ν bounds the outer face and its orientation corresponds to the rotation of ν. Such an embedding is obtained as a restriction of an embedding of $\hat{H}'_\nu$. Note that for every ν the graph H'_ν does not have multiple edges. Since H also does not have multiple edges, H'_ν and H'_μ, for $\nu \neq \mu$, are either disjoint (if $\nu\mu \notin E(H)$) or intersect in a single vertex $(\nu\mu)^*$ (if $\nu\mu \in E(H)$). It follows that H' does not have multiple edges. The desired embedding of H' is obtained by combining embeddings of H'_ν, for $\nu \in V(H)$, in the same isotopy class as the embeddings of H'_ν, that we fixed above, by identifying the corresponding vertices so that the restriction of the obtained embedding of H' to every H'_ν has the rest of H' in the interior of the outer face (of this restriction).

Note that the construction of (G', H', γ') can be carried out in $O\left(\sum_{v \in V(G')} deg(v)\right) = O(|V(G)|)$ thanks to the doubly-linked lists that we use to store the rotations of the vertices of G and H.

▶ **Lemma 5.** *The input (G, H, γ) is negative if one of the following three conditions is satisfied. (1) There exists a vertex v in G with four incident edges $vv_1, \ldots, vv_4 \in E(G)$ such that vv_1, vv_2, vv_3 and vv_4 appear in the rotation at v in the given order and $\gamma(vv_1) = \gamma(vv_3) \neq \gamma(vv_2), \gamma(vv_4)$. (2) The graph $\hat{H}'_\nu$, for some $\nu \in V(H)$, is not planar. (3) The rotation of a vertex $\nu_v \in V(H'_\nu)$, for some $\nu \in V(H)$ and $v \in V(G')$, in the obtained embedding of H'_ν is not consistent with the rotation of v in G'.*

Otherwise, the input (G, H, γ) is positive if and only if the input (G', H', γ') is positive.

3.3 Locally injective inputs

Let the **potential** $p(G, H, \gamma) = |E(G)| - |E(H)|$. Obviously, $p(G, H, \gamma) \geq 0$, and $p(G, H, \gamma) = 0$ if G is isomorphic to H. The input in the normal form (G, H, γ) is **locally injective** if

(i) the restriction of γ to $V(St(v))$ is injective, for all $v \in V(G)$; and

(ii) for every degree one vertex v in G its unique incident edge e satisfies the following. If $\gamma(e) = \gamma(f)$ then $e = f$ for all $f \in E(G)$.

Given an input (G, H, γ), the vertex $v \in V(G)$ is **fixed** if the condition of property (i) holds for v, and v is alone in its cluster, i.e., $\gamma(u) = \gamma(v)$ implies $u = v$. If v is fixed then

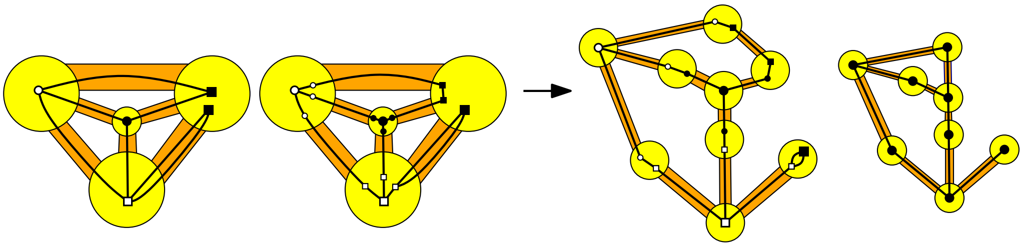


Figure 3 Constructing the normal form and derivating one more time the derivative from Figure 2 on the left, we obtain an input that is strongly locally injective in the normal form on the right.

we call $\gamma(v) = \nu \in V(H)$ also **fixed**. Note that the edges incident to fixed vertices do not contribute to the potential.

For a non-locally injective (G, H, γ) in the normal form, by Lemma 5 we either easily detect that there does not exist an H-compatible embedding of G in the given isotopy class, or we construct the input (G', H', γ') having a smaller potential after being brought to the normal form, such that (G', H', γ') is positive if and only if (G, H, γ) is positive. The following lemma implies that by iterating the derivative at most $|E(G)| = O(|V(G)|)$ many times we obtain an input that is locally injective.

▶ **Lemma 6.** *If (G, H, γ) is in the normal form then $p((G')^N, (H')^N, (\gamma')^N) \leq p(G, H, \gamma)$. If additionally (G, H, γ) is not locally injective then the inequality is strict.*

Given an input (G, H, γ) in the normal form. As in Section 3.2, let $V_{\geq 3} \subseteq V(G)$ denote the set of vertices in G consisting of the vertices $v \in V(G)$ with $val(v) \geq 3$. The input is **strongly locally injective** if it is locally injective and

(iii) every vertex in $V_{\geq 3}$ is fixed.

For convenience, we would like to work with strongly locally injective inputs, see Figure 3. The following lemma shows that if the input (G, H, γ) is locally injective, but not strongly, we just derivate it one more time in order to arrive at a strongly locally injective input.

▶ **Lemma 7.** *Suppose that (G, H, γ) in the normal form is locally injective. Then in $((G')^N, (H')^N, (\gamma')^N)$, every vertex $v \in V((G')^N)$, such that $val(v) \geq 3$, is fixed. Moreover, $((G')^N, (H')^N, (\gamma')^N)$ is still locally injective.*

Proof. The lemma follows directly from the definition of the derivative. ◀

Deciding in, roughly, quadratic time in $p(G, H, \gamma)$, which is sufficient for us, whether the strongly locally injective input (G, H, γ) is positive, is quite straightforward. The reason is that in this case the order of crossings of a valve with edges, that are incident to the same vertex v of G, along the valve in an H-compatible embedding of G is determined by the rotation at v. In order to decide if a desired H-compatible embedding of G exists, we just detect if for every valve ω such an order of all the edges crossing ω exists, such that together the orders are compatible. To this end we consider relations between unordered pairs of edges of G such that the edges in a pair are mapped by γ to the same edge of H, and two pairs are related if they intersect in a pair of vertices. In the following we assume that (G, H, γ) is strongly locally injective.

Let $\Xi = \{\{e, f\}|\ e, f \in E(G)\ s.t.\ e \neq f \text{ and } \gamma(e) = \gamma(f)\}$. Two elements $\{e_1, f_1\} \in \Xi$ and $\{e_2, f_2\} \in \Xi$ are **neighboring** if $|e_1 \cap e_2| = 1$, $|f_1 \cap f_2| = 1$ and $\gamma(e_1 \cap e_2) = \gamma(f_1 \cap f_2)$; we write $\{e_1, f_1\} \sim \{e_2, f_2\}$. An element $\{e_1, f_1\} \in \Xi$ is a **boundary pair** if there exists at most one $\{e_2, f_2\} \in \Xi$ such that $\{e_1, f_1\}$ and $\{e_2, f_2\}$ are neighboring. Let $\Xi_1, \ldots \Xi_l$ be equivalence classes of the transitive closure of the relation $\sim$. A boundary pair $\{e_1, f_1\} \in \Xi$ is **determined** if there exists a pair of edges e_2 and f_2 such that $|e_1 \cap e_2| = 1$, $|f_1 \cap f_2| = 1$, $\gamma(e_1 \cap e_2) = \gamma(f_1 \cap f_2)$ and $\gamma(e_2) \neq \gamma(f_2)$. By properties (i) and (iii) of strong local injectivity, the subgraph G_Ξ of G induced by $\bigcup_{\{e,f\} \in \Xi} \{e, f\}$ has maximum degree two. First, we consider the case when a connected component of G_Ξ does not contain a vertex of degree one.

▶ **Lemma 8.** *If there exists an equivalence class Ξ_c, such that the subgraph G_{Ξ_c} of G induced by $\bigcup_{\{e,f\} \in \Xi_c} \{e, f\}$ is a cycle, then (G, H, γ) is a negative input.*

Note that Lemma 8 does not cover the case when G_{Ξ_c} is a union of two cycles. By (ii), it must be that if Ξ_c contains a boundary pair, then it, in fact, contains exactly two boundary pairs, both of which are determined. Hence, in the following we assume that every Ξ_c either gives rise to a pair of cycles, or contains exactly two determined boundary pairs. We construct for every valve ω of $\rho \in E(H)$ the relation $(E_\rho, <_\omega)$, where $E_\rho = \{e \in E(H) | \ \gamma(e) = \rho\}$. We define relations $(E_\rho, <_\omega)$ by propagating relations enforced by the determined boundary pairs, for every determined pair contained in Ξ. We assume that $(E_\rho, <_\omega)$ encodes the increasing order of the crossing points of edges with ω as encountered when traversing $\omega \subset N_\varepsilon(\nu)$ in the direction inherited from the counterclockwise orientation of the boundary of $N_\varepsilon(\nu)$.

Let $\{e_1, f_1\} \in \Xi_c \subseteq \Xi$ be determined. Let $\Xi_c = \{\{e_1, f_1\}, \ldots \{e_m, f_m\}\}$ such that $\{e_p, f_p\} \sim \{e_{p+1}, f_{p+1}\}$. Let $\gamma(e_1) = \gamma(f_1) = \nu\mu, \gamma(e_0) = \nu\mu', \gamma(f_0) = \nu\mu''$, where $\mu' \neq \mu''$ and $|e_0 \cap e_1| = 1$ and $|f_0 \cap f_1| = 1$. W.l.o.g. we suppose that $\nu\mu, \nu\mu'$ and $\nu\mu''$ appear in the rotation of ν in this order counterclockwise. Let ω_1 be the valve of $\nu\mu$ at ν. Let ω_2 be the valve of $\nu\mu$ at μ. We put the relation $f_1 <_{\omega_1} e_1$ into $(E_{\nu\mu}, <_{\omega_1})$ and $e_1 <_{\omega_2} f_1$ into $(E_{\nu\mu}, <_{\omega_2})$. Recursively, we put $f_{p+1} <_{\omega_{2p+1}} e_{p+1}$ into $(E_{\nu\mu}, <_{\omega_{2p+1}})$ and $e_{p+1} <_{\omega_{2(p+1)}} f_{p+1}$ into $(E_{\nu\mu}, <_{\omega_{2(p+1)}})$, if $f_p <_{\omega_{2p-1}} e_p$ and $e_p <_{\omega_{2p}} f_p$, and vice-versa, where ω_{2p} and ω_{2p+1} are valves contained in the boundary of the same disc.

If G_{Ξ_c} is a union of two disjoint cycles we add $f_p <_{\omega_{2p}} e_p$ and $e_p <_{\omega_{2p-1}} f_p$, or $f_p >_{\omega_{2p}} e_p$ and $e_p >_{\omega_{2p-1}} f_p$ for every p, in correspondence with the isotopy class of G.

▶ **Lemma 9.** *Suppose that every equivalence class Ξ_c contains exactly two determined boundary pairs, or G_{Ξ_c} is a union of two disjoint cycles. We can test in $O((p(G, H, \gamma))^2 + |V(G)|)$ time if (G, H, γ) is positive or negative.*

3.4 Algorithm

We give a description of the decision algorithm proving the first part of the theorem. The running time analysis using Lemmas 6, 7 and 9, and the constructive algorithm is omitted in this extended abstract.

Decision Algorithm. Let $(G, H, \gamma) = (G_0, H_0, \gamma_0)$ be the input. We work with inputs in which G contains multiple edges and loops. However, w.l.o.g. we assume that G does not contain a pair of multiple edges joining the same pair of vertices that form a face of size two, i.e., a lens, except for the outer face. Moreover, we assume that whenever a lens is created during the execution of the algorithm, the lens is eliminated by deleting one of its edges.

An execution of the algorithm is divided into steps. During the s-th step we process (G_s, H_s, γ_s) and output $(G_{s+1}, H_{s+1}, \gamma_{s+1})$ as follows.

First, by following the procedure described in Section 3.1 we either construct an instance $((G_s)^N,(H_s)^N,(\gamma_s)^N)$ in the normal form that is positive if and only if (G_s,H_s,γ_s) is positive, or output that (G,H,γ) is negative, if the hypothesis of the first part of Lemma 4 is satisfied.

Second, if $((G_s)^N,(H_s)^N,(\gamma_s)^N)$ is not strongly locally injective we proceed as follows. If (G_s,H_s,γ_s) satisfies the hypothesis of the first part of Lemma 5 with $((G_s)^N,(H_s)^N,(\gamma_s)^N)$ playing the role of (G,H,γ) we output that (G,H,γ) is negative; otherwise we construct the derivative $((G_s^N)',(H_s^N)',(\gamma_s^N)')=(G_{s+1},H_{s+1},\gamma_{s+1})$ defined in Section 3.2 and proceed to the $(s+1)$-st step. Otherwise, $((G_s)^N,(H_s)^N,(\gamma_s)^N)$ is strongly locally injective and we construct equivalence classes $\Xi_1,\ldots\Xi_l$ from Section 3.3 defined by $((G_s)^N,(H_s)^N,(\gamma_s)^N)$ and proceed as follows.

We check if there exists a class Ξ_c satisfying the hypothesis of Lemma 8. If this is the case, then we output that (G,H,γ) is negative. Otherwise, we construct relations $(E_\rho,<_\omega)$, for every $\rho\in(H_s)^N$ and its valve ω. If there exists $(E_\rho,<_\omega)$ that is not a total order we output that (G,H,γ) is negative; otherwise we check if the isotopy class of an H-compatible embedding of G_s enforced by relations $(E_\rho,<_\omega)$ is the same the given one and output that (G,H,γ) is positive if and only if this is the case.

The correctness of the algorithm follows directly from Lemma 4,5,8, and 9.

References

1 Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba Tóth. Recognizing weakly simple polygons. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2016.8.

2 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Strip planarity testing for embedded planar graphs. *Algorithmica*, 77(4):1022–1059, 2017.

3 Patrizio Angelini and Giordano Da Lozzo. Clustered Planarity with Pipes. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, 2016.

4 Therese C. Biedl. Drawing planar partitions III: Two constrained embedding problems. *Rutcor Research Report 13-98*, 1998.

5 Thomas Bläsius and Ignaz Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. In Christian A. Duncan and Antonios Symvonis, editors, *Graph Drawing - 22nd International Symposium, GD 2014, Würzburg, Germany, September 24-26, 2014, Revised Selected Papers*, volume 8871 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2014. doi:10.1007/978-3-662-45803-7_37.

6 David Dylan Bremner. *Point visibility graphs and restricted-orientation polygon covering.* PhD thesis, Simon Fraser University, 1993.

7 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1655–1670, 2015.

8 Markus Chimani, Giuseppe Di Battista, Fabrizio Frati, and Karsten Klein. Advances on testing c-planarity of embedded flat clustered graphs. In Christian A. Duncan and Antonios Symvonis, editors, *Graph Drawing - 22nd International Symposium, GD 2014, Würzburg, Germany, September 24-26, 2014, Revised Selected Papers*, volume 8871 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 2014. doi:10.1007/978-3-662-45803-7_35.

9 Pier Francesco Cortese, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Maurizio Pizzonia. C-planarity of c-connected clustered graphs. *J. Graph Algorithms Appl.*, 12(2):225–262, 2008.

10 Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. On embedding a cycle in a plane graph. *Discrete Mathematics*, 309(7):1856 – 1869, 2009.

11 Reinhard Diestel. *Graph Theory.* Springer, New York, 2010.

12 István Fáry. On straight line representation of planar graphs. *Acta Univ. Szeged. Sect. Sci. Math.*, 11:229–233, 1948.

13 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. How to draw a planar clustered graph. In Ding-Zhu Du and Ming Li, editors, *Computing and Combinatorics*, volume 959 of *Lecture Notes in Computer Science*, pages 21–30. Springer Berlin Heidelberg, 1995.

14 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for clustered graphs. In Paul Spirakis, editor, *Algorithms — ESA '95*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer Berlin Heidelberg, 1995.

15 Radoslav Fulek. C-planarity of embedded cyclic c-graphs. In *International Symposium on Graph Drawing and Network Visualization*, pages 94–106. Springer, 2016.

16 Radoslav Fulek and Jan Kyncl. Hanani-tutte for approximating maps of graphs. *CoRR*, abs/1705.05243, 2017. `arXiv:1705.05243`.

17 Radoslav Fulek, Jan Kynčl, Igor Malinovic, and Dömötör Pálvölgyi. Clustered planarity testing revisited. *Electronic Journal of Combinatorics*, 22, 2015.

18 Michael T. Goodrich, George S. Lueker, and Jonathan Z. Sun. C-planarity of extrovert clustered graphs. In Patrick Healy and Nikola S. Nikolov, editors, *Graph Drawing, 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005, Revised Papers*, volume 3843 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2005. URL: `https://doi.org/10.1007/11618058_20`, `doi:10.1007/11618058_20`.

19 Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel, Merijam Percan, and René Weiskircher. Advances in c-planarity testing of clustered graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Graph Drawing, 10th International Symposium, GD 2002, Irvine, CA, USA, August 26-28, 2002, Revised Papers*, volume 2528 of *Lecture Notes in Computer Science*, pages 220–235. Springer, 2002. URL: `https://doi.org/10.1007/3-540-36151-0_21`, `doi:10.1007/3-540-36151-0_21`.

20 Vít Jelínek, Eva Jelínková, Jan Kratochvíl, and Bernard Lidický. Clustered planarity: Embedded clustered graphs with two-component clusters. In Ioannis G. Tollis and Maurizio Patrignani, editors, *Graph Drawing, 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*, volume 5417 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2008. `doi:10.1007/978-3-642-00219-9_13`.

21 Eva Jelínková, Jan Kára, Jan Kratochvíl, Martin Pergel, Ondřej Suchý, and Tomáš Vyskočil. Clustered planarity: Small clusters in cycles and Eulerian graphs. *J. Graph Algorithms Appl.*, 13(3):379–422, 2009.

22 Piotr Minc. Embedding simplicial arcs into the plane. *Topol. Proc. 22*, pages 305–340, 1997.

23 Ares Ribó Mor. *Realization and Counting Problems for Planar Structures: Trees and Linkages, Polytopes and Polyominoes.* PhD thesis, Freie U., Berlin, 2006.

24 K Sieklucki. Realization of mappings. *Fundamenta Mathematicae*, 65(3):325–343, 1969.

25 Mikhail Skopenkov. On approximability by embeddings of cycles in the plane. *Topology and its Applications*, 134(1):1–22, 2003.

26 Godfried Toussaint. On separating two simple polygons by a single translation. *Discrete & Computational Geometry*, 4(3):265–278, 1989.

Structural Pattern Matching – Succinctly*

Arnab Ganguly[1], Rahul Shah[2], and Sharma V. Thankachan[3]

1 **University of Wisconsin - Whitewater, Whitewater, USA**
`gangulya@uww.edu`

2 **Louisiana State University, Baton Rouge, USA and NSF, Arlington, USA**
`rahul@csc.lsu.edu, rahul@nsf.gov`

3 **University of Central Florida, Orlando, USA**
`sharma.thankachan@ucf.edu`

Abstract

Let T be a text of length n containing characters from an alphabet Σ, which is the union of two disjoint sets: Σ_s containing static characters (s-characters) and Σ_p containing parameterized characters (p-characters). Each character in Σ_p has an associated complementary character from Σ_p. A pattern P (also over Σ) matches an equal-length substring S of T iff the s-characters match exactly, there exists a one-to-one function that renames the p-characters in S to the p-characters in P, and if a p-character x is renamed to another p-character y then the complement of x is renamed to the complement of y. The task is to find the starting positions (occurrences) of all such substrings S. Previous indexing solution [Shibuya, SWAT 2000], known as *Structural Suffix Tree*, requires $\Theta(n \log n)$ bits of space, and can find all *occ* occurrences in time $O(|P| \log \sigma + occ)$, where $\sigma = |\Sigma|$. In this paper, we present the first succinct index for this problem, which occupies $n \log \sigma + O(n)$ bits and offers $O(|P| \log \sigma + occ \cdot \log n \log \sigma)$ query time.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases Parameterized Pattern Matching, Suffix tree, Burrows-Wheeler Transform, Wavelet Tree, Fully-functional succinct tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.35

1 Introduction

Text Indexing is a classical problem defined as: pre-process a text T of length n containing characters from an alphabet Σ of size $\sigma \leq n$ and then build a data structure, such that given a pattern P (also over Σ) as a query, we can report all the *occ* starting positions (or simply, occurrences) of P in T. *Suffix Tree* is the ubiquitous data structure for this purpose [14]. Unfortunately, it requires $\Theta(n \log n)$ bits of space, which is too large for most practical purposes (15-50 times the text). Grossi and Vitter [13], and Ferragina and Manzini [6] addressed this problem by introducing space-efficient indexes, namely *Compressed Suffix Arrays* (CSA) and *FM-Index* respectively. Subsequently, a lot of progress has been made either in improving these initial breakthroughs [2, 7, 8, 18, 20], or to achieve space-efficient indexes for other problems which require suffix trees as a component [16, 23].

The key concept behind the FM-Index and the CSA is the *suffix link*: the suffix link of a node u points to a node v iff the string from root to v is the same as the string from root to u with the first character truncated. Suffix links have the following so called *rank-preserving* property: the leaves obtained by following suffix links from the leaves in u's

* Work was partially supported by NSF Grant CCF–1527435.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 35; pp. 35:1–35:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

subtree appear in the same relative lexicographic order in the subtree of v. However, in many important variants [1, 4, 5, 10, 15, 21, 22] of the suffix tree, such as the parameterized suffix tree, the 2D suffix tree, and the structural suffix tree, this rank-preserving property of suffix links does not hold. Consequently, there has been very little progress in designing compressed representations of these suffix tree variants. Only recently, Ganguly et al. [9] designed the first succinct index for parameterized pattern matching [1]. We consider its generalization [21], which has applications in RNA structural matching.

Throughout this paper, we use the following terminologies: Σ is an alphabet of size $\sigma \geq 2$, which is the union of two disjoint sets – Σ_s having σ_s static characters (s-characters) and Σ_p having σ_p parameterized characters (p-characters). For each p-character, we associate a p-character, called the complement character. For a string S, $|S|$ is its length, $S[i], 1 \leq i \leq |S|$, is its ith character and $S[i,j]$ is its substring from i to j. If $i > j$, $S[i,j]$ denotes an empty string. Also S_i denotes the circular suffix starting at position i. Specifically, S_i is S if $i = 1$ and is $S[i, |S|] \circ S[1, i-1]$ otherwise, where $\circ$ denotes the *concatenation*.

▶ **Definition 1.** Two equal-length strings S and S' are a *structural-match* (s-match) iff

- $S[i] \in \Sigma_s \iff S'[i] \in \Sigma_s$,
- $S[i] = S'[i]$ when $S[i] \in \Sigma_s$,
- there exists a one-to-one matching-function f that renames the p-characters in S to the p-characters in S', i.e., $S'[i] = f(S[i])$ when $S[i] \in \Sigma_p$, and
- if a p-character x in S is renamed to y in S', then the complement (if exists) of x in S is renamed to the complement of y in S'.

Consider the following examples. Let $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x, y, z\}$, where the complement pairs are w-x and y-z. Then $AxByCx$ is an s-match with $AyBxCy$; in this case, there are no complementary requirements. Also, $AxBwCx$ is an s-match with $AzByCz$; here, x is paired with z, and w (complement of x) is paired with y (complement of z). However, $AxBwCx$ is not an s-match with $AzBxCz$ (even though the one-to-one criterion is satisfied); this is because as x is paired with z, w should have been paired with y. Lastly, $AxBwCx$ is not an s-match with $AzBxCy$ because x has to be renamed to both z and y, which violates the one-to-one criterion.

We consider the following indexing problem introduced by Shibuya [21].

▶ **Problem 2.** *Let T be a text of length n over Σ. We assume T terminates in a uniquely appearing s-character \$. Index T, such that given a pattern P (also over Σ), we can report all starting positions (occurrences) of the substrings of T that are an s-match with P.*

Shibuya presented a $\Theta(n \log n)$-bit and $O(|P| \log \sigma + occ)$-time index for this problem. We present the following new result.

▶ **Theorem 3.** *By using an $n \log \sigma + O(n)$-bit index of T, we can count the number of s-matches of a pattern P in $O(|P| \log \sigma)$ time. Subsequently, each match can be reported in $O(\log \sigma \log n)$ time.*

1.1 Overview of Techniques

We start with the closely related parameterized matching (p-matching) problem of Baker [1]. Two strings are a p-match if they satisfy the first three criteria in Definition 1. Thus if two strings are an s-match, they are definitely also a p-match, but may not be true the other way around. To create an index for the p-matching problem (i.e., replace s-match by p-match in Problem 2), Baker [1] introduced an encoding scheme such that two strings are a p-match iff their encoded strings are the same. Using this encoding scheme, Baker obtained a linear space index for the p-matching problem. Similarly, the key to obtain a linear-space index for Problem 2 is an encoding scheme such that two strings are an s-match if their encoded strings are the same. Luckily, we already have such an encoding scheme. Specifically, using the encoding scheme of Shibuya [21], we can construct the *structural suffix tree* (s-suffix tree) as follows: first encode each suffix of T and then create a compact trie of these encoded suffixes. To report the occurrences of a pattern, we first find the highest node u in the s-suffix tree such that the string obtained by concatenating the edge labels from root to u is prefixed by the encoded pattern. Then, we report the starting positions of the encoded suffixes corresponding to the leaves in the subtree of u. However, Shibuya's encoding scheme (as well as Baker's scheme) has the following drawback: on prepending the preceding character of a suffix, the encoding of the original suffix changes. Consequently, FM-Index [6] and CSA [13] no longer work for these definitions of pattern matching.

Since the p-matching problem of Baker [1] is similar to Problem 2, one may be tempted to think that we can simply re-use (with minor adjustments) the succinct data structure of Ganguly et al. [9] for the p-matching problem. Although, this is true, the extension is not trivial. This is because, in contrast to the encoding scheme [1] used for p-matching, Shibuya's encoding scheme has a caveat: when we prepend the previous character of a suffix, the change in the encoding of the original suffix can occur at two positions. Hence, the index of Ganguly et al. [9] will no longer directly work. The first step, therefore, is a new encoding scheme which alleviates this problem, and a version of the s-suffix tree based on this encoding scheme; Section 2 presents the details.

Since we have now restricted the number of points of change (on prepending) to at most one, we use techniques similar to that employed by Ganguly et al. [9]. We store the number of distinct p-characters up to this point of change (from the start of the suffix) in $\approx \log \sigma$ bits per suffix. However, we make a distinction between the cases when the change is due to the complement of the prepended p-character versus the change due to the same p-character. This forms the backbone of our data structure, and we call it the *Structural Burrows-Wheeler Transform* (sBWT); the details are in Section 3.

The next step is to compute the starting positions of the lexicographically arranged encoded (with our new encoding scheme) suffixes. We implement the *Structural LF mapping* (sLF mapping), using which we can decode the starting positions without explicitly storing them. Summarizing our discussions thus far, we can see that the key is to compute sLF mapping. To this end, we use the sBWT and the topology of the s-suffix tree; the crucial insight is provided in Lemma 9. Based on this lemma, we implement sLF mapping in Section 4; space and time complexities are described in Lemma 14.

The last piece of the puzzle is to compute the suffix range of the encoded pattern (i.e., find the range of leaves under the node u defined at the beginning of this section). We again use sLF mapping, the s-suffix tree topology, and sBWT to implement a backward search procedure (like that in the FM Index [6] and succinct index for the p-matching problem [9]). The details of the backward search procedure for s-matching are in Section 5.

2 Linear-Space Index

We first consider the encoding scheme by Shibuya [21]. A string S is encoded into an equal-length string $\mathsf{sencode}(S)$ by replacing the first occurrence of every p-character in S by 0 and any other occurrence of a p-character by the difference in text position from its previous occurrence. Specifically, for any $i \in [1, |S|]$, $\mathsf{sencode}(S)[i] = S[i]$ if $S[i]$ is an s-character;

otherwise, $\mathsf{sencode}(S)[i] = (i - j)$, where $j < i$ is the last occurrence of $S[i]$ before i. If j does not exist, then $j = i$.

Now, for every p-character $S[i]$, where $\mathsf{sencode}(S)[i] = 0$, we find the rightmost $j < i$ such $S[j]$ is the complement of $S[i]$. If j exists, then replace $\mathsf{sencode}(S)[i]$ by $-(i - j)$. For e.g., $\mathsf{sencode}(AxBwAwCxAx) = A0B(-2)A2C6A2$, where the first step yields the string $A0B0A2C6A2$. Here, $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x\}$; additionally, w and x are complement of each other.

▶ **Fact 4** ([21]). *Two strings S and S' are an s-match iff $\mathsf{sencode}(S) = \mathsf{sencode}(S')$. Also S and a prefix of S' are an s-match iff $\mathsf{sencode}(S)$ is a prefix of $\mathsf{sencode}(S')$.*

2.1 New Encoding Scheme

Unfortunately, for our purposes, the encoding scheme defined in the previous sub-section suffers from a drawback. Specifically, let S be a string and x be a p-character. Then $\mathsf{sencode}(xS)[2, |S| + 1]$ can differ from $\mathsf{sencode}(S)$ at two distinct positions. For example, consider the string $S = wAwBxAx$. Here, $\Sigma_s = \{A, B\}$ and $\Sigma_p = \{w, x\}$; additionally, w and x are complement of each other. Then, $\mathsf{sencode}(S) = 0A2B(-2)A2$ and $\mathsf{sencode}(xS) = \mathsf{sencode}(xwAwBxAx) = 0(-1)A2B5A2$. We want to avoid such an encoding scheme as it will prevent us from using the techniques of Ganguly et al. [9]. To this end, we present the following new encoding scheme.

We encode a string S as $\Phi(S)$ as follows. If $S[i]$ is static, then $\Phi(S)[i] = S[i]$. Consider a p-character $S[i]$ and let $j^+ < i$ and $j^- < i$ be the rightmost occurrence of $S[i]$ and the complement of $S[i]$ in $S[1, i - 1]$. If there is no occurrence j^+ (resp. j^-), we let $j^+ = -1$ (resp. $j^- = -1$). If $j^+ = j^- = -1$, then replace $S[i]$ by 0. Otherwise, if $j^+ > j^-$, then $\Phi(S)[i] = (i - j^+)$. Otherwise, if $j^- > j^+$, then $\Phi(S)[i] = -(i - j^-)$. For example, $\Phi(AxByCx) = A0B0C4$ and $\Phi(AxBwAwCxAx) = A0B(-2)A2C(-2)A2$. Here, $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x\}$; additionally, w and x are complement of each other.

Importantly, note that we alleviate the problem of Shibuya's encoding. Specifically, $\mathsf{sencode}(xS)[2, |S| + 1]$ can differ from $\mathsf{sencode}(S)$ at most at one position, which is easily illustrated by choosing $S = wAwBxAx$. All we are left to do is show that our encoding scheme still guarantees that two strings are an s-match iff the corresponding encoded strings are the same, which is handled by the following lemma.

▶ **Lemma 5.** *Two strings S and S' are an s-match iff $\Phi(S) = \Phi(S')$. Also S and a prefix of S' are a p-match iff $\Phi(S)$ is a prefix of $\Phi(S')$.*

Proof. If S and S' are an s-match, then $\Phi(S) = \Phi(S')$ as S can be renamed to S' by applying the necessary one-to-one function. Therefore, it suffices to show that $\Phi(S) = \Phi(S')$ implies S and S' are an s-match. We note that the ith zero in $\Phi(S)$ (resp. in $\Phi(S')$) corresponds to the ith distinct p-character, say c_i (resp. c'_i), in S (resp. in S') such that neither c_i (resp. c'_i) nor its complement appear before. Thus, we establish the one-to-one mapping $c_i \to c'_i$. Let p be the position of an occurrence of c_i in S. Let $q > p$ be the minimum position (if any) where c_i (or, its complement) occurs in $S[p + 1, |S|]$. Since $\Phi(S') = \Phi(S)$, q is also the minimum position where c'_i (or, its complement) occurs in $S'[p + 1, |S'|]$. Therefore, if any position p is the occurrence of c_i (resp. its complement) in S, then p is the occurrence of c'_i (resp. its complement) in S'. ◀

▶ **Convention 6.** *The integer characters (corresponding to p-characters) are lexicographically smaller than s-characters. An integer character i comes before another integer character j iff $i < j$. Also, \$ is the largest character.*

2.2 Structural Suffix Tree

Structural Suffix Tree (sST) is the compacted trie of all strings in $\mathcal{P} = \{\Phi(T[k,n]) \mid 1 \leq k \leq n\}$. Each edge is labeled with a string over $\Sigma' = \Sigma_s \cup \{0, 1, \ldots n-1\}$. We use $\mathsf{str}(u)$ to denote the concatenation of edge labels on the path from root to node u. The path of each leaf node corresponds to the encoding of a unique suffix of T, and leaves are ordered in the lexicographic order of the corresponding encoded suffix. Clearly, sST consists of n leaves (one per each encoded suffix) and at most $n-1$ internal nodes. We also store the structural suffix array $\mathsf{sSA}[1,n]$ i.e., $\mathsf{sSA}[i] = j$ and $\mathsf{sSA}^{-1}[j] = i$ iff $\Phi(T[j,n])$ is the ith lexicographically smallest string in $\mathcal{P}$. Note that $\mathsf{str}(\ell_i) = \Phi(T[\mathsf{sSA}[i], n])$, where ℓ_i is the ith leftmost leaf in sST. The total space required is $\Theta(n \log n)$ bits.

To find all occurrences of P, traverse sST from root by following the edges labels and find the highest node u (called *locus*) such that $\mathsf{str}(u)$ is prefixed by $\Phi(P)$. Then find the range $[sp, ep]$ (called *suffix range* of $\Phi(P)$) of leaves in the subtree of u and report $\{\mathsf{sSA}[i] \mid sp \leq i \leq ep\}$ as the output. The query time is $O(|P| \log \sigma + occ)$, where occ is the number of occurrences of P in T.

We remark that the structural suffix tree described here varies from that by Shibuya [21]. Their tree is based on sencode and can be constructed in $O(n \log \sigma)$ time using $\Theta(n \log n)$ bits of working space. Based on Fact 4 and Lemma 5, we observe that the longest common prefix (LCP) of any two encoded suffix is the same whether we use sencode or Φ as the encoding function. Therefore, given Shibuya's tree, we can easily create sST by relabeling the edges, and then sorting them based on their first character and Convention 6. The additional time needed is $O(n)$ using any linear-time sorting algorithm. Summarizing, we can create sST in $O(n \log \sigma)$ time using $\Theta(n \log n)$ bits of working space.

3 Structural Burrows-Wheeler Transform

We use a similar transform to that of the Burrows and Wheeler [3], which we call as the Structural Burrows-Wheeler Transform (sBWT). Sort the circular suffixes T_x, $1 \leq x \leq n$, based on their $\Phi(\cdot)$ encoding, where character precedence is determined by Convention 6. Then, obtain the last character $L[i]$ of the ith lexicographically smallest circular suffix. Denote by f_i^+ (resp. f_i^-) the first occurrence of $L[i]$ (resp. the complement of $L[i]$) in the circular suffix T_i. In case, there is no occurrence of $L[i]$'s complement, we take $f_i^- = n+1$.

The sBWT is defined as $\mathsf{sBWT}[i] =$

$$\begin{cases} L[i], & \text{if } L[i] \in \Sigma_s \\ \text{number of distinct p-characters in } T_{\mathsf{sSA}[i]}[1, f_i^+], & \text{if } L[i] \in \Sigma_p \text{ and } f_i^+ < f_i^- \\ -\text{number of distinct p-characters in } T_{\mathsf{sSA}[i]}[1, f_i^-], & \text{if } L[i] \in \Sigma_p \text{ and } f_i^+ > f_i^- \end{cases}$$

▶ **Observation 7.** *For any $1 \leq i \leq n$, let $c = \mathsf{sBWT}[i]$. Then, $\Phi(T_{\mathsf{sSA}[i]-1}) =$*

$$\begin{cases} c \circ \Phi(T_{\mathsf{sSA}[i]})[1, n-1], & \textit{if } c \in \Sigma_s \\ 0 \circ \Phi(T_{\mathsf{sSA}[i]})[1, f_i^+ - 1] \circ f_i^+ \circ \Phi(T_{\mathsf{sSA}[i]})[f_i^+ + 1, n-1], & \textit{if } c \in [1, \sigma_p] \\ 0 \circ \Phi(T_{\mathsf{sSA}[i]})[1, f_i^- - 1] \circ -f_i^- \circ \Phi(T_{\mathsf{sSA}[i]})[f_i^- + 1, n-1], & \textit{if } c \in [-\sigma_p, -1] \end{cases}$$

The structural last-to-first column (sLF) mapping of i is the position at which the character at $L[i]$ lies in the first column of the sorted encoded suffixes. Specifically, $\mathsf{sLF}(i) = \mathsf{sSA}^{-1}[\mathsf{sSA}[i] - 1]$, where $\mathsf{sSA}^{-1}[0] = \mathsf{sSA}^{-1}[n]$. The following lemma is a straightforward adaptation of Theorem 3 in [9].

▶ **Lemma 8.** *Assume* $\mathsf{sLF}(\cdot)$ *can be computed in* t_{sLF} *time. By using an additional* $O(n)$*-bit data structure, we can compute* $\mathsf{sSA}[\cdot]$ *in* $O(t_{\mathsf{sLF}} \cdot \log n)$ *time.*

4 Implementing Structural LF Mapping

As highlighted by Lemma 8, the objective is to compute sLF. In this section, we show that $\mathsf{sLF}(i)$ can be computed in $O(\log \sigma)$ time using $n \log \sigma + O(n)$ bits.

▶ **Lemma 9.** *Consider two suffixes* i *and* j *corresponding to the leaves* ℓ_i *and* ℓ_j *in* sST.

(a) *If* $L[i]$ *is parameterized and* $L[j]$ *is static, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$.

(b) *If both* $L[i]$ *and* $L[j]$ *are static, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff either* $\mathsf{sBWT}[i] < \mathsf{sBWT}[j]$*, or* $\mathsf{sBWT}[i] = \mathsf{sBWT}[j]$ *and* $i < j$.

(c) *Assume* $i < j$ *and both* $L[i]$ *and* $L[j]$ *are parameterized. Let* u *be the lowest common ancestor of* ℓ_i *and* ℓ_j *in* sST*, and* z *be the number of* 0*'s in the string* $\mathsf{str}(u)$*. Then,*

1. *If* $|\mathsf{sBWT}[i]|, |\mathsf{sBWT}[j]| \leq z$*, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff*
   - *either* $\mathsf{sBWT}[i], \mathsf{sBWT}[j] > 0$ *and* $\mathsf{sBWT}[i] \geq \mathsf{sBWT}[j]$,
   - *or* $\mathsf{sBWT}[i] < 0 < \mathsf{sBWT}[j]$,
   - *or* $\mathsf{sBWT}[i], \mathsf{sBWT}[j] < 0$ *and* $|\mathsf{sBWT}[i]| \leq |\mathsf{sBWT}[j]|$
2. *If* $|\mathsf{sBWT}[i]| \leq z < |\mathsf{sBWT}[j]|$*, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff* $\mathsf{sBWT}[i] < 0$
3. *If* $|\mathsf{sBWT}[i]| > z \geq |\mathsf{sBWT}[j]|$*, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff* $\mathsf{sBWT}[j] > 0$
4. *If* $|\mathsf{sBWT}[i]|, |\mathsf{sBWT}[j]| > z$*, then* $\mathsf{sLF}(i) > \mathsf{sLF}(j)$ *iff*
   - *either* $\mathsf{sBWT}[i] = z + 1$*, the first character on the* u *to* ℓ_i *path is* 0*, and the first character on the* u *to* ℓ_j *path is not an s-character,*
   - *or* $\mathsf{sBWT}[j] = -(z + 1)$*, and the first character on the* u *to* ℓ_j *path is* 0.

Proof. (a) and (b): Follows immediately from Convention 6 and Observation 7. (c) Let $d = |\mathsf{str}(u)|$. Define f_i to be smaller of the two values f_i^+ or f_i^-. Similarly, f_j is defined. Clearly, the conditions (1)-(4) can be written as: (1) Both $f_i, f_j \leq d$, (2) $f_i \leq d$ and $f_j > d$, (3) $f_i > d$ and $f_j \leq d$, and (4) Both $f_i, f_j > d$. Then the claims (1)-(3) follow from Observation 7 and Convention 6. To prove (4), observe that if the suffixes i and j swap order on being prepended by their previous characters then at least either f_i or f_j equals $d + 1$. The claim follows from Observation 7 and Convention 6. ◀

4.1 Wavelet Tree (WT) over sBWT

Let $A[1, m]$ be an array over an alphabet of size σ. There exists a data structure of size $m \log \sigma + o(m)$ bits, using which the following queries can be answered in $O(\log \sigma / \log \log m)$ time [6, 11, 12, 17]:

- $A[i]$,
- $\mathsf{rank}_A(i, x)$ = the number of occurrences of x in $A[1, i]$,
- $\mathsf{select}_A(i, x)$ = the ith occurrence of x in $A[1, m]$, and
- $\mathsf{count}_A(i, j, x, y)$ = number of elements in $A[i, j]$ that are at least x and at most y.

We drop the subscript A when the context is clear. Recall that the sBWT is a string of length n over the alphabet set $\Sigma_s \cup \{1, 2, \ldots, \sigma_p\} \cup \{-1, -2, \ldots, -\sigma_p\}$ of size $\sigma' = \sigma_s + 2\sigma_p \leq 2\sigma$. By using a WT over sBWT in $n \log \sigma' + o(n) = n \log \sigma + O(n)$ bits, we can support the above operations over sBWT in time $O(1 + \log \sigma / \log \log n)$.

4.2 Succinct representation of sST

A tree having m nodes can be represented in $2m + o(m)$ bits, such that if each node is labeled by its pre-order rank, the following operations can be supported in $O(1)$ time (note that $m < 2n$ in our case) [19]:

- $\mathsf{pre\text{-}order}(u)/\mathsf{post\text{-}order}(u)$ = the pre/post-order rank of node u,
- $\mathsf{parent}(u)$ = the parent of node u,
- $\mathsf{nodeDepth}(u)$ = the number of edges on the path from root to u,
- $\mathsf{child}(u, q)$ = the qth leftmost child of node u,
- $\mathsf{lca}(u, v)$ = the lowest common ancestor (LCA) of two nodes u and v,
- $\mathsf{L}(u)/\mathsf{R}(u)$ = the leftmost/rightmost leaf of the subtree rooted at u, and
- $\mathsf{levelAncestor}(u, D)$ = the ancestor of u such that $\mathsf{nodeDepth}(u) = D$.

Additionally, we can find the pre-order rank of the ith leftmost leaf in $O(1)$ time. Moving forward, we use ℓ_i to denote the ith leftmost leaf in sST.

4.3 ZeroDepth and ZeroNode

For a node u, $\mathsf{zeroDepth}(u)$ is the number of 0's in $\mathsf{str}(u)$. For a leaf ℓ_i, $\mathsf{sBWT}[i] \in [1, \sigma_p]$ (resp. $\mathsf{sBWT}[i] < 0$), we define $\mathsf{zeroNode}(\ell_i)$ to be the locus (if exists) of $\mathsf{str}(\ell_i)[1, f_i^+]$ (resp. the locus of $\mathsf{str}(\ell_i)[1, f_i^-]$). Equivalently, $\mathsf{zeroNode}(\ell_i)$ is the highest node (if exists) z on the root to ℓ_i path such that $\mathsf{zeroDepth}(w) \geq |\mathsf{sBWT}[i]|$. Moving forward, whenever we refer to $\mathsf{zeroNode}(\ell_i)$, we assume $\mathsf{sBWT}[i] \in [-\sigma_p, \sigma_p]$. We present the following lemma.

▶ **Lemma 10.** *By using the Wavelet Tree over* sBWT *and an additional $O(n)$-bit data structure, we can find* $\mathsf{zeroNode}(\ell_i)$ *in $O(\log \sigma)$ time.*

Proof. We begin with the following definitions. For any node x on the root to ℓ_i path π, define $\alpha(x)$ = the number of leaves ℓ_j, $j \in [\mathsf{L}(x), \mathsf{R}(x)]$ such that $L[j] \in \Sigma_p$ and $f_j \leq |\mathsf{str}(x)|$, and $\beta(x) = \mathsf{count}(\mathsf{L}(x), \mathsf{R}(x), -c, c)$, where $c = |\mathsf{sBWT}[i]|$. Consider a node u_k on π. Now, $\mathsf{zeroNode}(\ell_i)$ is below u_k iff $\beta(u_k) > \alpha(u_k)$. Therefore, $\mathsf{zeroNode}(\ell_i)$ is the shallowest node $u_{k'}$ on this path that satisfies $\beta(u_{k'}) \leq \alpha(u_{k'})$. Equipped with this knowledge, now we can binary search on π (using $\mathsf{nodeDepth}$ and $\mathsf{levelAncestor}$ operations) to find the exact location. The first question is to compute $\alpha(x)$, which is handled by Lemma 11. A normal binary search will have to consider n nodes on the path in the worst case. Lemma 12 shows how to reduce this to $\lceil \log \sigma \rceil$. Thus, the binary search has at most $\lceil \log \log \sigma \rceil$ steps, and the total time is $\log \log \sigma \times \lceil \frac{\log \sigma}{\log \log n} \rceil = O(\log \sigma)$, as required. ◀

The following are our helper lemmas for proving Lemma 10. The proofs are similar to those of Lemmas 4 and 5 in [9] respectively. We omit the proofs due to space limitations.

▶ **Lemma 11.** *We can compute $\alpha(x)$ in $O(1)$ time using an $O(n)$-bit data structure.*

▶ **Lemma 12.** *By using the Wavelet Tree over* sBWT *and an additional $O(n)$-bit data structure, in $O(\log \sigma)$ time, we can find an ancestor w_i of ℓ_i such that* $\mathsf{zeroDepth}(w_i) < |\mathsf{sBWT}[i]|$ *and w_i is at most $\lceil \log \sigma \rceil$ nodes above* $\mathsf{zeroNode}(\ell_i)$.

4.4 Additional Components

Define f_j to be the smaller of f_j^+ and f_j^-, where $L[j] \in \Sigma_p$. Let $\mathsf{leafLeadChar}(j)$ be a boolean variable, which is 0 iff $f_j = (|\mathsf{str}(v)| + 1)$, where $v = \mathsf{parent}(\mathsf{zeroNode}(j))$.

For a node u, $\mathsf{pCount}(v)$ is the rightmost child w of v such that the first character on the edge (v, w) is a p-character. Since $\sum_v \mathsf{pCount}(v) = O(n)$, we can compute $\mathsf{pCount}(v)$

in $O(1)$ by using an $O(n)$-bit data structure. Let $\mathsf{fCount}^+(x)$ (resp. $\mathsf{fCount}^-(x)$) be the number of leaves ℓ_j in x's subtree, such that $\mathsf{sBWT}[j] \in [1, \sigma_p]$ (resp. $\mathsf{sBWT}[j] \in [-\sigma_p, -1]$) and $|\mathsf{str}(y)| + 2 \leq f_j \leq |\mathsf{str}(x)| + 1$, where $y = \mathsf{parent}(x)$. Additionally, for any leaf ℓ_j, assign $\mathsf{fCount}^+(\ell_j) = 1$ (resp. $\mathsf{fCount}^-(\ell_j) = 1$) if $f_j > |\mathsf{str}(\ell_j)|$ and $\mathsf{sBWT}[j] \in [1, \sigma_p]$ (resp. $\mathsf{sBWT}[j] < 0$). Let $\mathsf{fSum}^+(x)$ be the sum of $\mathsf{fCount}^+(y)$ of all nodes y which come before x in pre-order and are not ancestors of x. Let $\overleftarrow{\mathsf{fSum}}^-(x)$ be the sum of $\mathsf{fCount}^-(y)$ of all nodes y which come after $\mathsf{R}(x)$ in pre-order. Let $\mathsf{fAncestor}^+(x)$ be the number of leaves ℓ_j such that $\mathsf{pre\text{-}order}(\ell_j) < \mathsf{pre\text{-}order}(x)$, $f_j^+ = |\mathsf{str}(\mathsf{lca}(\ell_j, x))| + 1$, $\mathsf{sBWT}[j] \in [1, \sigma_p]$, and the first character on the path from $\mathsf{lca}(\ell_j, x)$ to x is an s-character.

We present the following important lemma (proof is similar to that of Lemma 3 in [9] and is omitted due space restriction).

▶ **Lemma 13.** *By using an $O(n)$-bit data structure, for any node x, we can compute the following in $O(1)$ time: $\mathsf{fSum}^+(x)$, $\overleftarrow{\mathsf{fSum}}^-(x)$, and $\mathsf{fAncestor}^+(x)$.*

4.5 Computing $\mathsf{sLF}(i)$ when $\mathsf{sBWT}[i] \in [\sigma_p + 1, \sigma]$

Using Lemma 9, $\mathsf{sLF}(i) > \mathsf{sLF}(j)$ iff either $j \in [1, n]$ and $\mathsf{sBWT}[j] < \mathsf{sBWT}[i]$, or $j \in [1, i-1]$ and $\mathsf{sBWT}[i] = \mathsf{sBWT}[j]$. Then,

$$\mathsf{sLF}(i) = 1 + \mathsf{count}(1, n, 1, \mathsf{sBWT}[i] - 1) + \mathsf{count}(1, i-1, \mathsf{sBWT}[i], \mathsf{sBWT}[i])$$

4.6 Computing $\mathsf{sLF}(i)$ when $\mathsf{sBWT}[i] \in [1, \sigma_p]$

We first assume that $\mathsf{zeroNode}(\ell_i)$ is defined, i.e., $f_i \leq |\mathsf{str}(\ell_i)|$. This can be easily checked in $O(1)$ time by maintaining a bit-vector. First find $z = \mathsf{zeroNode}(\ell_i)$ and locate the node $v = \mathsf{parent}(z)$. Depending on whether $\mathsf{leafLeadChar}(i) = 0$, or not, we find the ranges $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and if required $\mathcal{S}_4$ and $\mathcal{S}_5$, as illustrated in Figure 1.

Sub-case 1 ($f_i = |\mathsf{str}(v)| + 1$). Let w be the parent of v. We partition the leaves into 4 sets: (a) $\mathcal{S}_1$: leaves to the left of v's subtree, (b) $\mathcal{S}_2$: leaves in z's subtree, (c) $\mathcal{S}_3$: leaves to the right of v's subtree, and (d) $\mathcal{S}_4$ (resp. $\mathcal{S}_5$): leaves in v's subtree, and to the left (resp. right) of that of z's subtree. In case, v is the root node r, we take $w = r$; consequently, $\mathcal{S}_1 = \mathcal{S}_3 = \emptyset$.

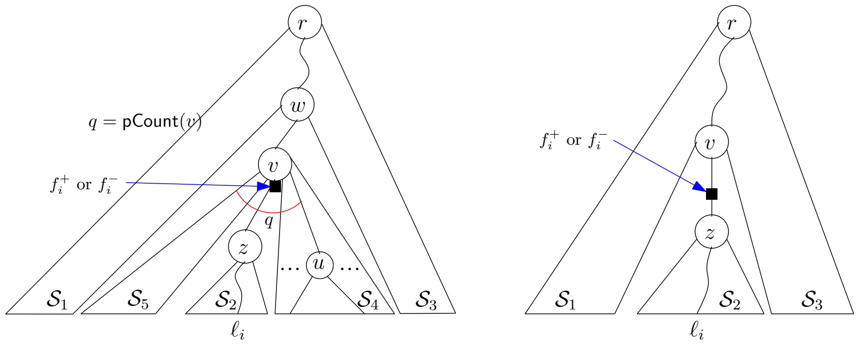
Sub-case 2 ($f_i > |\mathsf{str}(v)| + 1$). We partition the leaves into 3 sets: (a) $\mathcal{S}_1$ (resp. $\mathcal{S}_3$): leaves to the left (resp. right) of z's subtree. (b) $\mathcal{S}_2$: leaves in z's subtree.

Let $c = \mathsf{sBWT}[i]$. Define N_k to be the number of leaves ℓ_j in $\mathcal{S}_k$ such that $\mathsf{sLF}(j) \leq \mathsf{sLF}(i)$. Then, $\mathsf{sLF}(i) = N_1 + N_2 + N_3 + N_4 + N_5$ is computed as follows.

Computing N_1. For any $\ell_j \in \mathcal{S}_1$, $\mathsf{sLF}(j) < \mathsf{sLF}(i)$ iff one of the following holds: (1) $\mathsf{sBWT}[j] \in [1, \sigma_p]$ and $f_j^+ > 1 + |\mathsf{str}(\mathsf{lca}(\ell_i, \ell_j))|$, or (2) $\mathsf{sBWT}[j] \in [1, \sigma_p]$, $f_j^+ = 1 + |\mathsf{str}(\mathsf{lca}(\ell_i, \ell_j))|$, and the leading character on the path from $\mathsf{lca}(\ell_i, \ell_j)$ to ℓ_i is an s-character, or (3) $\mathsf{sBWT}[j] < 0$. Then,

$$N_1 = \begin{cases} \mathsf{fSum}^+(v) + \mathsf{fAncestor}^+(v) + \mathsf{count}(1, \mathsf{L}(v) - 1, -\sigma_p, -1), & \text{if } \mathsf{leafLeadChar}(i) = 0 \\ \mathsf{fSum}^+(z) + \mathsf{fAncestor}^+(z) + \mathsf{count}(1, \mathsf{L}(z) - 1, -\sigma_p, -1), & \text{otherwise} \end{cases}$$

Computing N_2. If $c > 0$, then for any leaf $\ell_j \in \mathcal{S}_2$, $\mathsf{sLF}(j) \leq \mathsf{sLF}(i)$ iff one of the following holds: (1) either $\mathsf{sBWT}[j] > c$ or $\mathsf{sBWT}[j] = c$ and $j \leq i$, (2) $\mathsf{sBWT}[j] < 0$. If $c < 0$, then



(a) f_i^+ or $f_i^- = |\mathsf{str}(v)| + 1$ **(b)** f_i^+ or $f_i^- > |\mathsf{str}(v)| + 1$

Figure 1 Illustration of various ranges when $T_{\mathsf{sSA}[i]}$ is preceded by a p-character

for any leaf $\ell_j \in \mathcal{S}_2$, $\mathsf{sLF}(j) \le \mathsf{sLF}(i)$ iff one of the following holds: (1) $-1 \ge \mathsf{sBWT}[j] > c$, (2) $\mathsf{sBWT}[j] = c$ and $j \le i$. Therefore,

$$N_2 = \begin{cases} \mathsf{count}(\mathsf{L}(z), \mathsf{R}(z), c+1, \sigma_p) + \mathsf{count}(\mathsf{L}(z), i, c, c) & \\ \qquad\qquad +\mathsf{count}(\mathsf{L}(z), \mathsf{R}(z), -\sigma_p, -1), & \text{if } c \in [1, \sigma_p] \\ \mathsf{count}(\mathsf{L}(z), \mathsf{R}(z), c+1, -1) + \mathsf{count}(\mathsf{L}(z), i, c, c), & \text{if } c < 0 \end{cases}$$

Computing N_3. For any leaf $\ell_j \in \mathcal{S}_3$, $\mathsf{sLF}(j) < \mathsf{sLF}(i)$ iff $\mathsf{sBWT}[j] < 0$ and $f_j^- \le 1 + |\mathsf{str}(\mathsf{lca}(z, \ell_j))|$. Therefore,

$$N_3 = \begin{cases} \mathsf{count}(\mathsf{R}(v)+1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(v), & \text{if } \mathsf{leafLeadChar}(i) = 0 \\ \mathsf{count}(\mathsf{R}(z)+1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(z), & \text{otherwise} \end{cases}$$

Computing N_4. Let u be the $\mathsf{pCount}(v)$th child of v. For any leaf $\ell_j \in \mathcal{S}_4$ such that $\mathsf{sBWT}[j] \in [1, \sigma_p]$, $f_j^+ \neq |\mathsf{str}(v)| + 1$; otherwise, the suffix j should not have deviated from i at the node v. Likewise, if $\mathsf{sBWT}[j] < 0$, then $f_j^- \neq |\mathsf{str}(v)| + 1$.

If $c > 0$, then N_4 is the number of leaves ℓ_j in $\mathcal{S}_4$ such that $j \le \mathsf{R}(u)$ and either (1) $\sigma_p \ge \mathsf{sBWT}[j] \ge c$, or (2) $\mathsf{sBWT}[j] < 0$. If $c < 0$, then N_4 is the number of leaves ℓ_j in $\mathcal{S}_4$ such that $j \le \mathsf{R}(u)$ and $-1 \ge \mathsf{sBWT}[j] > c$. Therefore,

$$N_4 = \begin{cases} \mathsf{count}(\mathsf{R}(z)+1, \mathsf{R}(u), c, \sigma_p) + \mathsf{count}(\mathsf{R}(z)+1, \mathsf{R}(u), -\sigma_p, -1), & \text{if } c \in [1, \sigma_p] \\ \mathsf{count}(\mathsf{R}(z)+1, \mathsf{R}(u), c+1, -1), & \text{if } c < 0 \end{cases}$$

Computing N_5. Note that for any leaf $\ell_j \in \mathcal{S}_5$ such that $\mathsf{sBWT}[j] \in [1, \sigma_p]$, $f_j^+ \neq |\mathsf{str}(v)|+1$; otherwise, the suffix j should not have deviated from i at the node v. Likewise, if $\mathsf{sBWT}[j] < 0$, then $f_j^- \neq |\mathsf{str}(v)| + 1$. Also, the leading character of the path from v to ℓ_j is negative.

If $c > 0$, then N_5 is the number of leaves ℓ_j in $\mathcal{S}_5$ that satisfies one of the following: (1) $\sigma_p \ge \mathsf{sBWT}[j] \ge c$, or (2) $\mathsf{sBWT}[j] < 0$. If $c < 0$, then N_5 is the number of leaves ℓ_j in $\mathcal{S}_5$ such that $-1 \ge \mathsf{sBWT}[j] > c$. Therefore,

$$N_5 = \begin{cases} \mathsf{count}(\mathsf{L}(v), \mathsf{L}(z)-1, c, \sigma_p) + \mathsf{count}(\mathsf{L}(v), \mathsf{L}(z)-1, -\sigma_p, -1), & \text{if } c \in [1, \sigma_p] \\ \mathsf{count}(\mathsf{L}(v), \mathsf{L}(z)-1, c+1, -1), & \text{if } c < 0 \end{cases}$$

Algorithm 1 computes $\mathsf{sLF}(i)$

```
$c \leftarrow \mathsf{sBWT}[i]$
if $(c > \sigma_p)$ then
    $\mathsf{sLF}(i) \leftarrow 1 + \mathsf{count}(1, n, -\sigma_p, c-1) + \mathsf{count}(1, i-1, c, c)$
else
    $z \leftarrow \mathsf{zeroNode}(\ell_i)$, $L_z \leftarrow \mathsf{L}(z)$, $R_z \leftarrow \mathsf{R}(z)$
    if $(\mathsf{leafLeadChar}(i)$ is $0)$ then
        $v \leftarrow \mathsf{parent}(z)$, $L_v \leftarrow \mathsf{L}(v)$, $R_v \leftarrow \mathsf{R}(v)$
        $u \leftarrow \mathsf{child}(v, \mathsf{pCount}(v))$, $R_u \leftarrow \mathsf{R}(u)$
        $N_1 \leftarrow \mathsf{fSum}^+(v) + \mathsf{count}(1, L_v - 1, -\sigma_p, -1)$
        $N_3 \leftarrow \mathsf{count}(R_v + 1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(v)$
        if $(c > 0)$ then
            $N_4 \leftarrow \mathsf{count}(R_z + 1, R_u, c, \sigma_p) + \mathsf{count}(R_z + 1, R_u, -\sigma_p, -1)$
            $N_5 \leftarrow \mathsf{count}(L_v, L_z - 1, c, \sigma_p) + \mathsf{count}(L_v, L_z - 1, -\sigma_p, -1)$
        else
            $N_4 \leftarrow \mathsf{count}(R_z + 1, R_u, c + 1, -1)$
            $N_5 \leftarrow \mathsf{count}(L_v, L_z - 1, c + 1, -1)$
    else
        $N_1 \leftarrow \mathsf{fSum}^+(z) + \mathsf{count}(1, L_z - 1, -\sigma_p, -1)$
        $N_3 \leftarrow \mathsf{count}(R_z + 1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(z)$
    if $(c > 0)$ then
        $N_2 \leftarrow \mathsf{count}(L_z, R_z, c + 1, \sigma_p) + \mathsf{count}(L_z, i, c, c) + \mathsf{count}(L_z, R_z, -\sigma_p, -1)$
    else
        $N_2 \leftarrow \mathsf{count}(L_z, R_z, c + 1, -1) + \mathsf{count}(L_z, i, c, c)$
    $\mathsf{sLF}(i) \leftarrow N_1 + N_2 + N_3 + N_4 + N_5$
```

Now, we arrive at the scenario when $\mathsf{zeroNode}(\ell_i)$ is not defined, i.e., $f_i > |\mathsf{str}(\ell_i)|$. Following the arguments in this section, it is easy to arrive at the following:

$$\begin{aligned}\mathsf{sLF}(i) = 1 + \mathsf{fSum}^+(\ell_i) + \mathsf{fAncestor}^+(\ell_i) + \mathsf{count}(1, i-1, -\sigma_p, -1)\\ + \mathsf{count}(i+1, n, -\sigma_p, -1) - \mathsf{fSum}^-(\ell_i), \text{ when } f_i > |\mathsf{str}(\ell_i)|\end{aligned}$$

Summarizing the discussions in this section, we have proved the following.

▶ **Lemma 14.** *We can compute* $\mathsf{sLF}(i)$ *in* $O(\log \sigma)$ *time using the Wavelet Tree over* sBWT *and an additional* $O(n)$*-bit data structure.*

5 Finding Suffix Range via Backward Search

We use an adaptation of the backward search algorithm in the FM-index [6]. In particular, given a proper suffix Q of P, assume that we know the suffix range $[sp_1, ep_1]$ of $\Phi(Q)$. Our task is to find the suffix range $[sp_2, ep_2]$ of $\Phi(c \circ Q)$, where c is the character previous to Q in P. If c is a static character, then

$$\begin{aligned} sp_2 &= 1 + \mathsf{count}(1, n, -\sigma_p, c-1) + \mathsf{count}(1, sp_1 - 1, c, c)\\ ep_2 &= \mathsf{count}(1, n, -\sigma_p, c-1) + \mathsf{count}(1, ep_1, c, c)\end{aligned}$$

Now, we consider the scenario when c is a p-character.

5.1 Case 1 (Neither c nor its complement appears in Q)

Let d be the number of distinct p-characters in Q, which can be computed in $O(1)$ time after pre-processing P in $O(|P|\log\sigma)$ time. Note that $\mathsf{sLF}(i) \in [sp_2, ep_2]$ iff $i \in [sp_1, ep_1]$, $\mathsf{sBWT}[i] \in [-\sigma_p, \sigma_p]$ and $f_i > |Q|$. Then,

$$(ep_2 - sp_2 + 1) = \mathsf{count}(sp_1, ep_1, d+1, \sigma_p) + \mathsf{count}(sp_1, ep_1, -\sigma_p, -d-1)$$

Let $u = \mathsf{lca}(\ell_{sp_1}, \ell_{ep_1})$. For any i, $\mathsf{sLF}(i) < sp_2$ iff (1) $i < sp_1$, $\mathsf{sBWT}[i] \in [1, \sigma_p]$ and $f_i^+ > 1 + |\mathsf{str}(\mathsf{lca}(u, \ell_i))|$, or (2) $i < sp_1$, $\mathsf{sBWT}[i] \in [1, \sigma_p]$, $f_i^+ = 1 + |\mathsf{str}(\mathsf{lca}(u, \ell_i))|$, and the leading character on the path from $\mathsf{lca}(u, \ell_i)$ to u is an s-character, or (3) $i \in [sp_1, ep_1]$, $\mathsf{sBWT}[i] < 0$ and $f_i^- \leq |Q|$, or (4) $i < sp_1$ and $\mathsf{sBWT}[i] < 0$, or (5) $i > ep_1$, $\mathsf{sBWT}[i] < 0$ and $f_i^- \leq 1 + |\mathsf{str}(\mathsf{lca}(u, \ell_i))|$. Therefore,

$$\begin{aligned} sp_2 = 1 + \mathsf{fSum}^+(u) + \mathsf{fAncestor}^+(u) + \mathsf{count}(sp_1, ep_1, -d, -1) \\ + \mathsf{count}(1, sp_1 - 1, -\sigma_p, -1) + \mathsf{count}(ep_1 + 1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(u) \end{aligned}$$

5.2 Case 2 (c or its complement appears in Q)

Assume that the number of characters until the first occurrence of c (resp. c's complement) in Q is f^+ (resp. f^-). If f^+ or f^- does not exist, we take it to be $|Q| + 1$. Let d^+ and d^- be respectively the number of distinct p-characters in $Q[1, f^+]$ and $Q[1, f^-]$ respectively. After an initial $O(|P|\log\sigma)$ time pre-processing, d^+ and d^- can retrieved in $O(1)$ time.

Case when $f^+ < f^-$: Note that $\mathsf{sLF}(i) \in [sp_2, ep_2]$ iff $i \in [sp_1, ep_1]$, $\mathsf{sBWT}[i] \in [1, \sigma_p]$ and $f_i^+ = f^+$. Consider any $i, j \in [sp_1, ep_1]$ such that $i < j$, both $\mathsf{sLF}(i), \mathsf{sLF}(j) \in [sp_2, ep_2]$, and both $\mathsf{sBWT}[i], \mathsf{sBWT}[j] \in [1, \sigma_p]$. Now, $f_i^+ = f_j^+ = f^+$, and $\mathsf{sLF}(i) < \mathsf{sLF}(j)$. Therefore,

$$\begin{aligned} (ep_2 - sp_2 + 1) &= \mathsf{count}(sp_1, ep_1, d^+, d^+), \text{ and} \\ sp_2 &= \mathsf{sLF}(\min\{j \mid j \in [sp_1, ep_1] \text{ and } \mathsf{sBWT}[j] = d^+\}) \\ &= \mathsf{sLF}(\mathsf{select}(1 + \mathsf{rank}(sp_1 - 1, d^+), d^+)) \end{aligned}$$

Case when $f^+ > f^-$: Based on the above arguments, we can derive the following.

$$\begin{aligned} (ep_2 - sp_2 + 1) &= \mathsf{count}(sp_1, ep_1, -d^-, -d^-), \text{ and} \\ sp_2 &= \mathsf{sLF}(\min\{j \mid j \in [sp_1, ep_1] \text{ and } \mathsf{sBWT}[j] = -d^-\}) \\ &= \mathsf{sLF}(\mathsf{select}(1 + \mathsf{rank}(sp_1 - 1, d^-), d^-)) \end{aligned}$$

Thus, the suffix range of $\Phi(P)$ is computed in $O(|P|\log\sigma)$ time. Applying Lemmas 8 and 14, we arrive at Theorem 3.

References

1 Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 71–80. ACM, 1993. doi:10.1145/167088.167115.

2 Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.

3 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation (now part of Hewlett-Packard, Palo Alto, CA), 1994.

4 Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM J. Comput.*, 33(1):26–42, 2003. doi:10.1137/S0097539701424465.

5 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, volume 8214 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2013. doi:10.1007/978-3-319-02432-5_13.

6 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

7 Johannes Fischer. Wee LCP. *Inf. Process. Lett.*, 110(8-9):317–320, 2010. doi:10.1016/j.ipl.2010.02.010.

8 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007. doi:10.1007/978-3-540-74450-4_41.

9 Arnab Ganguly, Rahul Shah, and Sharma V Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 397–407. Society for Industrial and Applied Mathematics, 2017.

10 Raffaele Giancarlo. The suffix of a square matrix, with applications. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 402–411. ACM/SIAM, 1993.

11 Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and S. Srinivasa Rao. On the size of succinct indices. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2007. doi:10.1007/978-3-540-75520-3_34.

12 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. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

13 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

14 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.

15 Dong Kyue Kim, Yoo Ah Kim, and Kunsoo Park. Generalizations of suffix arrays to multi-dimensional matrices. *Theor. Comput. Sci.*, 302(1-3):223–238, 2003. doi:10.1016/S0304-3975(02)00828-9.

16 Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52:1–52:47, 2013. doi:10.1145/2535933.

17 Gonzalo Navarro. Wavelet trees for all. *J. Discrete Algorithms*, 25:2–20, 2014. doi:10.1016/j.jda.2013.07.004.

18 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), 2007. doi:10.1145/1216370.1216372.

19 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.

20 Luís M. S. Russo, Gonzalo Navarro, and Arlindo L. Oliveira. Fully compressed suffix trees. *ACM Transactions on Algorithms*, 7(4):53, 2011. doi:10.1145/2000807.2000821.

21 Tetsuo Shibuya. Generalization of a suffix tree for RNA structural pattern matching. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 393–406. Springer, 2000. doi:10.1007/3-540-44985-X_34.

22 Tetsuo Shibuya. Geometric suffix tree: Indexing protein 3-d structures. *J. ACM*, 57(3):15:1–15:17, 2010. doi:10.1145/1706591.1706595.

23 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.

On Structural Parameterizations of the Edge Disjoint Paths Problem[*]

Robert Ganian[1], Sebastian Ordyniak[2], and Ramanujan Sridharan[3]

1 Algorithms and Complexity Group, TU Wien, Vienna, Austria
ganian@ac.tuwien.ac.at
2 Algorithms and Complexity Group, TU Wien, Vienna, Austria
ordyniak@ac.tuwien.ac.at
3 Algorithms and Complexity Group, TU Wien, Vienna, Austria
ramanujan@ac.tuwien.ac.at

Abstract

In this paper we revisit the classical Edge Disjoint Paths (EDP) problem, where one is given an undirected graph G and a set of terminal pairs P and asks whether G contains a set of pairwise edge-disjoint paths connecting every terminal pair in P. Our focus lies on structural parameterizations for the problem that allow for efficient (polynomial-time or fpt) algorithms. As our first result, we answer an open question stated in Fleszar, Mnich, and Spoerhase (2016), by showing that the problem can be solved in polynomial time if the input graph has a feedback vertex set of size one. We also show that EDP parameterized by the treewidth and the maximum degree of the input graph is fixed-parameter tractable.

Having developed two novel algorithms for EDP using structural restrictions on the input graph, we then turn our attention towards the augmented graph, i.e., the graph obtained from the input graph after adding one edge between every terminal pair. In constrast to the input graph, where EDP is known to remain NP-hard even for treewidth two, a result by Zhou et al. (2000) shows that EDP can be solved in non-uniform polynomial time if the augmented graph has constant treewidth; we note that the possible improvement of this result to an fpt-algorithm has remained open since then. We show that this is highly unlikely by establishing the W[1]-hardness of the problem parameterized by the treewidth (and even feedback vertex set) of the augmented graph. Finally, we develop an fpt-algorithm for EDP by exploiting a novel structural parameter of the augmented graph.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.2.1 Combinatorics

Keywords and phrases edge disjoint path problem, feedback vertex set, treewidth, fracture number, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.36

1 Introduction

The Edge Disjoint Paths (EDP) and Node Disjoint Paths (NDP) are fundamental routing graph problems. In the EDP (NDP) problem the input is a graph G, and a set P containing k pairs of vertices and the objective is to decide whether there is a set of k pairwise edge disjoint (respectively vertex disjoint) paths connecting each pair in P. These problems

[*] The authors acknowledge support by the Austrian Science Fund (FWF, project P26696). Robert Ganian is also affiliated with FI MU, Czech Republic.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 36; pp. 36:1–36:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and their optimization versions – MaxEDP and MaxNDP – have been at the center of numerous results in structural graph theory, approximation algorithms, and parameterized algorithms [20, 15, 4, 17, 9, 22, 19, 12, 10].

When k is a part of the input, both EDP and NDP are known to be NP-complete [14]. Robertson and Seymour's seminal work in the Graph Minors project [20] provides an $\mathcal{O}(n^3)$ time algorithm for both problems for every fixed value of k. In the realm of *Parameterized Complexity*, their result can be interpreted as *fpt-algorithms* for EDP and NDP parameterized by k. Here, one considers problems associated with a certain numerical parameter k and the central question is whether the problem can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ where f is a computable function and n the input size; algorithms with running time of this form are called fpt-algorithms [11, 7, 5].

While the aforementioned research considered the number of paths to be the parameter, another line of research investigates the effect of *structural parameters* of the input graphs on the complexity of these problems. Fleszar, Mnich, and Spoerhase [10] initiated the study of NDP and EDP parameterized by the feedback vertex set number (the size of the smallest feedback vertex set) of the input graph and showed that EDP remains NP-hard even on graphs with feedback vertex set number two. Since EDP is known to be polynomial time solvable on forests [12], this left only the case of feedback vertex set number one open, which they conjectured to be polynomial time solvable. Our first result is a positive resolution of their conjecture.

▶ **Theorem 1.** EDP *can be solved in time* $\mathcal{O}(|P||V(G)|^{\frac{5}{2}})$ *on graphs with feedback vertex set number one.*

A key observation behind the polynomial-time algorithm is that an EDP instance with a feedback vertex set $\{x\}$ is a Yes-instance if and only if, for every tree T of $G - \{x\}$, it is possible to connect all terminal pairs in T either to each other or to x through pairwise edge disjoint paths in T. The main ingredient of the algorithm is then a dynamic programming procedure that determines whether such a set exists for a tree T of $G - \{x\}$.

Continuing to explore structural parameterizations for the input graph of an EDP instance, we then show that even though EDP is NP-complete when the *input graph* has treewidth two, it becomes fixed-parameter tractable if we additionally parameterize by the maximum degree.

▶ **Theorem 2.** EDP *is fixed-parameter tractable parameterized by the treewidth and the maximum degree of the input graph.*

Having explored the algorithmic applications of structural restrictions on the input graph for EDP, we then turn our attention towards similar restrictions on the *augmented graph* of an EDP instance (G, P), i.e., the graph obtained from G after adding an edge between every pair of terminals in P. Whereas EDP is NP-complete even if the input graph has treewidth at most two [19], it can be solved in non-uniform polynomial time if the treewidth of the augmented graph is bounded [22]. It has remained open whether EDP is fixed-parameter tractable parameterized by the treewidth of the augmented graph; interestingly, this has turned out to be the case for the strongly related multicut problems [13]. Surprisingly, we show that this is not the case for EDP, by establishing the W[1]-hardness of the problem parameterized by not only the treewidth but also by the feedback vertex set number of the augmented graph.

▶ **Theorem 3.** *EDP is* W[1]*-hard parameterized by the feedback vertex set number of the augmented graph.*

Motivated by this strong negative result, our next aim was to find natural structural parameterizations for the augmented graph of an EDP instance for which the problem becomes fixed-parameter tractable. Towards this aim, we introduce the *fracture number*, which informally corresponds to the size of a minimum vertex set S such that the size of every component in the graph minus S is small (has size at most $|S|$). We show that EDP is fixed-parameter tractable parameterized by this new parameter.

▶ **Theorem 4.** EDP *is fixed-parameter tractable parameterized by the fracture number of the augmented graph.*

We note that the reduction in [10, Theorem 6] excludes the applicability of the fracture number of the *input graph* by showing that EDP is NP-complete even for instances with fracture number at most three. Finally, we complement Theorem 4 by showing that bounding the number of terminal pairs in each component instead of the its size is not sufficient to obtain fixed-parameter tractability.

2 Preliminaries

2.1 Basic Notation

We use standard terminology for graph theory, see for instance [6]. Given a graph G, we let $V(G)$ denote its vertex set, $E(G)$ its edge set and by $V(E')$ the set of vertices incident with the edges in E', where $E' \subseteq E(G)$. The (open) neighborhood of a vertex $x \in V(G)$ is the set $\{y \in V(G) : xy \in E(G)\}$ and is denoted by $N_G(x)$. For a vertex subset X, the neighborhood of X is defined as $\bigcup_{x \in X} N_G(x) \setminus X$ and denoted by $N_G(X)$. For a vertex set A, we use $G - A$ to denote the graph obtained from G by deleting all vertices in A, and we use $G[A]$ to denote the *subgraph induced on* A, i.e., $G - (V(G) \setminus A)$. A *forest* is a graph without cycles, and a vertex set X is a *feedback vertex set* (*FVS*) if $G - X$ is a forest. We use $[i]$ to denote the set $\{0, 1, \dots, i\}$. The *feedback vertex set number* of a graph G, denoted by $\mathbf{fvs}(G)$, is the smallest integer k such that G has a feedback vertex set of size k.

2.2 Edge Disjoint Path Problem

In the Edge Disjoint Paths (EDP) problem, one is given an undirected graph G a set P of terminal pairs (i.e., subsets of $V(G)$ of size two) and the question is whether there is there a set of pairwise edge disjoint paths connecting every set of terminal pairs in P. Let (G, P) be an instance of EDP; for brevity, we will sometimes denote a terminal pair $\{s, t\} \in P$ simply as st. For a subgraph H of G, we denote by $P(H)$ the subset of P containing all sets that have a non-empty intersection with $V(H)$ and for $P' \subseteq P$, we denote by $\widetilde{P'}$ the set $\bigcup_{p \in P'} p$. We will assume that, w.l.o.g., each vertex $v \in V(G)$ occurs in at most one terminal pair, each vertex in a terminal pair has degree 1 in G, and each terminal pair is not adjacent to each other; indeed, for any instance without these properties, we can add a new leaf vertex for terminal, attach it to the original terminal, and replace the original terminal with the leaf vertex [22].

▶ **Definition 5** ([22]). The *augmented graph* of (G, P) is the graph G^P obtained from G by adding edges between each terminal pair, i.e., $G^P = (V(G), E(G) \cup P)$.

2.3 Parameterized Complexity

The parameterized complexity paradigm allows a finer analysis of the complexity of problems by associating each problem instance L with a numerical parameter k; the pair (L, k) is then an instance of a parameterized problem. A parameterized problem is *fixed-parameter tractable* (FPT in short) if a given instance (L, k) can be solved in time $\mathcal{O}(f(k) \cdot |L|^{\mathcal{O}(1)}$ where f is an arbitrary computable function; we call algorithms running in this time *fpt-algorithms*. The complexity class W[1] is the analog of NP for parameterized complexity; under established complexity assumptions, problems that are hard for W[1] do not admit fpt-algorithms.

We refer the reader to the respective monographs [11, 7, 5] for an in-depth introduction to parameterized complexity.

2.4 Treewidth

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(T, \{B_t : t \in V(T)\})$ where $B_t \subseteq V$ for every $t \in V(T)$ and T is a tree such that:

1. for each edge $(u, v) \in E$, there is a $t \in V(T)$ such that $\{u, v\} \subseteq B_t$, and
2. for each vertex $v \in V$, $T[\{\, t \in V(T) \mid v \in B_t \,\}]$ is a non-empty (connected) tree.

The *width* of a tree-decomposition is $\max_{t \in V(T)} |B_t| - 1$. The *treewidth* [16] of G is the minimum width taken over all tree-decompositions of G and it is denoted by $\mathbf{tw}(G)$. We call the elements of $V(T)$ *nodes* and B_t *bags*.

While it is possible to compute the treewidth exactly using an fpt-algorithm [1], the asymptotically best running time is achieved by using the recent state-of-the-art 5-approximation algorithm of Bodlaender et al. [2].

▶ **Fact 6** ([2]). *There exists an algorithm which, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k)} \cdot n$ either outputs a tree-decomposition of G of width at most $5k + 4$ and $\mathcal{O}(n)$ nodes, or correctly determines that $\mathbf{tw}(G) > k$.*

A tree-decomposition $(T, B_t : t \in V(T))$ of a graph G is *nice* if the following conditions hold: (1) T is rooted at a node r such that $|B_r| = \emptyset$, (2) every node of T has at most two children, if a node t of T has two children t_1 and t_2, then $B_t = B_{t_1} = B_{t_2}$; in that case we call t a *join node*, (3) if a node t of T has exactly one child t', then exactly one of the following holds: (3A) $B_t = B_{t'} \cup \{v\}$, in which case we call t an *introduce node* or (3B) $B_t = B_{t'} \setminus \{v\}$ in which case we call t a *forget node*, and (4) if a node t of T is a leaf, then $|B_t| = 1$; we call these *leaf nodes*.

The main advantage of nice tree-decompositions is that they allow the design of much more transparent dynamic programming algorithms, since one only needs to deal with four specific types of nodes. It is well known (and easy to see) that for every fixed k, given a tree-decomposition of a graph $G = (V, E)$ of width at most k and with $\mathcal{O}(|V|)$ nodes, one can construct in linear time a nice tree-decomposition of G with $\mathcal{O}(|V|)$ nodes and width at most k [3]. Given a node t in T, we let Y_t be the set of all vertices contained in the bags of the subtree rooted at t, i.e., $Y_t = B_t \cup \bigcup_{p \text{ is separated from the root by } t} B_p$.

3 Closing the Gap on Graphs of Feedback Vertex Number One

In this section we develop a polynomial-time algorithm for EDP restricted to graphs with feedback vertex set number one. We refer to this particular variant as Simple Edge Disjoint Paths (SEDP): given an EDP instance (G, P) and a FVS $X = \{x\}$, solve (G, P).

Additionally to our standard assumptions about EDP (given in Subsection 2.2), we will assume that: (1) every neighbor of x in G is a leaf in $G - X$, (2) x is not a terminal, i.e., $x \notin \widetilde{P}$, and (3) every tree T in $G - X$ is rooted in a vertex r that is not a terminal. Property

(1) can be ensured by an additional leaf vertex l to any non-leaf neighbor n of x, removing the edge $\{n, x\}$ and adding the edge $\{l, x\}$ to G. Property (2) can be ensured by adding an additional leaf vertex l to x and replacing x with l in P and finally (3) can be ensured by adding a leaf vertex l to any non-terminal vertex r in T and replacing r with l in P.

A key observation behind our algorithm for SEDP is that whether or not an instance $\mathcal{I} = (G, P, X)$ has a solution merely depends on the existence of certain sets of pairwise edge disjoint paths in the trees T in $G - X$. In particular, as we will show in Lemma 8 later on, $\mathcal{I}$ has a solution if and only if every tree T in $G - X$ is $\emptyset$-connected (see Definition 7). The main ingredient of the algorithm is then a bottom-up dynamic programming algorithm that determines whether a tree T in $G-X$ is $\emptyset$-connected. We now define the various connectivity states of subtrees of T that we need to keep track of in the dynamic programming table.

▶ **Definition 7.** Let T be a tree in $G - X$ rooted at r (recall that we can assume that r is not in $\widetilde{P}$), $t \in V(T)$, and let S be a set of pairwise edge disjoint paths in $G[T_t \cup X]$ and $P' \subseteq P(T_t)$, where T_t is the subtree of T rooted at t.

We say that the set S *$\gamma_\emptyset$-connects* P' in $G[T_t \cup X]$ if for every $a \in \widetilde{P'} \cap T_t$, the set S either contains an a-x path disjoint from b, or it contains an a-b path disjoint from x, where $\{a, b\} \in P'$. Moreover, for $\ell \in \{\gamma_X\} \cup P(T_t)$, we say that the set S *ℓ-connects* T_t if S $\gamma_\emptyset$-connects $P(T_t) \setminus \{\ell\}$ and additionally the following conditions hold.

- If $\ell = \gamma_X$ then S also contains a path from t to x.
- If $\ell = p$ for some $p \in P(T_t)$ then:
  - If $p \cap T_t = \{a\}$ then S contains a t-a path disjoint from x.
  - If $p \cap T_t = \{a, b\}$ then S contains a t-a path disjoint from x and a b-x path disjoint from a or S contains a t-b path disjoint from x and an a-x path disjoint from b.

For $\ell \in \{\gamma_\emptyset, \gamma_X\} \cup P(T_t)$, we say that T_t is ℓ-connected if there is a set S which ℓ-connects $P(T_t)$ in $G[T_t \cup X]$.

Informally, a tree T_t is: (1) $\gamma_\emptyset$-connected if all its terminal pairs can be connected in $G[T_t \cup X]$ either to themselves or to x, (2) γ_X-connected if it is $\gamma_\emptyset$-connected and additionally there is a path from its root to x (which can later be used to connect some terminal not in T_t to x via the root of T_t), and (3) γ_p-connected if all but one of its terminals, i.e., one of the terminals in p, can be connected in $G[T_t \cup X]$ either to themselves or to x, and additionally one terminal in p can be connected to the root of T_t (from which it can later be connected to x or the other terminal in p).

▶ **Lemma 8.** *(G, X, P) has a solution if and only if every tree T in $G - X$ is $\gamma_\emptyset$-connected.*

Due to Lemma 8, our algorithm to solve EDP only has to determine whether every tree in $G - X$ is $\gamma_\emptyset$-connected. For a tree T in $G - X$, our algorithm achieves this by computing a set of labels $L(t)$, where $L(t)$ is the set of all labels $\ell \in \{\gamma_\emptyset, \gamma_X\} \cup P(T_t)$ such that T_t is ℓ-connected, via a bottom-up dynamic programming procedure. We begin by arguing that for a leaf vertex l, the value $L(l)$ can be computed in constant time.

▶ **Lemma 9.** *The set $L(l)$ for a leaf vertex l of T can be computed in time $\mathcal{O}(1)$.*

Proof. Since l is a leaf vertex, we conclude that T_l is $\gamma_\emptyset$-connected if and only if either $l \notin \widetilde{P}$ or $l \in \widetilde{P}$ and $(l, x) \in E(G)$. Similarly, T_l is γ_X-connected if and only if $l \notin \widetilde{P}$ and $(l, x) \in E(G)$. Finally, T_l is ℓ-connected for some $\ell \in P(T_l)$ if and only if $l \in \widetilde{P}$. Since all these properties can be checked in constant time, the statement of the lemma follows. ◀

We will next show how to compute $L(t)$ for a non-leaf vertex $t \in V(T)$ with children $t_1, \ldots, t_l$.

▶ **Definition 10.** We define the three sets $V_t^{\neg\gamma_\emptyset} = \{\, t_i \mid \gamma_\emptyset \notin L(t_i) \,\}$, $V_t^{\gamma_X} = \{\, t_i \mid \gamma_X \in L(t_i) \,\}$, and $V_t = \{t_1, \ldots, t_l\} \setminus (V_t^{\neg\gamma_\emptyset} \cup V_t^{\gamma_X})$.

That is, $V_t^{\neg\gamma_\emptyset}$ is the set of those children t_i such that T_i is *not* $\gamma_\emptyset$-connected, $V_t^{\gamma_X}$ is the set of those children t_i such that T_i is γ_X-connected and V_t is the set comprising the remaining children. Observe that $\{V_t, V_t^{\neg\gamma_\emptyset}, V_t^{\gamma_X}\}$ forms a partition of $\{t_1, \ldots, t_l\}$ and moreover $\gamma_\emptyset \in L(t)$ and $\gamma_X \notin L(t)$ for every $t \in V_t$. Let $H(t)$ be the graph with vertex set $V_t \cup V_t^{\neg\gamma_\emptyset}$ having an edge between t_i and t_j (for $i \neq j$) if and only if $L(t_i) \cap L(t_j) \neq \emptyset$ and not both t_i and t_j are in V_t. The following lemma is crucial to our algorithm, because it provides us with a simple characterization of $L(t)$ for a non-leaf vertex $t \in V(T)$.

▶ **Lemma 11.** *Let t be a non-leaf vertex of T with children $t_1, \ldots, t_l$. Then T_t is:*

- *$\gamma_\emptyset$-connected if and only if $L(t') \neq \emptyset$ for every $t' \in \{t_1, \ldots, t_l\}$ and $H(t)$ has a matching M such that $|V^{\neg\gamma_\emptyset} \setminus V(M)| \leq |V_t^{\gamma_X}|$,*
- *γ_X-connected if and only if $L(t') \neq \emptyset$ for every $t' \in \{t_1, \ldots, t_l\}$ and $H(t)$ has a matching M such that $|V^{\neg\gamma_\emptyset} \setminus V(M)| < |V_t^{\gamma_X}|$,*
- *ℓ-connected (for $\ell \in P(T_t)$) if and only if $L(t') \neq \emptyset$ for every $t' \in \{t_1, \ldots, t_l\}$ and there is a t_i with $\ell \in L(t_i)$ such that $H(t) - \{t_i\}$ has a matching M with $|V^{\neg\gamma_\emptyset} \setminus V(M)| \leq |V_t^{\gamma_X}|$.*

The following two lemmas show how the above characterization can be employed to compute $L(t)$ for a non-leaf vertex t of T. Since the matching employed in Lemma 11 needs to maximize the number of vertices covered in $V^{\neg\gamma_\emptyset}$, we first show how such a matching can be computed efficiently.

▶ **Lemma 12.** *There is an algorithm that, given a graph G and a subset S of $V(G)$, computes a matching M maximizing $|V(M) \cap S|$ in time $\mathcal{O}(\sqrt{|V|}|E|)$.*

▶ **Lemma 13.** *Let t be a non-leaf vertex of T with children $t_1, \ldots, t_l$. Then $L(t)$ can be computed from $L(t_1), \ldots, L(t_l)$ in time $\mathcal{O}(|P(T_t)| l^2 \sqrt{l})$.*

We are now ready to put everything together to decide whether a tree T is $\gamma_\emptyset$-connected.

▶ **Lemma 14.** *Let T be a tree in $G - X$. There is an algorithm that decides whether T is $\gamma_\emptyset$-connected in time $\mathcal{O}(|P(T)||V(T)|^{\frac{5}{2}})$.*

Proof. The algorithm computes the set of labels $L(t)$ for every vertex $t \in V(T)$ using a bottom-up dynamic programming approach. Starting from the leaves of T, for which the set of labels can be computed due to Lemma 9 in constant time, it uses Lemma 13 to compute $L(t)$ for every inner node t of T in time $\mathcal{O}(|P(T_t)| l^2 \sqrt{l})$. The total running time of the algorithm is then the sum of the running time for any inner node of T plus the number of leaves of T, i.e., $\mathcal{O}(|P(T)||V(T)|^{\frac{5}{2}})$. ◀

▶ **Theorem 1.** EDP *can be solved in time $\mathcal{O}(|P||V(G)|^{\frac{5}{2}})$ on graphs with feedback vertex set number one.*

Proof. We first employ Lemma 14 to determine whether every tree T of $G - X$ is $\gamma_\emptyset$-connected. If so we output YES and otherwise NO. Correctness follows from Lemma 8. ◀

4 Treewidth and Maximum Degree

The goal of this section is to obtain an fpt-algorithm for EDP parameterized by the treewidth ω and maximum degree Δ of the input graph.

▶ **Theorem 15.** EDP *can be solved in time* $2^{\mathcal{O}(\Delta\omega^2)} \cdot n$*, where* ω*,* Δ *and* n *are the treewidth, maximum degree and number of vertices of the input graph* G*, respectively.*

Proof Sketch. Let (G, P) be an instance of EDP and let $(T, \mathcal{B})$ be a nice tree-decomposition of G of width at most $k = 5\omega + 4$; recall that such $(T, \mathcal{B})$ can be computed in time $2^{\mathcal{O}(k)}$ by Fact 6. Consider the following leaf-to-root dynamic programming algorithm Å, executed on T. At each bag B_t associated with a node t of T, Å will compute a table $\mathcal{M}_t$ of *records*, which are tuples of the form $\{(\mathsf{used}, \mathsf{give}, \mathsf{single})\}$ where:

- used is a multiset of subsets of B_t of cardinality 2 with each subset occurring at most Δ times,
- give is a mapping from subsets of B_t of cardinality 2 to $[\Delta]$, and
- single is a mapping which maps each terminal $a_i \in Y_t$ such that its counterpart $b_i \notin Y_t$ to an element of B_t.

Before we proceed to describe the steps of the algorithm itself, let us first introduce the semantics of a record. For a fixed t, we will consider the graph G_t obtained from $G[Y_t]$ by removing all edges with both endpoints in B_t (we note that this “pruned” definition of G_t is not strictly necessary for the algorithm, but makes certain steps easier later on). Then $\alpha = \{(\mathsf{used}, \mathsf{give}, \mathsf{single})\} \in \mathcal{M}_t$ if and only if there exists a set of edge disjoint paths Q in G_t and a surjective mapping τ from terminal pairs occurring in Y_t to subsets of B_t of size two with the following properties:

- For each terminal pair ab that occurs in Y_t:
  - Q either contains a path whose endpoints are a and b, or
  - Q contains an a-x_1 path for some $x_1 \in B_t$ and a b-x_2 path for some $x_2 \in B_t$ which is distinct from x_1, and furthermore $\tau(ab) = \{x_1, x_2\} \in$used;
- for each terminal pair ab such that $a \in Y_t$ but $b \notin Y_t$:
  - Q contains a path whose endpoints are a and $x \in B_t$, where $(a, x) \in \mathsf{single}$;
- for each distinct $x_1, x_2 \in B_t$, Q contains precisely $\mathsf{give}(\{x_1, x_2\})$ paths from x_1 to x_2.

In the above case, we say that Q witnesses α. It is important to note that the equivalence between the existence of records and sets Q of pairwise edge disjoint paths only holds because of the bound on the maximum degree. That is because every vertex of G has degree at most Δ, it follows that any set Q of pairwise edge disjoint paths can contain at most Δ paths containing a vertex in the boundary. Moreover, we note that by reversing the above considerations, given a set of edge disjoint paths Q in G_t satisfying a certain set of conditions, we can construct in time $3^{\Delta k}$ a set of records in $\mathcal{M}_t$ that are witnessed by Q (one merely needs to branch over all options of assigning paths in α which end in the boundary: they may either contribute to give or to single or to used). These conditions are that each path either (i) connects a terminal pair, (ii) connects a terminal pair to two vertices in B_t, (iii) connects two vertices in B_t, or (iv) connects a terminal $a \in Y_t$ whose counterpart $b \notin Y_t$ to a vertex in B_t.

Å runs as follows: it begins by computing the records $\mathcal{M}_t$ for each leaf t of T. It then proceeds to compute the records for all remaining nodes in T in a bottom-up fashion, until it computes $\mathcal{M}_r$. Since $B_r = \emptyset$, it follows that (G, P) is a Yes-instance if and only if $(\emptyset, \emptyset, \emptyset) \in \mathcal{M}_r$. For each record α, it will keep (for convenience) a set Q_α of edge disjoint paths witnessing α. Observe that while for each specific α there may exist many possible choices of Q_α, all of these interact with B_t in the same way.

We make one last digression before giving the procedures used to compute $\mathcal{M}_t$ for the four types of nodes in nice tree-decompositions. First, observe that the size of one particular record is at most $\Delta k^2 + \Delta k^2 + |\mathsf{single}|$. Since the number of edge disjoint paths in G_t ending in B_t is upper-bounded by Δk, it follows that each record in $\mathcal{M}_t$ satisfies $|\mathsf{single}| \leq \Delta k$ and in particular each such record has size at most $\mathcal{O}(\Delta k^2)$. As a consequence, $|\mathcal{M}_t| \leq 2^{\mathcal{O}(\Delta k^2)}$ for each node t in T, which is crucial to obtain an upper-bound on the running time of Å. It remains to formalize the computation of the records for the four types of nodes of a nice tree decomposition.

Summary and running time

Algorithm Å begins by invoking Fact 6 to compute a tree-decomposition of width at most $k = 5\omega + 4$ and $\mathcal{O}(n)$ nodes, and then converts this into a nice tree-decomposition $(T, \mathcal{B})$ of the same width and also $\mathcal{O}(n)$ nodes. It then proceeds to compute the records $\mathcal{M}_t$ (along with corresponding witnesses) for each node t of T in a leaves-to-root fashion, using the procedures described above. The number of times any procedure is called is upper-bounded by $\mathcal{O}(n)$, and the running time of every procedure is upper-bounded by the worst-case running time of the procedure for forget nodes. There, for each record β in the data table of the child of t, the algorithm takes its witness Q_β and uses branching to construct at most $k^{\mathcal{O}(\Delta k)}$ new witnesses (after the necessary conditions are checked). Each such witness Q_α gives rise to a set of records that can be computed in time $3^{\Delta k}$, which are then added to $\mathcal{M}_t$ (unless they are already there). All in all, the running time of this procedure is upper-bounded by $2^{\mathcal{O}(\Delta k^2)} \cdot k^{\mathcal{O}(\Delta k)} \cdot 3^{\Delta k} = 2^{\mathcal{O}(\Delta k^2)}$, and the run-time of the algorithm follows. ◀

5 Lower Bounds of EDP for Parameters of the Augmented Graph

This section is devoted to a proof of the following theorem.

▶ **Theorem 3.** *EDP is* W[1]*-hard parameterized by the feedback vertex set number of the augmented graph.*

Note that since the feedback vertex set number upper-bounds the treewidth, Theorem 3 complements the result in [22] showing that EDP is solvable in polynomial time for bounded treewidth.

As intermediate steps for the proof of Theorem 3 we establish the W[1]-hardness of several interesting variants of a multidimensional version of the well-known Subset Sum problem as well as several directed and undirected versions of EDP, which we believe are interesting in their own right. Namely, the proof starts by showing W[1]-hardness for the following problem using a reduction from the well-known Multi-Colored Clique (MCC) problem [7].

Multidimensional Subset Sum (MSS)

Input:	An integer k, a set $S = \{s_1, \ldots, s_n\}$ of item-vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$ and a target vector $t \in \mathbb{N}^k$.
Parameter:	k
Question:	Is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t$?

Using a reduction from MSS, the proof then continues by establishing W[1]-hardness for the following more relaxed version of MSS.

Multidimensional Relaxed Subset Sum (MRSS)

Input: An integer k, a set $S = \{s_1, \dots, s_n\}$ of item-vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$, a target vector $t \in \mathbb{N}^k$, and an integer k'.

Parameter: $k + k'$

Question: Is there a subset $S' \subseteq S$ with $|S'| \leq k'$ such that $\sum_{s \in S'} s \geq t$?

Next, we reduce from MRSS to the following directed variant of EDP.

Multiple Directed Edge Disjoint Paths (MDEDP)

Input: A directed graph G, a set P of ℓ triples (s_i, t_i, n_i) with $1 \leq i \leq \ell$, $s_i, t_i \in V(G)$, and $n_i \in \mathbb{N}$.

Parameter: $\mathbf{fvs}(\overline{G}) + |P|$

Question: Is there a set of pairwise edge disjoint paths containing n_i paths from s_i to t_i for every i with $1 \leq i \leq \ell$?

In the above, $\overline{G}$ denotes the underlying undirected graph of a given directed graph G.

Using a known result that allows to reduce the directed EDP problem to the undirected EDP problem given by Vygen [21], we then show that the undirected variant of the above problem, which we refer to as MUEDP, of MDEDP is also W[1]-hard. The final step of the proof is then a straightforward reduction from MUEDP to the EDP problem parameterized by the feedback vertex set number of the augmented graph.

6 An fpt-Algorithm for EDP using the Augmented Graph

In light of Theorem 3, it is natural to ask whether there exist natural structural parameters of the augmented graph which would give rise to fpt-algorithms for EDP but which cannot be used on the input graph. In other words, does considering the augmented graph instead of the input graph provide any sort of advantage in terms of fpt-algorithms? In this section we answer this question affirmatively by showing that EDP is fixed-parameter tractable parameterized by the so-called *fracture number* of the augmented graph. We note that a parameter similar to the fracture number has recently been used to obtain fpt-algorithms for Integer Linear Programming [8].

▶ **Definition 16.** A vertex subset X of a graph H is called a *fracture modulator* if each connected component in $H \setminus X$ contains at most $|X|$ vertices. We denote the size of a minimum-cardinality fracture modulator in H as $\text{frac}(H)$ or the *fracture number* of H.

We begin by making a simple structural observation about fracture modulators.

▶ **Lemma 17.** *Let (G, P) be an instance of* EDP *and let k be the fracture number of its augmented graph. Then there exists a fracture modulator X of G^P of size at most $2k$ such that X does not contain any terminal vertices. Furthermore, such a fracture modulator X can be constructed from any fracture modulator of size at most k in linear time.*

We note that the problem of computing a fracture modulator of size at most k has been recently considered in the context of Integer Linear Programming [8].

▶ **Lemma 18** ([8, Theorems 7 and 8]). *There exists an algorithm which takes as input a graph G and an integer k, runs in time at most $\mathcal{O}((k+1)^k |E(G)|)$, and outputs a fracture modulator of cardinality at most k if such exists. Moreover, there is a polynomial-time algorithm that either computes a fracture modulator of size at most $(k+1)k$ or outputs correctly that no fracture modulator of size at most k exists.*

For the rest of this section, let us fix an instance (G,P) of EDP with a fracture modulator X of G^P of cardinality k which does not contain any terminals. Furthermore, since the subdivision of any edge (i.e., replacing an edge by a path of length 2) in (G,P) does not change the validity of the instance, we will assume without loss of generality that $G[X]$ is edgeless; in particular, any edges that may have had both endpoints in X will be subdivided, creating a new connected component of size 1.

Our next step is the definition of *configurations*. These capture one specific way a connected component C of $G^P - X$ may interact with the rest of the instance. It will be useful to observe that for each terminal pair ab there exists precisely one connected component C of $G^P - X$ which contains both of its terminals; we say that *ab occurs* in C. For a connected component C, we let C^+ denote the induced subgraph on $G^P[C \cup X]$.

A *trace* is a tuple $(x_1, \dots, x_\ell)$ of elements of X. A configuration is a tuple (α, β) where

- α is a multiset of at most k traces, and
- β is a mapping from subsets of X of cardinality 2 to $[k^2]$.

A component C of G^P *admits* a configuration (α, β) if there exists a set of edge disjoint paths $\mathcal{F}$ in C^+ and a surjective mapping τ (called the *assignment*) from α to the terminal pairs that occur in C with the following properties.

- For each terminal pair st that occurs in C:
  - $\mathcal{F}$ either contains a path whose endpoints are s and t, or
  - $\mathcal{F}$ contains an s-x_1 path for some $x_1 \in X$ and a t-x_2 path for some distinct $x_2 \in X$ and there exists a trace $L = (x_1, \dots, x_2) \in \alpha$ such that $\tau(L) = st$.
- for each distinct $a, b \in X$, $\mathcal{F}$ contains precisely $\beta(\{a,b\})$ paths from a to b.
- $\mathcal{F}$ contains no other paths than the above.

Intuitively, α stores information about how one particular set of edge disjoint paths A which originate in C is routed through the instance: they may either be routed only through C^+ (in which case they don't contribute to α), or they may leave C^+ (in which case α stores the order in which these paths visit vertices of X, i.e., their trace). On the other hand, β stores information about how paths that originate outside of C can potentially be routed through C (in a way which does not interfere with A). Observe that for any particular α there may exist several distinct configurations ((α, β_1), (α, β_2) and so forth); similarly, for any particular β there may exist several distinct configurations ((α_1, β), (α_2, β) and so forth).

If a set $\mathcal{F}$ of edge disjoint paths in C^+ satisfies the conditions specified above for a configuration (α, β), we say that $\mathcal{F}$ *gives rise* to (α, β). Clearly, given $\mathcal{F}$ and (α, β), it is possible to determine whether $\mathcal{F}$ gives rise to (α, β) in time polynomial in $|V(C)|$.

While configurations capture information about how a component can interact with a set of edge disjoint paths, our end goal is to have a way of capturing all important information about a component irrespective of any particular selection of edge disjoint paths. To this end, we introduce the notion of *signatures*. A signature of a component C, denoted $\text{sign}(C)$, is the set of all configurations which C admits. The set of all configurations is denoted by Λ.

▶ **Lemma 19.** *Given a component C, it is possible to compute $sign(C)$ in time at most $k^{\mathcal{O}(k^2)}$. Furthermore, $|sign(C)| \leq |\Lambda| \leq k^{\mathcal{O}(k^2)}$.*

Our next step is the formulation of a clear condition linking configurations of components in $G^P - X$ and solving (G,P). This condition will be of importance later, since it will be checkable by an integer linear program. For a trace α, we say that *a, b occur consecutively* in α if elements a and b occur consecutively in the sequence α (regardless of their order), i.e., $\alpha = (\dots, a, b, \dots)$ or $\alpha = (\dots, b, a, \dots)$. Let $\mathcal{D}$ be the set of connected components of $G^P - X$.

A *configuration selector* is a function which maps each connected component C in $G^P - X$ to a configuration $(\alpha, \beta) \in \text{sign}(C)$. We say that a configuration selector S is *valid* if it satisfies the condition that $\text{dem}(ab) \leq \text{sup}(ab)$ for every $\{a, b\} \subseteq X$, where dem (*demand*) and sup (*supply*) are defined as follows: $\text{dem}(ab)$ is the number of traces in $\bigcup_{C \in \mathcal{D}} S(C)$ where ab occur consecutively and $\text{sup}(ab)$ is the sum of all the values $\beta(a, b)$ in $\bigcup_{C \in \mathcal{D}} S(C)$.

The next, crucial lemma links the existence of a valid configuration selector to the existence of a solution for EDP.

▶ **Lemma 20.** *(G, P) is a* Yes*-instance if and only if there is a valid configuration selector.*

The problem whether there is a valid configuration selector can be easily translated into an integer linear program with a number of variables that can be bounded in terms of k. It then follows from [18] that the problem is fixed-parameter tractable parameterized by k.

▶ **Lemma 21.** *There exists an algorithm which takes as input an* EDP *instance (G, P) and a fracture modulator X of G^P and determines whether there exists a valid configuration selector S in time at most $2^{2^{k^{\mathcal{O}(k^2)}}} \cdot |V(G)|$.*

▶ **Theorem 4.** EDP *is fixed-parameter tractable parameterized by the fracture number of the augmented graph.*

Proof. We begin by computing a fracture modulator of the augmented graph by Lemma 18. We then use Lemma 21 to determine whether a valid configuration selector S exists, which by Lemma 20 allows us to solve EDP. ◀

Having established that EDP is fixed-parameter tractable parameterized by the fracture number, let us briefly consider potential extensions of the parameter. In particular, one might be tempted to think that tractability still applies if instead of bounding the size of each component one only bounds the number of terminal pairs in each component. We conclude this section by showing that this is not the case: even if both the deletion set and the number of terminal pairs in each component are bounded by a constant, EDP remains NP-complete.

▶ **Theorem 22.** EDP *is NP-complete even if the augmented graph G^P of the instance has a deletion set D of size 6 such that each component of $G^P - D$ contains at most 1 terminal pair.*

The proof of Theorem 22 is based on a polynomial reduction from the Multiple Edge Disjoint Paths (MEDP) problem, where given an undirected graph G, three pairs (s_1, t_1), (s_2, t_2), and (s_3, t_3) of terminals and three integers n_1, n_2, and n_3 one asks whether there is a set of pairwise edge disjoint paths containing n_1 paths between s_1 and t_1, n_2 paths between s_2 and t_2, and n_3 paths between s_3 and t_3. MEDP is known to be strongly NP-complete [21, Theorem 4].

7 Conclusion

Our results close a wide gap in the understanding of the complexity landscape of EDP parameterized by structural parameterizations. On the positive side we present three novel algorithms for the classical EDP problem: a polynomial-time algorithm for instances with a FVS of size one, an fpt-algorithm w.r.t. the treewidth and maximum degree of the input graph, and an fpt-algorithm for instances that have a small deletion set into small components in the augmented graph. On the negative side we solve a long-standing open problem concerning the complexity of EDP parameterized by the treewidth of the augmented graph: unlike related multicut problems [13], EDP is W[1]-hard parameterized by the feedback vertex set number of the augmented graph.

References

1 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

2 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

3 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

4 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An O(sqrt(n)) approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(7):137–146, 2006.

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

6 Reinhard Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 4th edition, 2010.

7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

8 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In *IJCAI 2017*, pages 607–613, 2017.

9 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In *Proc. SWAT 2016*, volume 53 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl, 2016.

10 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. In *Proc. ESA 2016*, pages 42:1–42:17, 2016.

11 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series.* Springer Verlag, Berlin, 2006.

12 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.

13 Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. *Inf. Process. Lett.*, 103(4):136–141, 2007.

14 Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

15 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Stephan Kreutzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proc. STOC 2014*, pages 70–78. ACM, 2014.

16 T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, Berlin, 1994.

17 Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Math. Program.*, 99(1):63–87, 2004.

18 H. W. Lenstra and Jr. Integer programming with a fixed number of variables. *MATH. OPER. RES*, 8(4):538–548, 1983.

19 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Applied Mathematics*, 115(1-3):177–186, 2001.

20 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

21 Jens Vygen. Np-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61(1):83–90, 1995.

22 Xiao Zhou, Syurei Tamura, and Takao Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26(1):3–30, 2000.

Barrier Coverage with Non-uniform Lengths to Minimize Aggregate Movements[*][†]

Serge Gaspers[‡][1], Joachim Gudmundsson[2], Julián Mestre[3], and Stefan Rümmele[4]

1 **UNSW Sydney, and Data61, CSIRO, Alexandria, Australia**
sergeg@cse.unsw.edu.au

2 **The University of Sydney, Australia**
joachim.gudmundsson@gmail.com

3 **The University of Sydney, Australia**
mestre@it.usyd.edu.au

4 **UNSW Sydney and The University of Sydney, Australia**
stefan.rummele@sydney.edu.au

Abstract

Given a line segment $I = [0, L]$, the so-called *barrier*, and a set of n sensors with varying ranges positioned on the line containing I, the *barrier coverage* problem is to move the sensors so that they cover I, while minimising the total movement. In the case when all the sensors have the same radius the problem can be solved in $O(n \log n)$ time (Andrews and Wang, Algorithmica 2017). If the sensors have different radii the problem is known to be NP-hard to approximate within a constant factor (Czyzowicz *et al.*, ADHOC-NOW 2009).

We strengthen this result and prove that no polynomial time $\rho^{1-\varepsilon}$-approximation algorithm exists unless P = NP, where ρ is the ratio between the largest radius and the smallest radius. Even when we restrict the number of sensors that are allowed to move by a parameter k, the problem turns out to be W[1]-hard. On the positive side we show that a $((2+\varepsilon)\rho+2/\varepsilon)$-approximation can be computed in $O(n^3/\varepsilon^2)$ time and we prove fixed-parameter tractability when parameterized by the total movement assuming all numbers in the input are integers.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Barrier coverage, Sensor movement, Approximation, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.37

1 Introduction

The original motivation for the problem of covering barriers comes from intrusion detection, where the goal is to guard the boundary (barrier) of a region in the plane. In this case the barrier can be described by a polygon and the initial position of the sensors can be anywhere in the plane. The barrier coverage problem, and many of its variants, has received much attention in the wireless sensor community, see for example [2, 4, 10] and the recent surveys [12, 13]. Large scale barriers with more than a thousand sensors have been experimentally tested and evaluated [2].

[*] This work was supported by the Australian Research Council (ARC) under the Discovery Projects funding scheme (DP150101134).

[†] A full version of the paper is available at [9], https://arxiv.org/abs/1709.10285.

[‡] Serge Gaspers is the recipient of an ARC Future Fellowship (FT140100048).

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 37; pp. 37:1–37:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

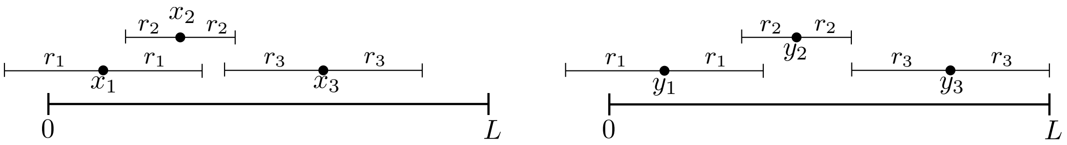


Figure 1 (left) Illustrating an instance with three sensors $\{1, 2, 3\}$ and sensor intervals. (right) The sensors have moved such that the sensor intervals cover the barrier $[0, L]$.

In a general setting of the barrier coverage problem each sensor has a fixed sensor radius and is initially placed in the plane and the cost of moving a sensor is proportional to the Euclidean distance it is moved. In this paper we consider the special case where we have n sensors on the real line. Each sensor $i = 1, \ldots, n$ has a location x_i and a radius r_i. When located at y_i, the i-th sensor covers the *interval* $B(y_i, r_i) = [y_i - r_i, y_i + r_i]$. The goal is to move around the sensor intervals to cover the interval $[0, L]$, the so-called *barrier*. In other words, for each sensor, we need to decide its new location y_i so that $[0, L] \subseteq \bigcup_i B(y_i, r_i)$. The cost of the solution is the sum of sensor movements: $\text{cost}(y) = \sum_i |y_i - x_i|$, and the objective is to find a feasible solution of minimum cost.

1.1 Our Results and Related Work

Even though the barrier coverage problem, and many of its variants, has received a lot of attention from the wireless sensor community, not much is known from a theoretical point of view. In the literature three different optimisation criteria have been considered: minimize the sum of movements (min-sum), minimize the maximum movement (min-max) and, minimize the number of sensors that move (min-num).

Dobrev *et al.* [7] studied the min-sum and min-max version in the case when the sensors' start position can be anywhere in the plane and k parallel barriers are required to be covered. However, they restricted the movement of the sensors to be perpendicular to the barriers. They showed an $O(kn^{k+1})$ time algorithm. If the barriers are allowed to be horizontal and vertical then the problem is NP-complete, even for two barriers.

Most of the existing research has focussed on the special case when the barrier is a line segment I and all the sensors are initially positioned on a line containing I.

The Min-Sum model

If all intervals have the same radius, it is not difficult to show that any solution can be converted into one where $x_i < x_j$ if and only if $y_i < y_j$ without incurring any extra cost. Czyzowicz *et al.* [6] showed an $O(n^2)$ time algorithm for this case which was later improved to $O(n \log n)$ by Andrews and Wang [1]. Andrews and Wang also showed a matching $\Omega(n \log n)$ lower bound. When the radii are non-uniform, this is not the case anymore. In fact, Czyzowicz *et al.* [6] showed that this variant of the problem is NP-hard, and remarked that not even a 2-approximation is possible in polynomial time. In fact their hardness proof can be modified to show (Theorem 7) that no approximation factor is possible. The catch is that the instance used in the reduction needs to have some intervals that are very small and some intervals that are very large. This is a scenario that is not likely to happen in practice, so the question is whether there is an approximation algorithm whose factor depends on the ratio of the largest radius to the smallest radius.

Let ρ be the ratio between the largest radius $r_{\max} = \max_i r_i$ and the smallest radius $r_{\min} = \min_i r_i$. Theorem 7 states that no $\rho^{1-\varepsilon}$ approximation algorithm exists for any $\varepsilon > 0$

unless $\mathsf{P} = \mathsf{NP}$. On the positive side we show an $O(n^3/\varepsilon^2)$ time $((2+\varepsilon)\rho+2/\varepsilon)$-approximation algorithm for any given $\varepsilon > 0$. The general idea is to look at "order-preserving" solutions, that is, solutions where the set of sensors covering the barrier maintains their individual order from left to right. This will be described in more detail in Section 2.

We also study the problem from the perspective of parameterized complexity and show that the problem is hard even if the number of intervals required to move is small, that is $\mathsf{W}[1]$-hardness with respect to parameter number of moved intervals. Complementary, we provide a fixed-parameter tractable algorithm when the problem is parameterized by the budget, i.e., the target sum of movements.

The Min-Max and Min-Num models

Czyzowicz *et al.* [6] also considered min-max version of the problem, where the aim is to minimize the maximum movement. If the sensors have the same radius they gave an $O(n^2)$ time algorithm. Chen *et al.* [5] improved the bound to $O(n \log n)$. In the same paper Chen *et al.* presented an $O(n^2 \log n)$ time algorithm for the case when the sensors have different radius. For the min-num version Mehrandish *et al.* [11] showed that the problem can be solved in polynomial time using dynamic programming if the sensor radii are uniform, otherwise the problem is NP-hard.

2 Order-Preserving Approximations

Let y be a solution to the barrier problem. We say a subset of intervals $S \subseteq \{1, \ldots, n\}$ is *active* for a solution y if the intervals in S alone are enough to cover the barrier. Additionally, we say that S is a *minimal active set* if no proper subset of S is active. Notice that in an optimal solution y if $y_i \neq x_i$ then i must belong to a minimal active set. Without loss of generality we assume that $x_1 \leq x_2 \leq \cdots \leq x_n$. We say a solution y is *order-preserving* if it has an active set S such that for any $i, j \in S$ with $i < j$, we have $y_i < y_j$.

Our algorithm is based on finding a nearly optimal order-preserving solution. First we show, in Section 2.1, that there always exists an order-preserving solution that is a good approximation of the optimal unrestricted solution, and prove that our analysis is almost tight. Then, in Section 2.2, we show how to compute a nearly optimal order-preserving solution in polynomial time.

2.1 Quality of Order-Preserving Solutions

The high level idea to prove that there exists an order-preserving solution that approximates the optimal solution is to start from an arbitrary optimal solution y and progressively modify the positions of two overlapping active intervals so that they are in the right order and together cover the exact same portion of the barrier, as shown in Fig. 2. We refer to this process as the *untangling* process.

This untangling process continues until all overlapping active intervals are in order. Let us denote the resulting solution with $\hat{y}$. Our goal is to charge the cost of $\hat{y}$ to the intervals in such a way that the total charge an interval can receive is comparable to its contribution to the cost of y. More formally, we define an *β-balanced cost sharing scheme* to be a function $\xi : S \to \mathbb{R}^+$, where

1. $\text{cost}(\hat{y}) \leq \sum_{i \in S} \xi(i)$, and
2. $\xi(i) \leq \beta\,|x_i - y_i|$ for all $i \in S$.

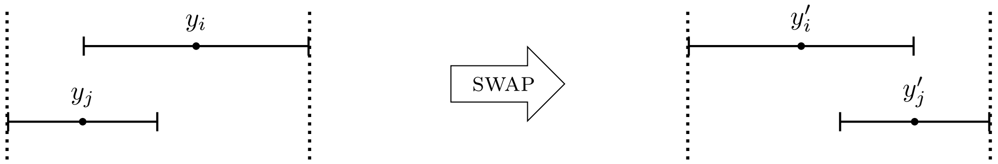


Figure 2 Two overlapping intervals i and j being swapped. After the swap the union of the intervals cover the same section of the barrier but their centers swap order.

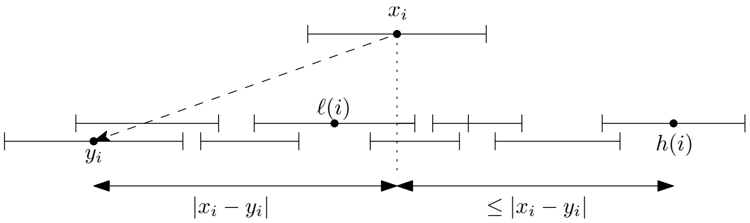


Figure 3 An interval i and its relation to $\tilde{\gamma}(i)$. In this case $y_i < x_i$, but a symmetric picture holds when $y_i > x_i$.

It is easy to see that the existence of a well balanced cost sharing scheme implies a good approximation guarantee.

▶ **Lemma 1.** *Let $\hat{y}$ be the result of untangling an optimal solution y. If $\hat{y}$ admits an β-balanced cost sharing scheme then $\hat{y}$ is β-approximate.*

Proof. We bound the cost of $\hat{y}$ as follows: $\text{cost}(\hat{y}) \leq \sum_i \xi(i) \leq \sum_i \beta|x_i - y_i| = \beta \cdot \text{cost}(y) = \beta \cdot \text{OPT}$, where the first two inequalities follow from the definition of β-balancedness and the last equality follows from the fact that y is optimal. ◀

To show the existence of a good cost sharing scheme, we will study the structure of an optimal solution y and its untangling process leading to the order-preserving solution $\hat{y}$.

Let $\gamma(i) \subseteq S$ be the set of indices that *cross* i, that is, $i < j$ and $y_i > y_j$, or $i > j$ and $y_i < y_j$. Let $\tilde{\gamma}(i) = \{j \in \gamma(i) : |x_i - y_i| \geq |x_j - y_j|\}$, that is, the set of sensors in $\gamma(i)$ that move at most as far as i. If $y_i < x_i$ we define $h(i)$ to be the y-rightmost sensor in $\tilde{\gamma}(i)$, and we let $\ell(i)$ be the y-rightmost sensor in $\tilde{\gamma}(i)$ to the left or equal of x_i. See Figure 3. Symmetrically, if $y_i \geq x_i$ we define $h(i)$ to be the y-leftmost sensor in $\tilde{\gamma}(i)$, and $\ell(i)$ to be the y-leftmost sensor in $\tilde{\gamma}(i)$ to the right or equal of x_i. For sake of brevity, when the interval i is clear from context, we refer to $h(i)$ as h and to $\ell(i)$ as ℓ. Note that $\ell(i)$ is not well-defined in the case when there are no intervals between x_i and y_i.

Let us make some observations about the intervals. Figure 3 sums up these observations by depicting i together with $\tilde{\gamma}(i)$ with ℓ and h highlighted.

▶ **Observation 1.** *Every $j \in \tilde{\gamma}(i)$ must have $y_j \in [x_i - |x_i - y_i|, x_i + |x_i - y_i|]$.*

Proof. Note that if $x_i = y_i$ then the claim is trivially true since $\tilde{\gamma}(i) = \emptyset$.

Without loss of generality assume $x_i > y_i$, since the case $x_i < y_i$ is symmetric. Since $j \in \tilde{\gamma}(i)$ we have $|x_j - y_j| \leq |x_i - y_i|$, and it follows that $x_j < x_i$ and $y_j > y_i$. Therefore, $y_j > y_i = x_i - |x_i - y_i|$ and $y_j < x_j + |x_i - y_i| < x_i + |x_i - y_i|$. ◀

▶ **Observation 2.** *Let y be an optimal solution and let S be a minimal active set of y. Every point stabs (intersects) at most two intervals in S.*

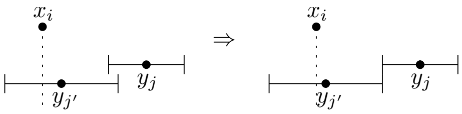


Figure 4 Illustrating the proof of Observation 3, showing that the intervals of j and j' cannot overlap in y.

Proof. If three active intervals in S are stabbed by one point, then one of those intervals can be removed without making the solution infeasible, thus contradicting minimality of S. ◀

▶ **Observation 3.** *In an optimal solution y, if $y_i < x_i$ then the intervals $j \in \widetilde{\gamma}(i)$ such that $y_j > x_i$ do not overlap; similarly, if $y_i > x_i$ then the intervals $j \in \widetilde{\gamma}(i)$ such that $y_j < x_i$ do not overlap.*

Proof. Without loss of generality assume $x_i > y_i$, since the case $x_i < y_i$ is symmetric. If there were two indices $j, j' \in \widetilde{\gamma}(i)$ that overlap in y and $y_j > y_{j'} > x_i$, then we could reduce $y_{j'}$ by $r_j + r_{j'} - (y_j - y_{j'})$ to get another feasible solution with lower cost, since $x_j, x_{j'} < x_i$. See Figure 4 for an illustration. ◀

▶ **Observation 4.** *If ℓ is well defined for i in an optimal solution y then*

$$\sum_{j\in\widetilde{\gamma}(i)} 2r_j \le 3|x_i - y_i| + r_\ell + r_h.$$

Proof. Note that if $x_i = y_i$ then the claim is trivially true since $\widetilde{\gamma}(i) = \emptyset$.

Without loss of generality assume $x_i > y_i$, since the case $x_i < y_i$ is symmetric. By Observation 2 every point in the interval $[y_i, x_i]$ stabs at most two intervals from $\widetilde{\gamma}(i)$. By Observation 3 every point in the interval $[x_i, y_h]$ stabs at most one interval $j \in \widetilde{\gamma}(i)$ such that $y_j > x_i$. This accounts for the term $3|x_i - y_i|$. Additionally, we have to add r_h to account for the interval $[y_h, y_h + r_h]$ and r_ℓ, since $\ell(i)$ might overlap interval $[x_i, x_i + r_\ell]$. Let j be the y-leftmost sensor in $\widetilde{\gamma}(i)$. We do not have to account for the fact that x_j might end left of y_i, that is the interval $[y_j - r_j, y_i]$. The reason is that $|y_i - y_j + r_j| < r_i$ and counted the interval $[y_i, y_i + r_i]$ already needlessly when considering that $[y_i, x_i]$ stabs at most two intervals from $\widetilde{\gamma}(i)$. It follows that $\sum_{j\in\widetilde{\gamma}(i)} 2r_j \le 3|x_i - y_i| + r_\ell + r_h$. ◀

▶ **Observation 5.** *If ℓ is well defined for i in an optimal solution y then*

$$|\widetilde{\gamma}(i)| \le 3 + \frac{3\,|x_i - y_i| - 2\,r_i - r_\ell - r_h}{2\,r_{\min}}.$$

Proof. From Observation 4 we have $\sum_{j\in\widetilde{\gamma}(i)} 2r_j \le 3\,|x_i - y_i| + r_\ell + r_h$. Notice that each interval in $\widetilde{\gamma}(i)$ has length at least $2r_{\min}$, therefore the number of intervals in $\widetilde{\gamma}(i)$ is no more than $\sum_{j\in\widetilde{\gamma}(i)} 2r_j$ divided by $2r_{\min}$. To get a better bound we count three intervals explicitly: $\ell(i)$, $h(i)$, and j, where j is the y-leftmost sensor if $x_i > y_i$ or the rightmost otherwise.

Note that if $x_i = y_i$ then the claim is trivially true since $\widetilde{\gamma}(i) = \emptyset$. Without loss of generality assume $x_i > y_i$, since the case $x_i < y_i$ is symmetric. Ignoring j, we can adjust the bound from Observation 4 as follows. Since by Observation 2 every point stabs at most two intervals, only j might overlap with i in y. Hence, we only need to consider the interval $[y_i + r_i, x_i]$ where every point stabs at most two intervals from $\widetilde{\gamma}(i)$. Hence, ignoring the three explicitly counted intervals, the sum of the lengths of the remaining intervals of $\widetilde{\gamma}(i)$ can be bounded by $2(|x_i - y_i| - r_i) + |x_i - y_i| + r_\ell + r_h - 2r_\ell - 2r_h$. Therefore, we have $|\widetilde{\gamma}(i)| \le 3 + \frac{3|x_i - y_i| - 2r_i - r_\ell - r_h}{2r_{\min}}$. ◀

Now everything is in place to describe our cost sharing schemes. Our first scheme is simpler to describe and is $(3\rho+4)$-balanced. Our second scheme is a refinement and is $\big((2+\epsilon)\rho+2/\epsilon\big)$-balanced for any $\epsilon > 0$.

▶ **Lemma 2.** *For an optimal solution y to the barrier problem there is an untangling $\hat{y}$ of y such that there is a $(3\rho+4)$-balanced cost sharing scheme.*

Proof. The high level idea of our charging scheme is as follows: When i swaps places with $j \in \tilde{\gamma}(i)$, we charge i enough to pay for the movements of both i and j. In particular if $\tilde{\gamma}(i) = \emptyset$ then we do not charge i at all, that is, $\xi(i) = 0$.

From now on we assume that $\tilde{\gamma}(i) \neq \emptyset$. For the analysis it will be useful to study how i moves in the untangling process. If $y_i < x_i$ then swapping i and $j \in \tilde{\gamma}(i)$ always moves i to the right; similarly, if $y_i > x_i$ then swapping i and $j \in \tilde{\gamma}(i)$ always moves i to the left. On the other hand, when swapping i and $j \in \gamma(i) \setminus \tilde{\gamma}(i)$, the interval i can move either left or right.

We consider two scenarios. If $\hat{y}_i$ ends up on the same side of x_i as y_i then $|x_i - \hat{y}_i| \leq \sum_{j \in \gamma(i)\setminus\tilde{\gamma}(i)} 2r_j + |x_i - y_i|$, so we charge $2r_j$ to each $j \in \gamma(i) \setminus \tilde{\gamma}(i)$ and $|x_i - y_i|$ to i. Thus, under this scenario, the total amount charged to i is

$$\xi(i) \leq 2r_i|\tilde{\gamma}(i)| + |x_i - y_i| \tag{1}$$

The second scenario is when $\hat{y}_i$ and y_i end up on opposite sides of x_i then $|x_i - \hat{y}_i| \leq \sum_{j\in\gamma(i)} 2r_j - |x_i - y_i|$, so we charge $\sum_{j\in\tilde{\gamma}(i)} 2r_j - |x_i - y_i|$ to i and $2r_j$ to each $j \in \gamma(i) \setminus \tilde{\gamma}(i)$. Thus, under this scenario, the total amount charged to i is

$$\xi(i) \leq 2r_i|\tilde{\gamma}(i)| + \sum_{j\in\tilde{\gamma}(i)} 2r_j - |x_i - y_i|. \tag{2}$$

The rest of the proof is broken up into four cases.

Case 1: Intervals i and $h(i)$ overlap in y.

In this case $\tilde{\gamma}(i) = \{h(i)\}$ and $\tilde{\gamma}(h(i)) = \emptyset$. Furthermore, if there is another interval i' such that $h(i') = h(i)$ then i' and $h(i')$ do not overlap. Indeed, if y_i lies in between $y_{i'}$ and $y_{h(i)}$ then i' and $h(i)$ cannot overlap otherwise there is a point covered by i, i' and $h(i)$; if $y_{i'}$ lies in between y_i and $y_{h(i)}$ we get a similar contradiction, so it must be that $y_{h(i)}$ lies in between y_i and $y_{i'}$. See Fig. 5. This means that i and i' cross, so either $i \in \tilde{\gamma}(i')$ or $i' \in \tilde{\gamma}(i)$, which, together with $h(i) = h(i')$, yields a contradiction.

Therefore, we can run the untangling process so that all pairs i and $h(i)$ that overlap in y are swapped first. Let y' be the solution after these initial swaps are carried out. Then,

$$\begin{aligned} |x_i - {y'}_i| + |x_{h(i)} - {y'}_{h(i)}| &\leq |x_i - y_i| + |x_{h(i)} - y_{h(i)}| + 2|r_i - r_{h(i)}| \\ &\leq 4(|x_i - y_i| + |x_{h(i)} - y_{h(i)}|) \qquad \leq 6|x_i - y_i|. \end{aligned}$$

The first inequality is due to the fact that additional cost comes from swapping i and h, where at most one them moves in a direction that increases the cost and they are overlapping. Hence the additional cost is bounded by $2|r_i - r_{h(i)}|$. The second inequality is due to the fact that the movement $|x_i - y_i| + |x_{h(i)} - y_{h(i)}|$ needs to be larger than $|r_i - r_{h(i)}|$ for i and h to swap positions and both be active.

Later on in the untangling process, i and h may be swapped with another interval, call it j, causing them to move further and to increase their contribution towards $\text{cost}(\hat{y})$. If this happens, we charge the movement of i, or h, to j. Therefore, setting $\xi(i) = 6|x_i - y_i|$ is enough to cover the contribution of i and h to the cost of y that is not covered by other intervals. Obviously, the scheme so far is $(3\rho + 4)$-balanced.

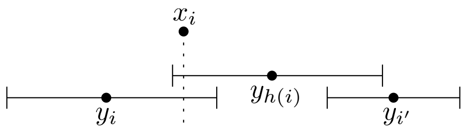


Figure 5 If $h(i) = h(i')$ then i and i' must lie on opposite sides of $h(i)$ in y.

The proof of Cases 2 and 3 are deferred to the long version [9] where it is shown that when ℓ is not well-defined (Case 2) or ℓ is well-defined and intervals ℓ and i overlap in y (Case 3), then $\frac{\xi(i)}{|x_i - y_i|} \leq 2\rho + 1$.

Case 4: ℓ is well-defined and intervals i and ℓ do not overlap in y.

The assumption implies $|x_i - y_i| \geq r_i + r_\ell$. Since we will use Observation 4 to bound $\sum_{j \in \widetilde{\gamma}(i)} 2r_j$, it follows that the sub-case when i is charged the most is when y_i and $\hat{y}_i$ are on opposite sides of x_i, so we start with the bound provided by (2):

$$\begin{aligned}
\xi(i) &\leq 2r_i|\widetilde{\gamma}(i)| + \sum_{j \in \widetilde{\gamma}(i)} 2r_j - |x_i - y_i| \\
&\leq r_i \left(6 + \frac{3|x_i - y_i| - 2r_i - r_\ell - r_h}{r_{\min}}\right) + 2|x_i - y_i| + r_\ell + r_h \\
&= \left(3\frac{r_i}{r_{\min}} + 2 + r_i \frac{6 - 2r_i/r_{\min} - r_\ell/r_{\min} - r_h/r_{\min} + r_h/r_i + r_\ell/r_i}{|x_i - y_i|}\right)|x_i - y_i| \\
&\leq \left(3\frac{r_i}{r_{\min}} + 2 + \frac{4 - 2r_i/r_{\min} + 2r_{\min}/r_i}{1 + r_{\min}/r_i}\right)|x_i - y_i| \qquad \leq \left(3\frac{r_i}{r_{\min}} + 4\right)|x_i - y_i| \\
&\leq (3\rho + 4)\,|x_i - y_i|
\end{aligned}$$

where the second inequality follows from Observations 4 and 5, the third inequality follows from $|x_i - y_i| \geq r_i + r_\ell$, the forth inequality follows from the fact that the right hand side of the previous line decreases with r_ℓ and r_h, and so it is maximized when $r_\ell = r_h = r_{\min}$, and the fifth inequality follows from the fact that third term inside the parenthesis is a decreasing function for $r_i \geq r_{\min}$. This completes the proof of Lemma 2. ◀

▶ **Lemma 3.** *For an optimal solution y to the barrier problem there is an untangling $\hat{y}$ of y such that there is a $\big((2 + \epsilon)\rho + 2/\epsilon\big)$-balanced charging scheme.*

Proof sketch. The key insight to get this charging scheme is to realize that the intervals $j \in \widetilde{\gamma}(i)$ such that y_i and y_j end up on opposite sides of x_i must have $|x_j - y_j| > 0$, so we can use some of this cost to pay for the distance it moves when swapping places with i. If $|x_i - y_j| \geq \epsilon|x_i - y_i|$ then swapping i and j causes j to move $2r_i$, we charge that to j instead of i like before. In this modified charging scheme i gets charged $(1 - \epsilon)\frac{r_i}{r_{\min}}|x_i - y_i|$ less because it does not pay for the movement of $j \in \widetilde{\gamma}(i)$ with $y_j > x_i(1 + \epsilon)$. On the other hand, it has to pay for its own movement when swapped with some j' such that $i \in \widetilde{\gamma}(j')$ and $|x_i - y_i| \geq \epsilon|x_{j'} - y_{j'}|$. However, it can be shown that the total extra charge that an interval i is given is at most $\frac{2}{\epsilon}|x_i - y_i|$. Therefore, the scheme is $\big((2 + \epsilon)\rho + 2/\epsilon\big)$-balanced. ◀

Selecting ϵ appropriately gives a minimum approximation of $2(\rho + \sqrt{2\rho})$. We conclude this sub-section by showing that our analysis is almost tight.

▶ **Lemma 4.** *There is a family of instances where the ratio of the cost of the best order-preserving solution to the cost of the unrestricted optimal solution tends to ρ.*

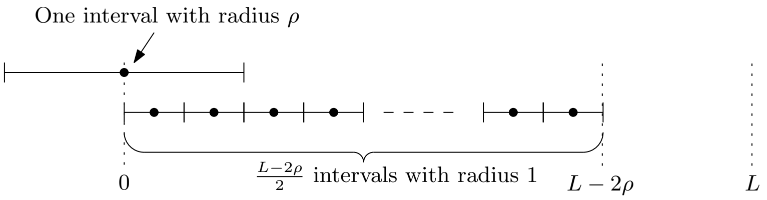

One interval with radius ρ
0
$\frac{L-2\rho}{2}$ intervals with radius 1
$L-2\rho$
L

Figure 6 A family of instances showing that order preserving solution cannot guarantee better than ρ approximation.

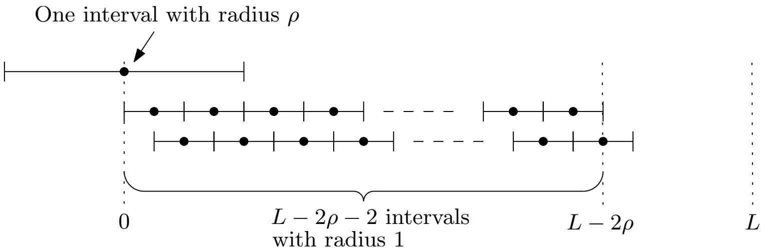

One interval with radius ρ
0
$L-2\rho-2$ intervals with radius 1
$L-2\rho$
L

Figure 7 A family of instances showing that our untangling process can yield solutions that are 2ρ away from the optimum.

Proof. Consider the instance in Figure 6. There are $\frac{L-2\rho}{2}$ unit-radius intervals covering $[0, L-2\rho]$ and one ρ-radius interval covering $[-\rho, \rho]$. The optimal solution moves the long interval $L-\rho$ distance to the right to cover $[L-2\rho, L]$, at a cost of $L-\rho$. On the other hand, the order-preserving solution involves moving each small interval 2ρ units to the right, at a cost of $2\rho\frac{L-2\rho}{2}$. For large enough L the ratio of the cost of these solutions tends to ρ. ◀

As a closing note, we mention that our analysis of the current untangling procedure is nearly tight. Indeed, consider the instance in Figure 7. The optimal solution moves the long interval $L-\rho$ distance to the right. If there is a small gap between two consecutive small intervals, every interval will be active; therefore, in the untangled solution every small interval is moved a distance of 2ρ to the right. This means that the ratio of the cost of the untangled solution to OPT tends to 2ρ as L grows.

2.2 Computing Good Order-Preserving Solutions

First we describe a pseudo-polynomial time algorithm for finding an optimal order-preserving solution. Then we show how to get a $(1+\epsilon)$-approximate order-preserving solution in strongly-polynomial time.

▶ **Lemma 5.** *Assuming the coordinates defining the instance are integral, there is an $O(\text{OPT}^2 n)$ time algorithm for computing an optimal order-preserving solution, where* OPT *is the value of said solution.*

Proof. Consider the following dynamic programming formulation where we let $T[i, b]$ be the largest value such that there is an order-preserving solution using the intervals $1, \ldots, i$ to cover $[0, T[i, b]]$ having cost at most b. For $i = 0$ there is no active set and so $T[0, b] = 0$ for all b. For $i > 0$, if i is not part of the active set of the solution that defines $T[i, b]$ then $T[i, b] = T[i-1, b]$. For $i > 0$, if i is part of the active set in the optimal solution then we can condition on how much i moves, say k units either to the left or to the right. The most coverage that we can possibly get is to move i to $y_i = T[i-1, b-k] + r_i$, which would allow a cover up to $T[i-1, b-k] + 2r_i$; however, this is only possible if $|T[i-1, b-k] + r_i - x_i| \leq k$. On the other hand, if $|T[i-1, b-k] + r_i - x_i| > k$ then it must be that $x_i < T[i-1, b-k] + r_i$ (otherwise k needs to be larger) and the best coverage we can get is then $x_i + k$, which should be larger than $T[i-1, b-k]$. At this point it is straightforward to write a recurrence for $T[i, b]$ that can be computed in $O(b)$ time given the values for $T[i-1, *]$. There are $n \times \text{OPT}$ dynamic programming states and each takes $O(\text{OPT})$ time to compute. ◀

▶ **Lemma 6.** *There is an $O(n^3/\epsilon^2)$ time algorithm for computing a $(1+\epsilon)$-approximate order-preserving solution.*

Proof. For $q = \frac{\epsilon \cdot \text{OPT}}{n}$ we define the following objective function: $\text{cost}'(y) = \sum_i \left\lceil \frac{|y_i - x_i|}{q} \right\rceil$. This new cost function is closely related to the original objective, namely: $\text{cost}(y) \leq q \cdot \text{cost}'(y) \leq \text{cost}(y) + qn$. Using the same dynamic formulation as the one used in the pseudo-polynomial time algorithm, we can optimize cost' in $O(n^3/\epsilon^2)$ time. Furthermore, the value of this solution under the original objective is at most $(1+\epsilon)\text{OPT}$, so the claim follows. ◀

3 Inapproximability Results

The known NP-hardness proof for the barrier coverage problem [6] is a reduction from 3-Partition. The reduction takes an instance of 3-Partition and creates an instance of the barrier coverage problem with integral values, $n+1$ different radii values, and $\rho = cn^d$ for some constants c and d. Computing a 2-approximate solution in this instance is enough to decide the 3-Partition instance. Therefore, there is no 2-approximation unless P = NP. In fact, the same reduction can be used to obtain inapproximability results in terms of ρ.

▶ **Theorem 7.** *There is no polynomial time $\rho^{1-\epsilon}$-approximation algorithm for any constant $\epsilon > 0$ unless* P = NP*.*

Proof. As noted in [6], a similar reduction can be used to construct an instance with $\rho = \alpha cn^d$ for $\alpha > 1$ such that an α-approximation is enough to decide the 3-Partition instance. If we set $\alpha = (cn^d)^{\frac{1-\epsilon}{\epsilon}}$ then we get that $\alpha = \rho^{1-\epsilon}$ and the claim follows. ◀

4 Parameterized Complexity

We show that the barrier coverage problem is hard, even if we only allow a small number of sensors to move. Formally, we show that the following problem is W[1]-hard when parameterized by k.

k-move-Barrier-Coverage	
Instance:	Sensors $(x_1, r_1), \ldots, (x_n, r_n)$, $L \in \mathbb{R}$, $B \in \mathbb{R}$, and $k \in \mathbb{N}$.
Problem:	Does there exist a barrier coverage y of interval $[0, L]$ such that $\text{cost}(y) \leq B$ and $|\{i \mid x_i \neq y_i\}| \leq k$?

To show W[1]-hardness, we will reduce from Exact-Cover.

Exact-Cover	
Instance:	Universe $U = \{u_1, \ldots, u_m\}$, set of subsets $S = \{S_1, \ldots, S_n\} \subseteq 2^U$, and $k \in \mathbb{N}$.
Problem:	Does there exist $T = \{T_1, \ldots, T_l\} \subseteq S$ such that $l \leq k$, $\bigcup_{i=1}^{l} T_i = U$, and $T_i \cap T_j = \emptyset$ for $1 \leq i < j \leq l$?

A special case of Exact-Cover is the problem Perfect-Code, which was shown to be W[1]-hard when parameterized by k [8] (W[1]-membership was proved later [3]). Hence, Exact-Cover is W[1]-hard when parameterized by k. Actually, W[1]-hardness for Perfect-Code was shown for the case where one asks for a solution of size exactly k and not, as in our problem definition, a solution of size at most k. However, the proof can easily be adapted to our problem variant.

▶ **Theorem 8.** *k-move-Barrier-Coverage is* W[1]*-hard when parameterized by k.*

Proof. We reduce from Exact-Cover. Let $U = \{u_1, \dots, u_m\}$, $S = \{S_1, \dots, S_n\} \subseteq 2^U$, and k be an instance of Exact-Cover. We construct an instance (x_1, r_1), …, (x_n, r_n), L and B for k-move-Barrier-Coverage as follows. For $1 \leq i \leq n$ and $1 \leq j \leq m$ we define

$$e_{i,j} = \begin{cases} (n+1)^{j-1} & \text{if } u_j \in S_i, \\ 0 & \text{otherwise.} \end{cases} \qquad d_{i,j} = \begin{cases} (n+1)^{j+m} & \text{if } u_j \in S_i, \\ 0 & \text{otherwise.} \end{cases}$$

Our instance consists of intervals having radius $r_i = \frac{1}{2}\sum_{j=1}^{m} e_{i,j}$ and initial position $x_i = -r_i - \sum_{j=1}^{m} d_{i,j}$ for $1 \leq i \leq n$. Furthermore, we set $L = \sum_{j=1}^{m}(n+1)^{j-1}$ and $B = \sum_{j=1}^{m}(n+1)^{j+m} + k\sum_{j=1}^{m}(n+1)^{j-1}$. This reduction can be constructed in polynomial time. Figure 8 shows part of the reduction for a small example instance.

For the correctness, first assume that the Exact-Cover instance is a yes-instance, i.e., there exists $T = \{T_1, \dots, T_l\} \subseteq S$ such that $l \leq k$, $\bigcup_{i=1}^{l} T_i = U$, and $T_i \cap T_j = \emptyset$ for $1 \leq i < j \leq l$. Let $I \subseteq \{1, \dots, n\}$ be the indices of the intervals corresponding to sets $\{T_1, \dots, T_l\}$. By construction, $|I| \leq k$. We have to show that $[0, L]$ can be covered by moving only the intervals identified by I and that this solution has cost at most B. Since $\bigcup_{i=1}^{l} T_i = U$, for every $u_j \in U$ there exists exactly one $i \in I$ such that $u_j \in S_i$. Hence, $\sum_{i\in I} r_i = \frac{1}{2}\sum_{i\in I}\sum_{j=1}^{m} e_{i,j} = \frac{1}{2}\sum_{j=1}^{m}(n+1)^{j-1}$. Therefore, the total length of the selected intervals is exactly L and we can cover $[0, L]$.

Next, we consider the cost of this solution. Moving all the intervals identified by I to the beginning of the barrier, that is, to position $-r_i$ for interval $i \in I$ results in cost $\sum_{j=1}^{m}(n+1)^{j+m}$. Again, the argument is that for every $u_j \in U$ there exists exactly one $i \in I$ such that $u_j \in S_i$. Hence, $\sum_{i\in I} |-r_i - x_i| = \sum_{i\in I}\sum_{j=1}^{m} d_{i,j} = \sum_{j=1}^{m}(n+1)^{j+m}$. Additionally, the movement of these k intervals to the exact position on L can be bounded by kL resulting in a total cost of at most $\sum_{j=1}^{m}(n+1)^{j+m} + k\sum_{j=1}^{m}(n+1)^{j-1} = B$.

For the reverse direction, assume that there exists a barrier coverage y of interval $[0, L]$ such that $\text{cost}(y) \leq B$ and $|\{i \mid x_i \neq y_i\}| \leq k$. Let $I \subseteq \{1, \dots, n\}$ be the indices of the moved intervals. We have to show that $T = \{S_i \mid i \in I\}$ is a solution for the Exact-Cover instance, that is, every element $u \in U$ is contained exactly once in the sets of T. Assume towards a contradiction, that this is not true. Let $u_c \in U$ be the element with the highest index such that u_c is either not contained in T or it occurs more than once. Since elements $u_{c+1}, \dots, u_m$ occur exactly once, they contribute the length $\sum_{j=c+1}^{m}(n+1)^{j-1}$ towards covering $[0, L]$. Therefore, $\sum_{j=1}^{c}(n+1)^{j-1}$ remains to be covered. We have two cases:

- ***u_c* is not contained in *T*.** Then the maximum length we can cover is if every element $u_1, \dots, u_{c-1}$ is contained in every moved interval. Since $n \cdot \sum_{j=1}^{c-1}(n+1)^{j-1} = (n+1)^{c-1} - 1 < \sum_{j=1}^{c}(n+1)^{j-1}$, this is not enough and contradicts our assumption that y is a barrier coverage. Hence, u_c is contained in T.
- ***u_c* occurs in multiple moved intervals.** Since elements $u_{c+1}, \dots, u_m$ occur exactly once, they contribute $\sum_{j=c+1}^{m}(n+1)^{j+m}$ to the total cost just for moving the corresponding intervals to the beginning of the barrier. Since u_c occurs at least twice, it will contribute $2 \cdot (n+1)^{c+m}$ to the total cost just for moving the corresponding intervals to the beginning

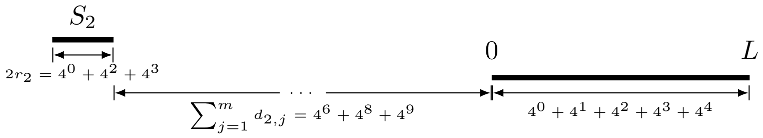


Figure 8 Part of the reduction from Exact-Cover to k-move-Barrier-Coverage for an instance $U = \{u_1, \dots, u_5\}$, $S = \{S_1, S_2, S_3\}$, with $S_2 = \{u_1, u_3, u_4\}$.

of the barrier. But $2(n+1)^{c+m} + \sum_{j=c+1}^{m}(n+1)^{j+m} = (n+1)^{c+m} + \sum_{j=c}^{m}(n+1)^{j+m}$, which is larger than our budget B, because $B \leq \sum_{j=1}^{m}(n+1)^{j+m} + n\sum_{j=1}^{m}(n+1)^{j-1} < \sum_{j=0}^{m}(n+1)^{j+m} = \sum_{j=0}^{c-1}(n+1)^{j+m} + \sum_{j=c}^{m}(n+1)^{j+m}$ and $\sum_{j=0}^{c-1}(n+1)^{j+m} < (n+1)^{c+m}$. Hence, u_c is contained exactly once in the sets of T, which contradicts our assumption. Therefore, T is indeed a solution for the Exact-Cover instance. ◀

Complementary to this W[1]-hardness result, we will show next, that the problem is fixed-parameter tractable when parameterized by the budget B. To this end we have to change the problem to restrict the input to integers instead of real numbers.

Barrier-Coverage
Instance: Sensors $(x_1, r_1), \dots, (x_n, r_n)$ with $x_i, r_i \in \mathbb{N}$ for each $i \in \{1, \dots, n\}$, $L \in \mathbb{N}$, and $B \in \mathbb{N}$.
Problem: Does there exist a barrier coverage y of interval $[0, L]$ such that $\mathrm{cost}(y) \leq B$?

▶ **Theorem 9.** *The* Barrier-Coverage *problem can be solved in* $2^{2B^2(B+1)} \cdot n^{O(1)}$ *time.*

Proof. Our algorithm is a branching algorithm, which, for any candidate sensor branches on which integer point in the gaps (empty intervals) to move this sensor to (or leave it at its original position). The crucial observations will be that we can give a bound on the number of candidate sensors we need to consider to move into the gaps as well as on the positions where they end up in the final configuration, both in terms of the budget B. The sum of the gaps on the barrier is at most B, otherwise we have a trivial no-instance. Given a gap G, we only need to consider intervals that are distance $\leq B$ left and right of G, since intervals further away cost too much to move them into G. Assume the interval of G is $[y_l, y_r]$. We consider the range left of G, that is $[y_l - B, y_l]$ (the right side is symmetrical). At each point p_i in $[y_l - B, y_l]$, we consider all the intervals whose right end equals p_i, that is intervals (x_j, r_j) with $x_j + r_j = p_i$. Let S_i denote the set of these intervals. We would like to branch on which intervals (if any) from S_i move into the gap G, but $|S_i|$ is not necessarily bounded by a function of B. Hence, we sort the intervals in S_i by length and consider only the $B+1$ longest ones. This is sound, since our budget allows us to move at most B intervals and additionally, an interval from S_i might need to remain stationary in order to cover p_i. Assume there exists an optimal solution in which interval $(x_j, r_j) \in S_i$ is moved to position $y_j \neq x_j$ and (x_j, r_j) is not among the top $B+1$ longest ones. Then at most $B-1$ of the longest intervals in S_i where moved. This leaves at least two remaining intervals among the $B+1$ many. Assume (x_k, r_k) is the shorter one of those two. Moving (x_k, r_k) the same distance to the right as (x_j, r_j) was moved, covers everything (x_j, r_j) was covering and has the same cost. Additionally, $[x_k - r_k, x_k + r_k]$ is still covered by the longer interval which we did not move. Hence, to conclude, we need to consider at most $B+1$ intervals for each of the B points left and right of a gap.

The only thing remaining, is to show that it suffices to consider integer points for the solution. The proof of this is deferred to the long version [9]. Therefore, for our branching algorithm, the total number of intervals to consider is bounded by B and their possible new positions is bounded by the budget B as well, which leads to fixed-parameter tractability in B because B decreases by at least one in each recursive call. ◀

5 Conclusion

We showed a $((2+\varepsilon)\rho + 2/\varepsilon)$-approximation for the barrier coverage problem for the case when the sensors initially are on a line containing the barrier. This works well when the ratio between the largest radius and the smallest radius is small, but in theory the difference could be arbitrarily large. However, we also proved that no polynomial time $\rho^{1-\varepsilon}$-approximation algorithm exists unless $\mathsf{P} = \mathsf{NP}$. There are still several open problems for this special case that would be interesting to pursue.

1. Improve the approximation ratio analysis of an order-preserving solution. Ideally, down to $\rho + O(1)$.
2. Determine if the problem is fixed-parameter tractable for parameter k when the interval radii are $1, 2, \ldots, k$.
3. Approximate the weighted version where each interval has a weight and we want to minimize $\sum_i w_i |x_i - y_i|$.

References

1 A. M. Andrews and H. Wang. Minimizing the aggregate movements for interval coverage. *Algorithmica*, 78(1):47–85, 2017. doi:10.1007/s00453-016-0153-8.

2 Anish Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. S. Kulkarni, M. Aramugam, L. Wang, M. G. Gouda, Y.-R. Choi, D. E. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: Elements of an extreme scale wireless sensor network. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 102–108, 2005.

3 M. Cesati. Perfect code is W[1]-complete. *Inform. Process. Lett.*, 81(3):163–168, 2002.

4 A. Chen, S. Kumar, and T.-H. Lai. Local barrier coverage in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 9(4):491–504, 2010.

5 D. Z. Chen, Y. Gu, J. Li, and H. Wang. Algorithms on minimizing the maximum sensor movement for barrier coverage of a linear domain. *Discrete & Computational Geometry*, 50(2):374–408, 2013.

6 J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia, and M. Yazdani. On minimizing the sum of sensor movements for barrier coverage of a line segment. In *Proceedings of the 9th International Conference Ad-HocMobile and Wireless Networks (ADHOC-NOW)*, pages 29–42, 2009.

7 S. Dobrev, S. Durocher, M. Eftekhari, K. Georgiou, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, S. Shende, and J. Urrutia. Complexity of barrier coverage with relocatable sensors in the plane. *Theoretical Computer Science*, 579:64 – 73, 2015.

8 R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theoretical Computer Science*, 141(1&2):109–131, 1995.

9 S. Gaspers, J. Gudmundsson, J. Mestre, and S. Rümmele. Barrier coverage with non-uniform lengths to minimize aggregate movements. *CoRR*, abs/1709.10285, 2017.

10 S. Kumar, T.-H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proc. of the 11th International Conference on Mobile Computing and Networking*, pages 284–298, 2005.

11 M. Mehrandish, L. Narayanan, and J. Opatrny. Minimizing the number of sensors moved on line barriers. In *IEEE Wireless Communications and Networking Conference*, pages 653–658, 2011.

12 D. Tao and T. Y. Wu. A survey on barrier coverage problem in directional sensor networks. *IEEE Sensors Journal*, 15(2):876–885, 2015.

13 F. Wu, Y. Gui, Z. Wang, X. Gao, and G. Chen. A survey on barrier coverage with sensors. *Frontiers of Computer Science*, 10(6):968–984, 2016.

Sorting with Recurrent Comparison Errors*†

Barbara Geissmann[1], Stefano Leucci[2], Chih-Hung Liu[3], and Paolo Penna[4]

1 **Department of Computer Science, ETH Zürich, Switzerland**
`barbara.geissmann@inf.ethz.ch`
2 **Department of Computer Science, ETH Zürich, Switzerland**
`stefano.leucci@inf.ethz.ch`
3 **Department of Computer Science, ETH Zürich, Switzerland**
`chih-hung.liu@inf.ethz.ch`
4 **Department of Computer Science, ETH Zürich, Switzerland**
`paolo.penna@inf.ethz.ch`

Abstract

We present a sorting algorithm for the case of *recurrent* random comparison *errors*. The algorithm essentially achieves simultaneously good properties of previous algorithms for sorting n distinct elements in this model. In particular, it runs in $O(n^2)$ time, the *maximum dislocation* of the elements in the output is $O(\log n)$, while the *total dislocation* is $O(n)$. These guarantees are the best possible since we prove that even randomized algorithms cannot achieve $o(\log n)$ maximum dislocation with high probability, or $o(n)$ total dislocation in expectation, regardless of their running time.

1998 ACM Subject Classification F.2.2 Sorting and Searching

Keywords and phrases sorting, recurrent comparison error, maximum and total dislocation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.38

1 Introduction

We study the problem of *sorting* n distinct elements under *recurrent* random comparison *errors*. In this classical model, each comparison is wrong with some fixed probability p, and correct with probability $1 - p$. The probability of errors are independent over all possible pairs of elements, but errors are recurrent: If the same comparison is repeated at any time during the computation, the result is always the same, i.e., always wrong or always correct.

In such a scenario not all sorting algorithms perform equally well in terms of the output, as some of them are more likely to produce “nearly sorted” sequences than others. To measure the quality of an output permutation in terms of sortedness, a common way is to consider the *dislocation* of an element, which is the difference between its position in the permutation and its true rank among all elements. Two criteria based on the dislocations of the elements are the *total dislocation* of a permutation, i.e., the sum of the dislocations of all n elements, and the *maximum dislocation* of any element in the permutation. Naturally, the running *time* remains an important criteria for evaluating sorting algorithms.

To the best of our knowledge, for recurrent random comparison errors, the best results with respect to *running time*, *maximum*, and *total dislocation* are achieved by the following two different algorithms:

* Research supported by SNSF (Swiss National Science Foundation, project number 200021_165524).
† A full version of the paper is available at `https://arxiv.org/abs/1709.07249`.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 38; pp. 38:1–38:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Our and previous results. The constant c in [2] depends on both the error probability p, and the success probability of the algorithm. For example, for a success probability of $1 - 1/n$, the analysis in [2] yields $c = \frac{110525}{(1/2-p)^4}$. The total dislocation in [12] follows trivially by the maximum dislocation, and no further analysis is given.

	Braverman and Mossel [2]	Klein et al [12]	Ours
# Steps	$O(n^{3+24c})$	$O(n^2)$	$O(n^2)$
Maximal Dislocation	w.h.p. $O(\log n)$	w.h.p. $O(\log n)$	w.h.p. $O(\log n)$
Total Dislocation	w.h.p. $O(n)$	w.h.p. $O(n \log n)$	in exp. $O(n)$

- Braverman and Mossel [2] give an algorithm which guarantees maximum dislocation $O(\log n)$ and total dislocation $O(n)$, both with high probability. The main drawback of this algorithm seems to be its running time, as the constant exponent can be rather high;
- Klein et al. [12] give a much faster $O(n^2)$-time algorithm which guarantees maximum dislocation $O(\log n)$, with high probability. They however do not provide any upper bound on the total dislocation, which by the previous result is obviously $O(n \log n)$.

In this paper we investigate whether it is possible to guarantee *all* of these bounds together, that is, if there is an algorithm with running time $O(n^2)$, maximum dislocation $O(\log n)$, and total dislocation $O(n)$.

1.1 Our contribution

We propose a new algorithm whose performance guarantees are essentially the best of the two previous algorithms (see Table 1). Indeed, our algorithm Window Sort takes $O(n^2)$ time and guarantees the maximum dislocation to be $O(\log n)$ with probability $1 - 1/n$ and the expected total dislocation to be $O(n)$. The main idea is to iteratively sort n elements by comparing each element with its neighbouring elements lying within a *window* and to halve the window size after every iteration. In each iteration, each element is assigned a rank based on the local comparisons, and then they are placed according to the computed ranks.

Our algorithm is inspired by Klein et al.'s algorithm [12] which distributes elements into *buckets* according to their computed ranks, compares each element with elements in neighboring buckets to obtain a new rank, and halves the range of a bucket iteratively. Note however that the two algorithms operate in a different way, essentially because of the following key difference between *bucket* and *window*. The number of elements in a bucket is not fixed, since the computed rank of several elements could be the same. In a window, instead, the number of elements is *fixed*. This property is essential in the analysis of the total dislocation of Window Sort, but introduces a potential *offset* between the computed rank and the computed position of an element. Our analysis consists in showing that such an offset is sufficiently small, which we do by considering a number of "delicate" conditions that the algorithm should maintain throughout its execution with sufficiently high probability.

We first describe a standard version of our algorithm which achieves the afore mentioned bounds for any error probability $p < 1/32$. We then improve this result to $p < 1/16$ by using the idea of shrinking the window size at a different rate. An experimental evaluation of our algorithms –which, due to space limitations, can be found in the full version of the paper– shows that the performance of the standard version is significantly better than the theoretical guarantees. In particular, the experiments suggest that the expected total dislocation is $O(n)$ for $p < 1/5$, while the maximum dislocation is $O(\log n)$ for $p < 1/4$.

In addition, we prove that no sorting algorithm can guarantee the maximum dislocation to be $o(\log n)$ with high probability, and no sorting algorithm can guarantee the expected total dislocation to be $o(n)$.

1.2 Further Related Work on Sorting with Comparison Errors

Computing with errors is often considered in the framework of a two-person game called *Rényi-Ulam Game*: The *responder* thinks of an object in the search space, and the *questioner* has to find it by asking questions to which the responder provides answers. However, some of the answers are incorrect on purpose; the responder is an *adversarial lier*. These games have been extensively studied in the past on various kinds of search spaces, questions, and errors; see Pelc's survey [14] and Cicalese's monograph [3].

Feige et al. [7] studied several comparison based algorithms with independent random errors where the error probability of a comparison is less than half, the repetitions of an comparison can obtain different outcomes, and all the comparisons are independent. They required the reported solution to be correct with a probability $1 - q$, where $0 < q < 1/2$, and proved that for sorting, $O(n \log(n/q))$ comparisons suffice, which gives also the running time. In the same model, sorting by random swaps represented as Markovian processes have been studied under the question of the number of inversions (reversed pairs) [8, 9], which is within a constant factor of the total dislocation [15]. Karp and Kleinberg [11] studied a noisy version of the classic binary search problem, where elements cannot be compared directly. Instead, each element is associated with a coin that has an unknown probability of observing heads when tossing it and these probabilities increase when going through the sorted order.

For recurring errors, Coppersmith and Rurda [4] studied a simple algorithm that gives a 5-approximation on the weighted feedback arc set in tournaments (FAST) problem if the weights satisfy probability constraints. The algorithm consists of ordering the elements based on computed ranks, which for unweighted FAST are identical to our computed ranks. Damaschke [6] also gave a *subquadratic* time algorithm returning a sequence with $O(k)$ inversions when at most k errors can occur. Alonso et al. [1] and Hadjicostas and Lakshamanan [10] studied Quicksort and recursive Mergesort, respectively, with random comparison errors.

Paper organization

The rest of this paper is organized as follows. We present the Window Sort algorithm in Section 2 and analyze the maximum and total dislocation in Section 3 and Section 4, respectively. Then, we explain how to modify Window Sort to allow larger error probabilities in Section 5. Additionally, we provide a lower bounds on both the maximum and average dislocation for any sorting algorithm (due to space limitations the corresponding proofs are omitted and can be found in the full version of the paper).

2 Window Sort

Window Sort consists of multiple iterations of the same procedure: Starting with a permutation σ and a *window size* w, we compare each element x in σ with its left $2w$ and right $2w$ adjacent elements (if they exist) and count its *wins*, i.e., the number of times a comparison outputs x as the larger element. Then, we obtain the *computed rank* for each element based on its *original position* in σ and its wins: if $\sigma(x)$ denotes the original position of x in σ, the computed rank of x equals $\max\{0, \sigma(x) - 2w\}$ plus the number of its wins. And we get a new permutation σ' by placing the elements ordered by their computed ranks.

```
Algorithm 1: WINDOW SORT (on a permutation $\sigma$ on $n$ elements)
  Initialization: The initial window size is $w = n/2$. Each element $x$ has two
  variables $wins(x)$ and $computed\_rank(x)$ which are set to zero.
  repeat
      1. foreach $x$ at position $l = 1, 2, 3, \ldots, n$ in $\sigma$ do
             foreach $y$ whose position in $\sigma$ is in $[l - 2w, l - 1]$ or in $[l + 1, l + 2w]$ do
                 if $x > y$ then
                     $wins(x) = wins(x) + 1$
             $computed\_rank(x) = \max\{l - 2w, 0\} + wins(x)$
      2. Place the elements into $\sigma'$ ordered by non-decreasing $computed\_rank$,
         break ties arbitrarily.
      3. Set all $wins$ to zero, $\sigma = \sigma'$, and $w = w/2$.
  until $w < 1$;
```

Finally, we set $w' = w/2$ and start a new iteration on σ' with window size w'. In the very first iteration, $w = n/2$. We formalize WINDOW SORT in Algorithm 1.

In the following, w.l.o.g. we assume to sort elements $\{1, \ldots, n\}$, i.e., we refer to both an element x and its rank by x. Let σ denote the permutation of the elements at the beginning of the current iteration of WINDOW SORT and let σ' denote the permutation obtained after this iteration (i.e., the permutation on which the next iteration performs). Similarly, let w and $w' = w/2$ denote the window size of the current and the next iteration. Furthermore, let π denote the sorted permutation. We define four important terms for an element x in σ:

- *Current/Original position*: The position of x in σ: $\sigma(x)$
- *Computed rank*: The current position of x minus $2w$ (zero if negative) plus its number of wins: $computed\_rank(x)$
- *Computed position*: The position of x in σ': $\sigma'(x)$

▶ **Theorem 1.** WINDOW SORT *takes $O(n^2)$ time.*

Proof. Consider the three steps in Algorithm 1. The number of comparisons in an iteration in the outer loop is $4w$, for w the current size of the window. Therefore, the first step needs $O(nw)$ time. For the second step we could apply for instance Counting Sort (see e.g. [5]), which takes $O(n)$ time, since all computed ranks lie between zero and n. Thus, the total running time is upper bounded by $\sum_{i=0}^{\infty} O(\frac{4n^2}{2^i}) = O(8n^2)$. ◀

2.1 Preliminaries

We first introduce a condition on the errors in comparisons between an element x and a fixed subset of elements which depends only on the window size w.

▶ **Definition 2.** We define $ERRORS(x, w)$ as the set of errors among the comparisons between x and every $y \in [x - 4w, x + 4w]$.

▶ **Theorem 3.** WINDOW SORT *returns a sequence of maximum dislocation at most $9w^\star$ whenever the initial comparisons are such that*

$$|ERRORS(x, w)| \le w/4 \tag{1}$$

hold for all elements x and for all $w = n/2, n/4, \ldots, 2w^\star$.

The proof of this theorem follows in the end of this section. In the analysis, we shall prove the following:

- If the computed rank of each element is close to its (true) rank, then the dislocation of each elements is small (Lemma 7).
- The computed rank of each element is indeed close to its (true) rank if the number of errors involving the element under consideration is small (Lemma 4).
- The number of positions an element can move in further iterations is small (Lemma 8).

We now introduce a condition that implies Theorem 3: Throughout the execution of Window Sort we would like every element x to satisfy the following condition:

($*$) For window size w, the dislocation of x is at most w.

We also introduce two further conditions, which essentially relax the requirement that all elements satisfy ($*$). The first condition justifies the first step of our algorithm, while the second condition restricts the range of elements that get compared with x in some iteration:

($\bullet$) For window size w, element x is larger (smaller) than all the elements lying apart by more than $2w$ positions left (right) of x's original position.

($\circ$) For window size w, x and its left $2w$ and right $2w$ adjacent elements satisfy condition ($*$).

Note that if ($*$) holds for all elements, then ($\bullet$) and ($\circ$) also hold for all elements. For elements that satisfy both ($\bullet$) and ($\circ$), the computed rank is close to the true rank if there are few errors in the comparisons:

▶ **Lemma 4.** *For every window size w, if an element x satisfies satisfy both* ($\bullet$) *and* ($\circ$)*, then the absolute difference between the computed rank and its true rank is bounded by*

$$|computed\_rank(x) - x| \leq |ERRORS(x, w)| \ .$$

Proof. This follows immediately from condition ($\bullet$). ◀

We now consider the difference between the computed rank and the computed position of an element, which we define as the *offset* of this element. Afterwards, we consider the difference between the original position and the computed position of an element.

▶ **Fact 5.** *Observe that by Step 1 of the algorithm it holds that, for every permutation σ, every window size w and every element x, the difference between $\sigma(x)$ and the computed rank of x is at most $2w$, $|computed\_rank(x) - \sigma(x)| \leq 2w$.*

▶ **Lemma 6.** *For any permutation of n elements and for each element x, if the difference between the computed rank and x is at most m for every element, then the difference between the computed position and x is at most $2m$ for every element.*

The proof of this lemma is analogue to the proof of Lemma 7 below.

▶ **Lemma 7.** *For every permutation σ and window size w, the offset of every element x is at most $2w$, $|computed\_rank(x) - \sigma'(x)| \leq 2w$.*

Proof. Let the computed rank of x be k. The computed position $\sigma'(x)$ is larger than the number of elements with computed rank smaller than k, and at most the number of elements with computed ranks at most k. By Fact 5, every element y with $\sigma(y) < k - 2w$ has a computed rank smaller than k, and every element y with $\sigma(y) > k + 2w$ has a computed rank larger than k. ◀

▶ **Lemma 8.** *Consider a generic iteration of the algorithm with permutation σ and a window size w. In this iteration, the position of each element changes by at most $4w$. Moreover, the position of each element changes by at most $8w$ until the algorithm terminates.*

Proof. By Fact 5, Lemma 7 and triangle inequality,

$$|\sigma(x)-\sigma'(x)| \leq |\mathit{computed\_rank}(x)-\sigma(x)|+|\mathit{computed\_rank}(x)-\sigma'(x)| \leq 2w+2w = 4w.$$

Since w is halved after every iteration, the final difference is at most $\sum_{i=0}^{\infty} \frac{4w}{2^i} = 8w$. ◀

Finally, we conclude Theorem 3 and show that the dislocation of an element is small if the number of errors is small:

Proof of Theorem 3. Consider an iteration of the algorithm with current window size w. We show that, if $(*)$ holds for all elements in the current iteration, then (1) implies that $(*)$ also holds for all elements in the next iteration, i.e., when the window size becomes $w/2$. In order for $(*)$ to hold for the next iteration, the computed *position* of each element should differ from the true rank by at most $w/2$,

$$|\sigma'(x) - x| \leq w/2 \ .$$

By Lemma 6, it is sufficient to require that the computed *rank* of each element differs from its true rank by at most $w/4$,

$$|\mathit{computed\_rank}(x) - x| \leq w/4$$

By Lemma 4, the above inequality follows from the hypothesis $|\mathit{ERRORS}(x, w)| \leq w/4$.

We have thus shown that after the iteration with window size $2w^*$, all elements have dislocation at most w^*. By Lemma 8, the subsequent iterations will move each element by at most $8w^*$ positions. ◀

▶ Remark. If we care only about the maximum dislocation, then we could obtain a better bound of w by simply stopping the algorithm at the iteration where the window size is w (for a w which guarantees the condition above with high probability). In order to bound also the total dislocation, we let the algorithm continue all the way until window size $w = 1$. This will allow us to show that the total dislocation is linear in expectation.

3 Maximum Dislocation

In this section we give a bound on the maximum dislocation of an element after running Window Sort on n elements. We prove that it is a function of n and of the probability p that a single comparison fails. Our main result is the following:

▶ **Theorem 9.** *For a set of n elements, with probability $1 - 1/n$, the maximum dislocation after running* Window Sort *is $9 \cdot f(p) \cdot \log n$ where*

$$f(p) = \begin{cases} \frac{400p}{(1-32p)^2} & \text{\textit{for}} \quad 1/64 < p < 1/32, \\ \frac{4}{\ln\left(\frac{1}{32p}\right)-(1-32p)} & \text{\textit{for}} \quad 1/192 < p \leq 1/64, \\ 6 & \text{\textit{for}} \quad p \leq 1/192. \end{cases}$$

It is enough to prove that the condition in Theorem 3 holds for all $w \geq 2f(p)\log n$ with probability at least $1 - 1/n$.

▶ **Lemma 10.** *For every fixed element x and for every fixed window size $w \geq 2f(p)\log n$, the probability that*

$$|ERRORS(x, w)| > w/4 \tag{2}$$

is at most $1/n^3$.

By the union bound, the probability that (2) holds for some x and for some w is at most $1/n$. That is, the condition of Theorem 3 holds with probability at least $1 - 1/n$ for all $w \geq 2w^\star = 2f(p)\log n$, which then implies Theorem 9.

3.1 Proof of Lemma 10

Since each comparison fails with probability p independently of the other comparisons, the probability that the event in (2) happens is equal to the probability that at least $w/4$ errors occur in $8w$ comparisons. We denote such probability as $\Pr(w)$, and show that $\Pr(w) \leq 1/n^3$.

We will make use of the following standard Chernoff Bounds (see for instance in [13]):

▶ **Theorem 11** (Chernoff Bounds). *Let $X_1, \cdots, X_n$ be independent Poisson trials with $\Pr(X_i) = p_i$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathrm{E}[X]$. Then the following bounds hold:*

$$(i)\ \text{For } 0 < \delta < 1, \qquad \Pr(X \geq (1+\delta)\mu) \leq e^{-\frac{\mu\delta^2}{3}}, \tag{3}$$

$$(ii)\ \text{For any } \delta > 0, \qquad \Pr(X \geq (1+\delta)\mu) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu, \tag{4}$$

$$(iii)\ \text{For } R \geq 6\mu, \qquad \Pr(X \geq R) \leq 2^{-R}. \tag{5}$$

▶ **Lemma 12.** *The probability $\Pr(w)$ (at least $w/4$ errors occur in $8w$ comparisons) satisfies*

$$\Pr(w) \leq \begin{cases} e^{-\frac{w(1-32p)^2}{384p}} & \text{for} \quad 1/64 < p < 1/32, \\ \left(\dfrac{e^{\frac{1-32p}{32p}}}{\left(\frac{1}{32p}\right)^{\frac{1}{32p}}}\right)^{8wp} & \text{for} \quad 1/192 < p \leq 1/64, \\ 2^{-\frac{w}{4}} & \text{for} \quad p \leq 1/192. \end{cases}$$

Proof. Let the random variable X denote the number of errors in the outcome of $8w$ comparisons. Clearly, $\mathrm{E}[X] = 8wp$, and

$$\Pr(w) = \Pr\left[X \geq \frac{w}{4}\right] = \Pr\left[X \geq \frac{\mathrm{E}[X]}{32p}\right] = \Pr\left[X \geq \left(1 + \frac{1-32p}{32p}\right)\mathrm{E}[X]\right].$$

Let $\delta = \frac{1-32p}{32p}$. If $1/64 < p < 1/32$, then $0 < \delta < 1$, and by Theorem 11, case (i), we have

$$\Pr(w) \leq e^{-\frac{1}{3}\delta^2\mu} \leq e^{-\frac{w\cdot(1-32p)^2}{384p}}.$$

Similarly, if $p \leq 1/64$, then $\delta \geq 1$, and by Theorem 11, case (ii), we have

$$\Pr(w) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu \leq \left(\frac{e^{\frac{1-32p}{32p}}}{\left(\frac{1}{32p}\right)^{\frac{1}{32p}}}\right)^{8wp}.$$

If $p \leq 1/192$, then $w/4 \geq 48wp = 6\,\mathrm{E}[X]$, and by Theorem 11 case (iii), $\Pr(w) \leq 2^{-\frac{w}{4}}$. ◀

▶ **Lemma 13.** *If $w \geq 2\,f(p)\log n$, with $n \geq 1$ and $f(p)$ as in Theorem 9, then* $\Pr(w) \leq 1/n^3$.

Proof. We show the first case, the other two are similar. If $1/64 < p < 1/32$, by Lemma 12,

$$\Pr(w) \leq e^{-\frac{w\cdot(1-32p)^2}{384p}} \leq e^{-\frac{800}{384}\log n} \leq e^{-3\lg n} \leq 1/n^3.$$ ◀

4 Total Dislocation

In this section, we prove that Window Sort orders n elements such that their total dislocation is linear in n times a factor which depends only on p:

▶ **Theorem 14.** *For a set of n elements, the expected total dislocation after running* Window Sort *is at most $n \cdot 60\, f(p) \log f(p)$.*

The key idea is to show that for an element x, only $O(w)$ elements adjacent to its (true) rank matter in all upcoming iterations. If this holds, it is sufficient to keep the following *weak invariant* for an element x throughout all iterations:

(♣) This invariant consists of three conditions that have to be satisfied:
- **(a)** x satisfies condition **(∗)**.
- **(b)** All elements with original position in $[x - 12w, x + 12w]$ satisfy condition **(∗)**.
- **(c)** All elements with original position in $[x - 10w, x + 10w]$ satisfy condition **(•)**.

Note that if x satisfies **(♣)**, all elements lying in $[x - 10w, x + 10w]$ satisfy both **(•)** and (◦).

The rest of this section is structured as follows: First we derive several properties of the weak invariant, then we prove an $n \log \log n$ bound on the expected total dislocation, and finally we extend the proof to achieve the claimed linear bound.

4.1 Properties of the Weak Invariant

We start with the key property of the weak invariant **(♣)** for some element x.

▶ **Lemma 15.** *Let σ be the permutation of n elements and w be the window size of some iteration in* Window Sort*. If the weak invariant* **(♣)** *holds for an element x in σ and the computed rank of every element y with $\sigma(y) \in [x - 10w, x + 10w]$ differs from y by at most $w/4$, then* **(♣)** *still holds for x in the permutation σ' of next iteration with window size $w/2$.*

Proof. Consider the set X of all elements y with *computed_rank*$(y) \in [x - 8w, x + 8w]$. Their computed ranks differ from their original positions by at most $2w$. Thus, all these elements are in the set $Y \supseteq X$ of all elements whose original positions are in $[x - 10w, x + 10w]$. By the assumption of the lemma, for each element $y \in Y$, $|computed\_rank(y) - y| \leq w/4$. Using the same reasoning as in the proof of Lemma 7, we conclude that

$$\text{for each element } y \in X,\ |\sigma'(y) - y| \leq w/2. \qquad \text{(A)}$$

Consider the set Z of all elements y with $\sigma'(y) \in [x - 6w, x + 6w]$. By Lemma 7, their computed ranks lie in $[x - 8w, x + 8w]$, thus $Z \subseteq X$, and by (A), $|\sigma'(y) - y| \leq w/2$ for each $y \in Z$. Thus, the second condition of **(♣)** holds for the next iteration.

We continue with the third condition. Consider the set $T \subseteq Z$ of all elements y with $\sigma'(y) \in [x - 5w, x + 5w]$. By the assumptions of the lemma, $y \in [x - 5w - w/2, x + 5w + w/2]$ and $\sigma(y) \in [x - 5w - 3w/2, x + 5w + 3w/2]$ for all $y \in T$. It is sufficient to show that every element in T is larger (or smaller) than all elements whose computed positions are smaller than $x - 6w$ (or larger than $x + 6w$), the rest follows from the second condition. We show the former case, the latter is symmetric. We distinguish three subcases: elements $y \in T$ with $\sigma'(y) < x - 6w$ and with $\sigma(y)$ (i) smaller than $x - 12$, (ii) between $x - 12$ and $x - 10w - 1$, or (iii) between $x - 10w$ and $x - 4w - 1$.

(i) This case follows immediately from the third condition of (♣).

(ii) This case follows immediately from the second condition of (♣).

(iii) By the assumption of our lemma, $|computed\_rank(y) - y| \leq w/4$. Thus, if the computed rank r of such an element y is smaller than $x - 6w$, then $y < x - 6w + w/4$. Otherwise, if $r \geq kx - 6w$, then by (A), $|y - \sigma'(y)| \leq w/2$. Thus, $y < x - 6w + w/2$.

Since we assume (♣) for x, $\sigma(x) \in [x-w, x+w]$ and $computed\_rank(x) \in [x-3w, x+3w]$. By Lemma 7, $\sigma'(x) \in [k - 5w, k + 5w]$, and thus $x \in Z$, which implies that the first condition of (♣) will still be satisfied for x for the next iteration. This concludes the proof. ◀

Next, we adopt Lemma 13 to analyze the probability of keeping the weak invariant for an element x and an arbitrary window size through several iteration of Window Sort.

▶ **Lemma 16.** *Consider an iteration of* Window Sort *on a permutation σ on n elements such that the window size is $w \geq 2\,f(p)\log w$, where $f(p)$ is defined as in Theorem 9. If the weak invariant* (♣) *for an element x holds, then with probability at least $1 - 42/w^2$,* (♣) *still holds for x when the window size is $f(p)\log w$ (after some iterations of* Window Sort*).*

Proof. By Lemma 15, the probability that (♣) fails for x before the next iteration is $(20w + 1) \cdot \Pr(w)$. Let $r = \log(\frac{w}{2\,f(p) log w})$, then the probability that (♣) fails for x during the iterations from window size w to window size $f(p) \cdot \log w$ is

$$\sum_{i=0}^{r}\left(\frac{20w}{2^i} + 1\right) \cdot \Pr\left(\frac{w}{2^i}\right) \leq \sum_{i=0}^{r}\left(\frac{21w}{2^i}\right) \cdot \Pr\left(2\,f(p)\log w\right) \leq 42w \cdot \Pr\left(2\,f(p)\log w\right),$$

where the first inequality is by fact that $\Pr(w)$ increases when w decreases. By Lemma 13, $\Pr(2\,f(p)\log w) \leq 1/w^3$, leading to the statement. ◀

4.2 Double Logarithmic Factor (Main Idea)

Given that Window Sort guarantees maximum dislocation at most $9\,f(p)\log n$ with probability at least $(1 - 1/n)$ (Theorem 9), this trivially implies that the expected total dislocation is at most $O(f(p)\log n)$. More precisely, the expected dislocation is at most

$$(1/n) \cdot n \cdot n + (1 - 1/n) \cdot n \cdot 9\,f(p)\log n \leq n \cdot (1 + 9\,f(p)\log n),$$

since a fraction $1/n$ of the elements is dislocated by at most n, while the others are dislocated by at most $9 \cdot f(p)\log n$.

We next describe how to improve this to $O(f(p)\log\log n)$ by considering in the analysis *two phases* during the execution of the algorithm:

- **Phase 1:** The first phase consists of the iterations up to window size $w = f(p)\log n$. With probability at least $(1 - 1/n)$ all elements satisfy $(*)$ during this phase.
- **Phase 2:** The second phase consists of the executions up to window size $w' = f(p)\log w$. If all elements satisfied $(*)$ at the end of the previous phase, then the probability that a fixed element violates (♣) during this second phase is at most $42/w^2$.

More precisely, by Theorem 9 and the proof of Theorem 3, the probability that $(*)$ holds for all elements when the window size is $f(p)\log n$ is at least $(1 - 1/n)$. We thus restart our analysis with $w = f(p)\log n$ and the corresponding permutation σ. Assume an element x satisfies (♣). By Lemma 16, the probability that (♣) fails for x before the window size is $f(p)\log w$ is at most $42/w^2$. By Lemma 8, an element moves by at most $8w$ positions from its original position, which is at most w apart from its true rank. Therefore, the expected dislocation of an element x is at most

$$(1/n) \cdot n + 42/w^2 \cdot 9w + 9\,f(p)\log w = O(1) + 9\,f(p)\log(f(p)\log n)\ ,$$

where the equality holds for sufficiently large n because $w = f(p)\log n$.

4.3 Linear Dislocation (Proof of Theorem 14)

In this section, we apply a simple idea to decrease the upper bound on the expected total dislocation after running Window Sort on n elements to $60\, f(p) \log f(p)$. We recurse the analysis from the previous Section 4.2 for several *phases*: Roughly speaking, an *iteration* in Window Sort halves the window size, a *phase* of iterations logarithmizes the window size.

- **Phase 1**: Iterations until the window size is $f(p) \log n$.
- **Phase 2:** Subsequent iterations until the window size is $f(p) \cdot \log(f(p) \log n)$.
- **Phase 3:** Subsequent iterations until the window size is $f(p) \cdot \log(f(p) \cdot \log(f(p) \log n))$.
- ...

We bound the expected dislocation of an element x, and let w_i denote the window size after the i-th phase. We have $w_0 = n$, $w_1 = f(p) \log n$, $w_2 = f(p) \log(f(p) \log n)$, and

$$w_{i+1} = f(p) \log w_i, \tag{6}$$

if $i \geq 1$ and $w_i \geq 2\, f(p) \log w_i$. Any further phase would just consist of a single iteration. In the remaining of this section, we only consider phases i for which Equation (6) is true, and we call them the *valid phases*.

By Lemma 16, if the weak invariant (♣) holds for x and window size w_{i-1}, the probability that it still holds for window size w_i is at least $1 - 42/w_{i-1}^2$. Similarly to the analysis in the Section 4.2, we get that a valid phase $i \geq 1$ contributes to the expected dislocation of x by

$$42/w_{i-1}^2 \cdot 9w_{i-1} = 378/w_{i-1}\,. \tag{7}$$

If we stop our analysis after c valid phases, then by (7) and Lemma 8, the expected dislocation of any element x is at most

$$\sum_{i=0}^{c-1} 378/w_i + 9w_c \leq 378/w_c + 9w_c\,. \tag{8}$$

The inequality holds since $\frac{w_{i-1}}{w_i} \leq 2$ for $1 < i < c$. We next define c such that phase c is valid and w_c only depends on $f(p)$. The term $\frac{w_{i-1}}{f(p) \log w_{i-1}} \geq 2$ holds for every valid phase i and decreases with increasing i. For instance for $w = 6f(p) \log f(p)$:

$$\frac{w}{f(p) \log w} = \frac{6\, f(p) \log f(p)}{f(p) \log(6\, f(p) \log f(p))} \geq \frac{6\, \log f(p)}{3\, \log f(p)} \geq 2\,.$$

Therefore, if we choose c such that $w_{c-1} \geq 6\, f(p) \log f(p) > w_c$, we can use that $f(p) \geq 6$ and upper bound w_c by

$$w_c = f(p) \log w_{c-1} \geq f(p) \log(6\, f(p) \log f(p)) \geq 6 \log(36 \log 6) \geq 39\ . \tag{9}$$

Equations (8) and (9) and Lemma 8 imply the following:

▶ **Lemma 17.** *The expected dislocation of each element x after running* Window Sort *is at most* $378/w_c + 9w_c < 10 + 9w_c \leq 10w_c \leq 60\, f(p) \log f(p)$.

This immediately implies Theorem 14.

5 Extension

The reason why we require the error probability p to be smaller than $1/32$ is to analyze the probability that at most $w/4$ errors occur in $8w$ comparisons, for $w \geq 1$. This bound on the number of errors appears since we halve the window size in every iteration. If we let the window size shrink by another rate $1/2 < \alpha < 1$, the limit of p will also change:

First, the running time of the adapted Window Sort will become $O(\frac{1}{1-\alpha}n^2)$. Second, for any permutation σ and window size w, in order to maintain condition $(*)$ for an element x, its computed position should differ from x by at most αw, and thus *computed_rank*(x) should differ from x by at most $\alpha w/2$.

Our new issue is thus the probability that at most $\alpha w/2$ errors occur in $8w$ comparisons: Since the expected number of errors is $8wp$, we have $\frac{\alpha w}{2} = \frac{\alpha}{16p} \cdot 8wp = (1 + \frac{\alpha - 16p}{16p}) \cdot 8wp$, and by the reasoning of Lemma 12, we have $\frac{\alpha-16p}{16p} > 0$, thus $p < \alpha/16$. (Note that $f(p)$ should change accordingly.)

Finally, the number of windows for the weak invariant should also change accordingly. Let m be the number of windows that matter for the weak invariant ($m = 12$ when $\alpha = 1/2$). According to the analysis in Section 4.1, we have $m - 6 \geq \alpha m$, implying that $m \geq \frac{6}{1-\alpha}$. Of course, the constant inside the linear expected total dislocation will also change accordingly.

▶ **Theorem 18.** *For an error probability $p < \alpha/16$, where $1/2 < \alpha < 1$, modified* Window Sort *on n elements takes $O(\frac{1}{1-\alpha}n^2)$ time, has maximum dislocation $9\,g(p,\alpha)\log n$ with probability $1 - 1/n$, and expected total dislocation $n \cdot (9 + \frac{2}{1-\alpha}) \cdot 6\,g(p,\alpha)\log g(p,\alpha)$, where*

$$g(p,\alpha) = \begin{cases} \frac{100p}{(\alpha-16p)^2} & \textit{for} \quad \alpha/32 < p < \alpha/16\ , \\ \frac{4}{(\ln(\alpha/16p)) - (\alpha - 16p)} & \textit{for} \quad \alpha/96 < p \leq \alpha/32\ , \\ 6 & \textit{for} \quad p \leq \alpha/96\ . \end{cases}$$

6 A lower bound on the maximum dislocation

In this section we prove a lower bound on both the maximum and the average dislocation that can be achieved w.h.p. by any sorting algorithm.

The following lemma – whose proof is omitted – is a key ingredient in our lower bounds:

▶ **Lemma 19.** *Let $x, y \in S$ with $x < y$. Let A be any (possibly randomized) algorithm. On a random instance, the probability that A returns a permutation in which elements x and y appear the wrong relative order is at least $\frac{1}{2}\left(\frac{p}{1-p}\right)^{2(y-x)-1}$.*

As a first consequence of the previous lemma, we obtain the following:

▶ **Theorem 20.** *No (possibly randomized) algorithm can achieve maximum dislocation $o(\log n)$ with high probability.*

Proof. By Lemma 19, any algorithm, when invoked on a random instance, must return a permutation ρ in which elements 1 and $h = \left\lfloor \frac{\log n}{2 \log 1-p/p} \right\rfloor$ appear in the wrong order with a probability larger than $\frac{1}{n}$. When this happens, at least one of the following two conditions holds: (i) the position of element 1 in ρ is at least $\lceil \frac{h}{2} \rceil$; or (ii) the position of element h in ρ is at most $\lfloor \frac{h}{2} \rfloor$. In any case, the maximum dislocation must be at least $\frac{h}{2} - 1 = \Omega(\log n)$. ◀

Finally, we are also able to prove a lower bound to the total dislocation (proof omitted due to space limitations).

▶ **Theorem 21.** *No (possibly randomized) algorithm can achieve expected total dislocation $o(n)$.*

References

1 Laurent Alonso, Philippe Chassaing, Florent Gillet, Svante Janson, Edward M Reingold, and René Schott. Quicksort with unreliable comparisons: a probabilistic analysis. *Combinatorics, Probability and Computing*, 13(4-5):419–449, 2004.

2 Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms*, pages 268–276, 2008. arXiv:0707.1051.

3 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.

4 Don Coppersmith, Lisa K. Fleischer, and Atri Rurda. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *ACM Trans. Algorithms*, 6(3):55:1–55:13, July 2010. doi:10.1145/1798596.1798608.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.).* MIT Press, 2009.

6 Peter Damaschke. The solution space of sorting with recurring comparison faults. In *Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*, pages 397–408, 2016.

7 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with Noisy Information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

8 Tomáš Gavenčiak, Barbara Geissmann, and Johannes Lengler. Sorting by swaps with noisy comparisons. In *To appear in: Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '17, 2017.

9 B. Geissmann and P. Penna. Sort well with energy-constrained comparisons. *ArXiv e-prints*, October 2016. arXiv:1610.09223.

10 Petros Hadjicostas and KB Lakshmanan. Recursive merge sort with erroneous comparisons. *Discrete Applied Mathematics*, 159(14):1398–1417, 2011.

11 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 881–890, 2007.

12 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In *Proceedings of the19th Annual European Symposium on Algorithm*, pages 736—-747, 2011.

13 Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

14 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

15 R. L. Graham Persi Diaconis. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):262–268, 1977.

Dominance Product and High-Dimensional Closest Pair under L_∞*

Omer Gold[1] and Micha Sharir[2]

1 Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
omergold@post.tau.ac.il

2 Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
michas@post.tau.ac.il

Abstract

Given a set S of n points in $\mathbb{R}^d$, the Closest Pair problem is to find a pair of distinct points in S at minimum distance. When d is constant, there are efficient algorithms that solve this problem, and fast approximate solutions for general d. However, obtaining an *exact* solution in very high dimensions seems to be much less understood. We consider the high-dimensional L_∞ Closest Pair problem, where $d = n^r$ for some $r > 0$, and the underlying metric is L_∞.

We improve and simplify previous results for L_∞ Closest Pair, showing that it can be solved by a deterministic strongly-polynomial algorithm that runs in $O(DP(n,d)\log n)$ time, and by a randomized algorithm that runs in $O(DP(n,d))$ expected time, where $DP(n,d)$ is the time bound for computing the *dominance product* for n points in $\mathbb{R}^d$. That is a matrix D, such that $D[i,j] = \left|\{k \mid p_i[k] \le p_j[k]\}\right|$; this is the number of coordinates at which p_j dominates p_i. For *integer* coordinates from some interval $[-M, M]$, we obtain an algorithm that runs in $\tilde{O}\left(\min\{Mn^{\omega(1,r,1)}, DP(n,d)\}\right)$ time[1], where $\omega(1,r,1)$ is the exponent of multiplying an $n \times n^r$ matrix by an $n^r \times n$ matrix.

We also give slightly better bounds for $DP(n,d)$, by using more recent rectangular matrix multiplication bounds. Computing the dominance product itself is an important task, since it is applied in many algorithms as a major black-box ingredient, such as algorithms for APBP (all pairs bottleneck paths), and variants of APSP (all pairs shortest paths).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Closest Pair, Dominance Product, L_∞, Matrix Multiplication

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.39

1 Introduction

Finding the closest pair among a set of n points in $\mathbb{R}^d$ was among the first studied algorithmic geometric problems, considered at the origins of computational geometry; see [19, 21]. The distance between pairs of points is often measured by the L_τ metric, for some $1 \le \tau \le \infty$, under which the distance between the points $p_i = (p_i[1], \ldots, p_i[d])$ and $p_j = (p_j[1], \ldots, p_j[d])$ is $\mathrm{dist}_\tau(p_i, p_j) = \|p_i - p_j\|_\tau = \left(\sum_{k=1}^{d} \left|p_i[k] - p_j[k]\right|^\tau\right)^{1/\tau}$, for $\tau < \infty$, and $\mathrm{dist}_\infty(p_i, p_j) =$

* Work on this paper has been supported by Grant 892/13 from the Israel Science Foundation, by Grant 2012/229 from the U.S.-Israeli Binational Science Foundation, by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

[1] The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 39; pp. 39:1–39:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\|p_i - p_j\|_\infty = \max_k \left|p_i[k] - p_j[k]\right|$, for $\tau = \infty$. The Closest Pair problem and its corresponding *decision* variant, under the L_τ-metric, are defined as follows.

Closest Pair: Given a set S of n points in $\mathbb{R}^d$, find a pair of distinct points $p_i, p_j \in S$ such that $\mathrm{dist}_\tau(p_i, p_j) = \min_{\ell \neq m}\{\mathrm{dist}_\tau(p_\ell, p_m) \mid p_\ell, p_m \in S\}$.

Closest Pair Decision: Given a set S of n points in $\mathbb{R}^d$, and a parameter $\delta > 0$, determine whether there is a pair of distinct points $p_i, p_j \in S$ such that $\mathrm{dist}_\tau(p_i, p_j) \leq \delta$.

Throughout the paper, the notation L_τ Closest Pair refers to the Closest Pair problem under some *specific* metric L_τ, for $1 \leq \tau \leq \infty$ (and we will mostly consider the case $\tau = \infty$).

In the *algebraic computation tree model* (see [3]), the Closest Pair problem has a complexity lower bound of $\Omega(n \log n)$ (for any L_τ metric), even for the one-dimensional case $d = 1$, as implied from a lower bound for the Element-Uniqueness problem [3].

As for upper bounds, Bentley and Shamos [4, 5] were the first who gave a deterministic algorithm for finding the closest pair under the L_2 metric that runs in $O(n \log n)$ time for any *constant* dimension $d \geq 1$, which is optimal in the algebraic computation tree model, for any fixed d. Their algorithm uses the *divide-and-conquer* paradigm, and became since, a classical textbook example for this technique. In 1976 Rabin presented, in a seminal paper [20], a *randomized* algorithm that finds the closest pair in $O(n)$ expected time, using the *floor* function (which is not included in the algebraic computation tree model). His algorithm uses random sampling to decompose the problem into smaller subproblems, and uses the floor function in solving them, for a total cost of $O(n)$ expected time. Later, in 1979, Fortune and Hopcroft [9] gave a deterministic algorithm that uses the floor function, and runs in $O(n \log \log n)$ time.

The bounds above hold as long as the dimension d is constant, as they involve factors that are exponential in d. Thus, when d is large (e.g., $d = n$), the problem seems to be much less understood. Shamos and Bentley [5] conjectured in 1976 that, for $d = n$, and under the L_2 metric, the problem can be solved in $O(n^2 \log n)$ time; so far, their conjectured bound is considerably far from the $O(n^\omega)$ state-of-the-art time bound for this case [14], where $\omega < 2.373$ denotes the exponent for matrix multiplication (see below). If one settles on approximate solutions, many efficient algorithms were developed in the last two decades, mostly based on LSH (locality sensitive hashing) schemes, and dimensionality reduction via the Johnson-Lindenstrauss transform; see [1, 2] for examples of such algorithms. These algorithms are often used for finding *approximate nearest neighbors*, which itself is of major importance and in massive use in many practical fields of computer science. Nevertheless, finding an *exact* solution seems to be a much harder task.

We consider the case where d depends on n, i.e., $d = n^r$ for some $r > 0$. Note that a naive brute-force algorithm runs in $O(n^2 d)$ time and works for any metric L_τ. For some L_τ metrics, a much faster solution is known; see [14]. Specifically, the L_2 Closest Pair problem can be solved by one algebraic matrix multiplication, so for example when $d = n$, it can be solved in $O(n^\omega)$ time (as already mentioned above). If $\tau \geq 2$ is an *even* integer, then L_τ Closest Pair can be solved in $O(\tau n^\omega)$ time. However, for other L_τ metrics, such as when τ is *odd* (or fractional), or the L_∞ metric, the known solutions are significantly inferior.

For the L_1 and L_∞ metrics, Indyk *et al.* [14] obtained the first (and best known until now) non-naive algorithms for the case $d = n$. For L_1, they gave an algorithm that runs in $O\left(n^{\frac{\omega+3}{2}}\right) = O(n^{2.687})$ time, and for L_∞, one that runs in $O\left(n^{\frac{\omega+3}{2}} \log \mathcal{D}\right) = O(n^{2.687} \log \mathcal{D})$ time, where $\mathcal{D}$ is the diameter of the given point-set. The bound for L_∞ is *weakly polynomial*, due to the dependence on $\mathcal{D}$, and, for real data, only yields an approximation. Their paper is perhaps the most related to our work.

Our new approach is based on two main observations. The first is showing a reduction from L_∞ Closest Pair Decision to another well-studied problem, *dominance product*. The second is by showing we can solve the optimization problem deterministically by executing the decision procedure only $O(\log n)$ times.

We also give improved runtime analysis for the dominance product problem, defined as follows.

Dominance Product: given a set S of n points $p_1, \dots, p_n$ in $\mathbb{R}^d$, compute a matrix D such that for each $i, j \in [n]$, $D[i,j] = \Big|\{k \mid p_i[k] \le p_j[k]\}\Big|$.

This matrix is called the *dominance product* or *dominance matrix* for S. For $d = n$, there is a non-trivial strongly subcubic algorithm by Matoušek [18] (see Section 4), and a slightly improved one by Yuster [24]. For $d \le n$, there are extensions of Matoušek's algorithm by Vassilevska-Williams, Williams, and Yuster [22]. All of them use fast matrix multiplications.

Dominance product computations were liberally used to improve some fundamental algorithmic problems. For example, Vassilevska-Williams, Williams, and Yuster [22], give the first strongly subcubic algorithm for the *all pairs bottleneck paths* (APBP) problem, using dominance product computations. Duan and Pettie [8] later improved their algorithm, also by using dominance product computations, in fact, their time bound for (max, min)-product match the current time bound of computing the dominance product of n points in $\mathbb{R}^n$. Yuster [24] showed that APSP can be solved in strongly subcubic time if the number of distinct weights of edges emanating from any fixed vertex is $O(n^{0.338})$. In his algorithm, he uses dominance product computation as a black box.

1.1 Preliminaries

We review some notations that we will use throughout the paper. We denote by $[N] = \{1, \dots, \lceil N \rceil\}$ the set of the first $\lceil N \rceil$ natural numbers succeeding zero, for any $N \in \mathbb{R}^+$. For a point $p \in \mathbb{R}^d$, we denote by $p[k]$ the k-th coordinate of p, for $k \in [d]$. For a matrix A, we denote the *transpose* of A by A^T. The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

Most of the algorithms discussed in this paper heavily rely on fast matrix multiplication algorithms. Throughout the paper, $\omega < 2.373$ denotes the exponent of multiplying two $n \times n$ matrices [15, 23], and $\omega(1, r, 1)$ refers to the exponent of multiplying an $n \times n^r$ matrix by an $n^r \times n$ matrix, for any $r > 0$; see [12, 13, 17]. For more details on rectangular matrix multiplication exponents, we refer the reader to the seminal work of Huang and Pan [13], and to a more recent work of Le Gall [12, 16, 17].

1.2 Our Results

Let $DP(n,d)$ denote the runtime order for computing the dominance product (defined above) of n points in $\mathbb{R}^d$. We obtain the following results for the L_∞ Closest Pair problem in $\mathbb{R}^d$, where $d = n^r$, for some $r > 0$.

► **Theorem 1.** L_∞ Closest Pair *can be solved by a deterministic algorithm that runs in* $O(DP(n,d)\log n)$ *time.*

Theorem 1 improves the $O(n^{2.687} \log \mathcal{D})$ time bound of Indyk *et al.* [14] (see above) in two aspects. First, the polynomial factor $n^{2.687}$ goes slightly down to $DP(n,n) = n^{2.684}$, which we then improve further to $n^{2.6598}$ in Theorem 4; this holds also for Theorem 2, stated below. The second aspect is that the $\log \mathcal{D}$ factor is replaced by a $\log n$ factor, which makes our algorithm strongly-polynomial, independent of the diameter of the given point-set.

For the proof of Theorem 1, we first show a reduction from L_∞ Closest Pair Decision to dominance product computation, then we show that the optimization problem can be solved deterministically by executing the decision procedure only $O(\log n)$ times.

▶ **Theorem 2.** L_∞ Closest Pair *can be solved by a randomized algorithm that runs in* $O(DP(n,d))$ *expected time.*

▶ **Theorem 3.** *For points with integer coordinates from* $[-M,M]$, L_∞ Closest Pair *can be solved by a deterministic algorithm that runs in* $\tilde{O}\left(\min\{Mn^{\omega(1,r,1)}, DP(n,d)\}\right)$ *time.*

From Theorem 3 we obtain that for n points in $\mathbb{R}^n$ with small integer coordinates we can solve the *optimization* problem in $O(n^\omega)$ time, which is a significant improvement compared to the general case from Theorems 1 and 2.

Additionally, we give a coherent spelled-out runtime analysis for obtaining the best bounds for $DP(n,d)$, for the entire range $d=n^r$, using rectangular matrix multiplications. We demonstrate the use of our analysis by plugging into it the improved bounds for rectangular matrix multiplication by Le Gall [12], resulting in the bounds given below. Right before closing this version of the paper, Le Gall and Urrutia [17] reported further improvements on the bounds given in [12]. Their new bounds can be plugged into our analysis to give approximately 0.01 improvements on the exponents given below.

▶ **Theorem 4.** *given a set S of n points $p_1,\dots,p_n$ in $\mathbb{R}^d$, the dominance product of S can be computed in $O(DP(n,d))$ time, where*

$$DP(n,d) \le \begin{cases} d^{0.697}n^{1.896} + n^{2+o(1)} & \text{if } d \le n^{\frac{\omega-1}{2}} \le n^{0.687} \\ d^{0.909}n^{1.75} & \text{if } n^{0.687} \le d \le n^{0.87} \\ d^{0.921}n^{1.739} & \text{if } n^{0.87} \le d \le n^{0.963} \\ d^{0.931}n^{1.73} & \text{if } n^{0.963} \le d \le n^{1.056} \end{cases}$$

In particular, we obtain that $DP(n,n)=n^{2.6598}$ (using a more precise calculation), which improves Yuster's $O(n^{2.684})$ time bound. As mentioned above, this bound can be slightly improved, using the new rectangular matrix multiplication bounds of Le Gall and Urrutia [17].

2 L_∞ Closest Pair

Recall that, given a set S of n points $p_1,\dots,p_n$ in $\mathbb{R}^d$, the L_∞ Closest Pair problem is to find a pair of points (p_i,p_j), such that $i\neq j$ and $\|p_i-p_j\|_\infty = \min_{\ell\neq m\in[n]}\|p_\ell-p_m\|_\infty$. The corresponding decision version of this problem is to determine whether there is a pair of distinct points (p_i,p_j) such that $\|p_i-p_j\|_\infty\le\delta$, for a given $\delta>0$.

Naively, we can compute all the distances between every pair of points in $O(n^2d)$ time, and choose the smallest one. However, as we see next, a significant improvement can be achieved, for any $d=n^r$, for any $r>0$.

Specifically, we first obtain the following theorem.

▶ **Theorem 5.** *Given a parameter $\delta>0$, and a set S of n points $p_1,\dots,p_n$ in $\mathbb{R}^d$, the set of all pairs (p_i,p_j) with $\|p_i-p_j\|_\infty\le\delta$, can be computed in $O(DP(n,d))$ time.*

Proof. First, we note the following trivial but useful observation.

▶ **Observation 6.** *For a pair of points $p_i,p_j\in\mathbb{R}^d$, $\|p_i-p_j\|_\infty\le\delta \iff p_i[k]\le p_j[k]+\delta$ and $p_j[k]\le p_i[k]+\delta$, for every coordinate $k\in[d]$.*

Indeed, a pair of points (p_i, p_j) satisfies $\|p_i - p_j\|_\infty = \max_{k\in[d]} |p_i[k] - p_j[k]| \le \delta \iff$ for every coordinate $k \in [d]$, $|p_i[k] - p_j[k]| \le \delta$. The last inequalities hold iff $p_i[k] - p_j[k] \le \delta$ and $p_j[k] - p_i[k] \le \delta$, or, equivalently, iff $p_i[k] \le p_j[k] + \delta$ and $p_j[k] \le p_i[k] + \delta$, for each $k \in [d]$. Although the rephrasing in the observation is trivial, it is crucial for our next step. It can be regarded as a (simple) variant of what is usually referred to as "Fredman's trick" (see [11]).

For every $i \in [n]$ we create a new point $p_{n+i} = p_i + (\delta, \delta, \dots, \delta)$. Thus in total, we now have $2n$ points. Concretely, for every $i \in [n]$, we have the points

$$\begin{array}{llcccl} p_i & = (& p_i[1], & p_i[2], & \dots, & p_i[d] \quad), \\ p_{n+i} & = (& p_i[1] + \delta, & p_i[2] + \delta, & \dots, & p_i[d] + \delta \quad). \end{array}$$

We compute the dominance matrix D_δ for these $2n$ points, using the algorithm from Section 4.1. By Observation 6, a pair of points (p_i, p_j) satisfies

$$\|p_i - p_j\|_\infty \le \delta \iff (D_\delta[i, n+j] = d) \ \wedge \ (D_\delta[j, n+i] = d)\,,$$

so we can find all these pairs in $O(n^2)$ additional time. Clearly, the runtime is determined by the time bound of computing the dominance matrix D_δ, that is, $O(DP(n, d))$. ◀

The proof of Theorem 5 shows that solving the L_∞ Closest Pair Decision is not harder than computing the dominance matrix for n points in $\mathbb{R}^d$.

2.1 Solving the Optimization Problem

The algorithm from Theorem 5 solves the L_∞ Closest Pair Decision problem. It actually gives a stronger result, as it finds *all* pairs of points (p_i, p_j) such that $\|p_i - p_j\|_\infty \le \delta$. We use this algorithm in order to solve the optimization problem L_∞ Closest Pair.

As a "quick and dirty" solution, one can solve the optimization problem by using the algorithm from Theorem 5 to guide a binary search over the diameter $\mathcal{D}$ of the input point set, which is at most twice the largest absolute value of the coordinates of the input points. If the coordinates are integers then we need to invoke the algorithm from Theorem 5 $O(\log \mathcal{D})$ times. If the coordinates are reals, we invoke it $O(B)$ times for B bits of precision. However, the dependence on $\mathcal{D}$ makes this method weakly polynomial, and, for real data, only yields an approximation. As we show next, this naive approach can be replaced by strongly-polynomial algorithms, A deterministic one that runs in $O(DP(n, d) \log n)$ time, and a randomized one that runs in $O(DP(n, d))$ expected time.

Deterministic strongly-polynomial algorithm.

▶ **Theorem 7.** *Given a set S of n points $p_1, \dots, p_n$ in $\mathbb{R}^d$, the L_∞* Closest Pair *problem can be solved for S in $O(DP(n, d) \log n)$ time.*

Proof. Since the distance between the closest pair of points, say p_i, p_j, is

$$\delta_0 = \|p_i - p_j\|_\infty = \max_{k\in[d]} \left|p_i[k] - p_j[k]\right|,$$

it is one of the $O(n^2 d)$ values $p_\ell[k] - p_m[k]$, $\ell, m \in [n]$, $k \in [d]$. Our goal is to somehow search through these values, using the decision procedure (i.e., the algorithm from Theorem 5). However, enumerating all these values takes $\Omega(n^2 d)$ time, which is too expensive, and pointless anyway, since by having them, the closest pair can be found immediately. Instead, we proceed in the following more efficient manner.

For each $k \in [d]$, we sort the points of S in increasing order of their k-th coordinate. This takes $O(nd \log n)$ time in total. Let $\left(p_1^{(k)}, \ldots, p_n^{(k)}\right)$ denote the sequence of the points of S sorted in increasing order of their k-th coordinate. For each k, let $M^{(k)}$ be an $n \times n$ matrix, so that for $i, j \in [n]$, we have

$$M^{(k)}[i,j] = p_i^{(k)}[k] - p_j^{(k)}[k].$$

We are in fact interested only in the upper triangular portion of $M^{(k)}$, where its elements are positive, but for simplicity of presentation, we ignore this issue. (We view the row indices from bottom to top, i.e., the first row is the bottommost one, and the column indices from left to right.)

Observe that each row of $M^{(k)}$ is sorted in decreasing order and each column is sorted in increasing order. Under these conditions, the selection algorithm of Frederickson and Johnson [10] can find the t-largest element of $M^{(k)}$, for any $1 \le t \le n^2$, in $O(n)$ time.[2] Note that we do not need to explicitly construct the matrices $M^{(k)}$, this will be too expensive. The bound of Frederickson-Johnson's algorithm holds as long as each entry of $M^{(k)}$ is accessible in $O(1)$ time, like in our case.

We use this method to conduct a simultaneous binary search over all d matrices $M^{(k)}$ to find δ_0. At each step of the search we maintain two counters $L_k \le H_k$, for each k. Initially $L_k = 1$ and $H_k = n^2$. The invariant that we maintain is that, at each step, δ_0 lies in between the L_k-th and the H_k-th largest elements of $M^{(k)}$, for each k.

Each binary search step is performed as follows. We compute $r_k = \lfloor (L_k + H_k)/2 \rfloor$, for each k, and apply the Frederickson-Johnson algorithm to retrieve the r_k-th largest element of $M^{(k)}$, which we denote as δ_k, in total time $O(nd)$. We give δ_k the weight $H_k - L_k + 1$, and compute the weighted median δ_{med} of $\{\delta_1, \ldots, \delta_d\}$. We run the L_∞ Closest Pair Decision procedure of Theorem 5 on δ_{med}. Suppose that it determines that $\delta_0 \le \delta_{\text{med}}$. Then for each k for which $\delta_k \ge \delta_{\text{med}}$ we know that $\delta_0 \le \delta_k$, so we set $H_k := r_k$ and leave L_k unchanged. Symmetric actions are taken if $\delta_0 > \delta_{\text{med}}$. In either case, we remove roughly one quarter of the candidate differences; that is, the sum $\sum_{k \in [d]} (H_k - L_k + 1)$ decreases by roughly a factor of 3/4. Hence, after $O(\log n)$ steps, the sum becomes $O(d)$, and a straightforward binary search through the remaining values finds δ_0. The overall running time is

$$O(nd \log n + DP(n,d)(\log n + \log d)).$$

Since in our setting d is polynomial in n, and $nd \ll DP(n,d)$, we obtain that the overall runtime is $O(DP(n,d) \log n)$. This completes the proof of Theorem 1. ◀

Randomized algorithm. Using randomization, we can improve the time bound of the preceding deterministic algorithm to equal the time bound of computing the dominance product $O(DP(n,d))$ in expectation. This can be done by using a randomized optimization technique by Chan [6]. Among the problems for which this technique can be applied, Chan specifically addresses the Closest Pair problem.

▶ **Theorem 8** (Chan [6]). *Let U be a collection of objects. If the* Closest Pair Decision *problem can be solved in $O(T(n))$ time, for an arbitrary distance function $d : U \times U \to \mathbb{R}$, then the* Closest Pair *problem can be solved in $O(T(n))$ expected time, assuming that $T(n)/n$ is monotone increasing.*

[2] Simpler algorithms can select the t-largest element in such cases in $O(n \log n)$ time, which is also sufficient for our approach.

We refer the reader to [6], for the proof of Theorem 8. By Theorem 5, L_∞ Closest Pair Decision can be solved in $O(DP(n,d))$ time. Clearly, $DP(n,d)/n$ is monotone increasing in n. Hence, by Theorem 8, we obtain a randomized algorithm for L_∞ Closest Pair that runs in $O(DP(n,d))$ expected time, as stated in Theorem 2.

3 L_∞ Closest Pair with Integer Coordinates

A considerable part of the algorithm from the previous section is the reduction to computing a suitable dominance matrix. The algorithms for computing dominance matrices given in Section 4 do not make any assumptions on the coordinates of the points, and support real numbers. When the coordinates are bounded integers, we can improve the algorithms. In particular, for n points in $\mathbb{R}^n$ with small integer coordinates we can solve the *optimization* problem in $O(n^\omega)$ time, which is a significant improvement compared to the $O(n^{2.6598})$ time bound of our previous algorithm for this case[3]. Our improvement is based on techniques for computing $(\min,+)$-matrix multiplication over integer-valued matrices.

▶ **Theorem 9.** *Let S be a set of n points $p_1,\ldots,p_n$ in $\mathbb{R}^d$ such that $d=n^r$ for some $r>0$, and for all $i\in[n]$, $k\in[d]$, $p_i[k]$ is an integer in $[-M,M]$. Then the L_∞ closest pair can be computed in*

$$\tilde{O}\left(\min\left\{Mn^{\omega(1,r,1)},\, DP(n,d)\right\}\right) \text{ time.}$$

We first define $(\max,+)$-product and $(\min,+)$-product over matrices.

▶ **Definition 10** (Distance products of matrices)**.** Let A be an $n\times m$ matrix and B be an $m\times n$ matrix. The $(\max,+)$-product of A and B, denoted by $A\star B$, is the $n\times n$ matrix C whose elements are given by

$$c_{ij}=\max_{1\le k\le m}\{a_{ik}+b_{kj}\},\quad \text{for } i,j\in[n].$$

Similarly, the $(\min,+)$-product of A and B denoted by $A*B$ is the $n\times n$ matrix C' whose elements are given by

$$c'_{ij}=\min_{1\le k\le m}\{a_{ik}+b_{kj}\},\quad \text{for } i,j\in[n].$$

We refer to either of the $(\min,+)$-product or the $(\max,+)$-product as a *distance product.*

The distance product of an $n\times m$ matrix by an $m\times n$ matrix can be computed naively in $O(n^2m)$ time. When $m=n$, the problem is equivalent to APSP (all pairs shortest paths) problem in a directed graph with real edge weights, and the fastest algorithm known is a recent one by Chan and Williams [7] that runs in $O\left(n^3/2^{\sqrt{\Omega(\log n)}}\right)$ time. It is a prominent long-standing open problem whether a truly subcubic algorithm for this problem exists. However, when the entries of the matrices are integers, we can convert distance products of matrices into standard algebraic products. We use a technique by Zwick [25].

▶ **Lemma 11** (Zwick [25])**.** *Given an $n\times m$ matrix $A=\{a_{ij}\}$ and an $m\times n$ matrix $B=\{b_{ij}\}$ such that $m=n^r$ for some $r>0$, and all the elements of both matrices are integers from $[-M,M]$, their $(\min,+)$-product $C=A*B$ can be computed in $\tilde{O}(Mn^{\omega(1,r,1)})$ time.*

[3] For integer coordinates that are bounded by a constant, the L_∞-diameter of the points is also a constant (bounded by twice the largest coordinate), hence, one can use the decision procedure to (naively) guide a binary search over the diameter in constant time.

With minor appropriate modifications, the $(\max,+)$-product of matrices A and B can be computed within the same time as in Lemma 11.

We now give an algorithm for computing all-pairs L_∞ distances, by using the fast algorithm for computing $(\max,+)$-product over bounded integers.

▶ **Lemma 12.** *Let S be a set of n points $p_1,\dots,p_n$ in $\mathbb{R}^d$ such that $d = n^r$ for some $r > 0$, and for all $i \in [n]$, $p_i[k]$ is an integer from the interval $[-M, M]$, for all $k \in [d]$. Then the L_∞ distances between all pairs of points (p_i, p_j) from S can be computed in $\tilde{O}(Mn^{\omega(1,r,1)})$ time.*

Proof. We create the $n \times d$ matrix $A = \{a_{ik}\}$ and the $d \times n$ matrix $B = (-A)^T = \{b_{ki}\}$, where

$$\begin{aligned} a_{ik} &= p_i[k], \quad \text{for } i \in [n],\ k \in [d], \\ b_{ki} &= -p_i[k], \quad \text{for } i \in [n],\ k \in [d]. \end{aligned}$$

Now we compute the $(\max,+)$-product $C = A{\star}B$. The matrix L of all-pairs L_∞-distances is then easily seen to be

$$L[i,j] = \max\bigl\{C[i,j], C[j,i]\bigr\} = \|p_i - p_j\|_\infty ,$$

for every pair $i, j \in [n]$.

Clearly, the runtime is determined by computing the $(\max,+)$-product $C = A \star B$. This is done as explained earlier, and achieves the required running time. ◀

Consequently, by taking the minimum from the algorithm above, and the (say, deterministic) algorithm from Section 2, we obtain that for points in $\mathbb{R}^d$ with integer coordinates from $[-M, M]$, where $d = n^r$ for some $r > 0$, we can find the L_∞ closest pair in

$$\tilde{O}\left(\min\left\{Mn^{\omega(1,r,1)},\ DP(n,d)\right\}\right) \text{ time,}$$

as stated in Theorem 3.

4 Dominance Products

We recall the dominance product problem: given n points $p_1,\dots,p_n$ in $\mathbb{R}^d$, we want to compute a matrix D such that for each $i, j \in [n]$,

$$D[i,j] = \Bigl|\{k \mid p_i[k] \le p_j[k]\}\Bigr|.$$

It is easy to see that the matrix D can be computed naively in $O(dn^2)$ time. Note that, in terms of decision tree complexity, it is straightforward to show that $O(dn \log n)$ pairwise comparisons suffice for computing the dominance product of n points in $\mathbb{R}^d$. However, the actual best known time bound to solve this problem is significantly larger than its decision tree complexity bound.

The first who gave a truly subcubic algorithm to compute the dominance product of n points in $\mathbb{R}^n$ is Matoušek [18]. We first outline his algorithm, and then present our extension and improved runtime analysis.

▶ **Theorem 13** (Matoušek [18])**.** *Given a set S of n points in $\mathbb{R}^n$, the dominance matrix for S can be computed in $O(n^{\frac{3+\omega}{2}}) = O(n^{2.687})$ time.*

Proof. For each $j \in [n]$, sort the n points by their j-th coordinate. This takes a total of $O(n^2 \log n)$ time. Define the *j-th rank of point* p_i, denoted as $r_j(p_i)$, to be the position of p_i in the sorted list for coordinate j. Let $s \in [\log n, n]$ be a parameter to be determined later. Define n/s pairs (assuming for simplicity that n/s is an integer) of $n \times n$ Boolean matrices $(A_1, B_1), \dots, (A_{n/s}, B_{n/s})$ as follows:

$$A_k[i,j] = \begin{cases} 1 & \text{if } r_j(p_i) \in [ks, ks+s) \\ 0 & \text{otherwise,} \end{cases} \qquad B_k[i,j] = \begin{cases} 1 & \text{if } r_j(p_i) \geq ks+s \\ 0 & \text{otherwise,} \end{cases}$$

for $i, j \in [n]$. Put $C_k = A_k \cdot B_k^T$. Then $C_k[i,j]$ equals the number of coordinates t such that $r_t(p_i) \in [ks, ks+s)$, and $r_t(p_j) \geq ks + s$.

Thus, by letting $C = \sum_{k=1}^{n/s} C_k$, we have that $C[i,j]$ is the number of coordinates t such that $p_i[t] \leq p_j[t]$ and $\lfloor r_t(p_i)/s \rfloor < \lfloor r_t(p_j)/s \rfloor$.

Next, we compute a matrix E such that $E[i,j]$ is the number of coordinates t such that $p_i[t] \leq\ p_j[t]$ and $\lfloor r_t(p_i)/s \rfloor = \lfloor r_t(p_j)/s \rfloor$. Then $D := C + E$ is the desired dominance matrix.

To compute E, we use the n sorted lists we computed earlier. For each pair $(i,j) \in [n] \times [n]$, we retrieve $q := r_j(p_i)$. By reading off the adjacent points that precede p_i in the j-th sorted list in reverse order (i.e., the points at positions $q-1$, $q-2$, etc.), and stopping as soon as we reach a point p_k such that $\lfloor r_j(p_k)/s \rfloor < \lfloor r_j(p_i)/s \rfloor$, we obtain the list $p_{i_1}, \dots, p_{i_l}$ of $l \leq s$ points such that $p_{i_x}[j] \leq p_i[j]$ and $\lfloor r_j(p_i)/s \rfloor = \lfloor r_j(p_{i_x})/s \rfloor$. For each $x = 1, \dots, l$, we add a 1 to $E[i_x, i]$. Assuming constant time lookups and constant time probes into a matrix (as is standard in the real RAM model), this entire process takes only $O(n^2 s)$ time. The runtime of the above procedure is therefore $O(n^2 s + \frac{n}{s} \cdot n^\omega)$. Choosing $s = n^{\frac{\omega-1}{2}}$, the time bound becomes $O(n^{\frac{3+\omega}{2}})$. ◀

Yuster [24] has slightly improved this algorithm to run in $O(n^{2.684})$ time, by using rectangular matrix multiplication.

4.1 Generalized and Improved Bounds

We extend Yuster's idea to obtain bounds for dimension $d = n^r$, for the entire range $r > 0$, and, at the same time, give an improved time analysis, using the recent bounds for rectangular matrix multiplications of Le Gall [12, 16] coupled with an interpolation technique. This analysis is not trivial, as Le Gall's bounds for $\omega(1, r, 1)$ are obtained by solving a nonlinear optimization problem, and are only provided for a few selected values of r (see Table 1 in [16] and [12]). Combining Le Gall's exponents with an interpolation technique, similar to the one used by Huang and Pan [13], we obtain improved bounds for all values $d = n^r$, for any $r > 0$.

Note that the matrices A_k and B_k, defined above, are now $n \times d$ matrices. Thus, the sum C defined earlier, can be viewed as a product of block matrices

$$C = \begin{bmatrix} A_1 & A_2 & \cdots & A_{n/s} \end{bmatrix} \cdot \begin{bmatrix} B_1^T \\ B_2^T \\ \vdots \\ B_{n/s}^T \end{bmatrix}.$$

Thus, to compute C we need to multiply an $n \times (dn/s)$ matrix by a $(dn/s) \times n$ matrix. Computing E in this case can be done exactly as in Matoušek's algorithm, in $O(nds)$ time.

Consider first the case where d is small; concretely, $d \leq n^{\frac{\omega-1}{2}}$. In this case we compute C using the following result by Huang and Pan.

Table 1 The relevant entries from Le Gall's table (Table 1 in [16]), the value for ω_0 is taken from [15]. The dominance product can be computed in $O(n^{\omega_i})$ time, for dimension $d_i = n^{\zeta_i}$.

r	ω	ζ
$r_0 = 1.0$	$\omega_0 = 2.372864$	$\zeta_0 = 0.6865$
$r_1 = 1.1$	$\omega_1 = 2.456151$	$\zeta_1 = 0.7781$
$r_2 = 1.2$	$\omega_2 = 2.539392$	$\zeta_2 = 0.8697$
$r_3 = 1.3$	$\omega_3 = 2.624703$	$\zeta_3 = 0.9624$
$r_4 = 1.4$	$\omega_4 = 2.711707$	$\zeta_4 = 1.0559$

▶ **Lemma 14** (Huang and Pan [13]). *Let* $\alpha = \sup\{0 \le r \le 1 \mid w(1, r, 1) = 2 + o(1)\}$. *Then for all* $n^\alpha \le m \le n$, *one can multiply an* $n \times m$ *matrix with an* $m \times n$ *matrix in time* $O\left(m^{\frac{\omega-2}{1-\alpha}} n^{\frac{2-\omega\alpha}{1-\alpha}}\right)$.

Huang and Pan [13] showed that $\alpha > 0.294$. Recently, Le Gall [12, 16] improved the bound on α to $\alpha > 0.302$. By plugging this into Lemma 14, we obtain that multiplying an $n \times m$ matrix with an $m \times n$ matrix, where $n^\alpha \le m \le n$, can be done in time $O(m^{0.535} n^{1.839})$.

From the above, computing C and E can be done in $O\left((dn/s)^{0.535} n^{1.839} + dns\right)$ time. By choosing $s = n^{0.896}/d^{0.303}$, the runtime is asymptotically minimized, and we obtain the time bound $O(d^{0.697} n^{1.896})$. This time bound holds only when $n^\alpha < n^{0.302} \le dn/s \le n$, which yields the time bound

$$O(d^{0.697} n^{1.896} + n^{2+o(1)}), \text{ for } d \le n^{(\omega-1)/2} \le n^{0.687}.$$

We now handle the case $d > n^{(\omega-1)/2}$. Note that in this case, $dn/s > n$ (for s as above), thus, we cannot use the bound from Lemma 14. Le Gall [12, 16] gives a table (Table 1 in [16] and [12]) of values r (he refers to them as k), including values of $r > 1$ (which are those we need), with various respective exponents $\omega(1, r, 1)$. We will confine ourselves to the given bounds for the values $r_1 = 1.1$, $r_2 = 1.2$, $r_3 = 1.3$, and $r_4 = 1.4$. We denote their corresponding exponents $\omega(1, r_i, 1)$ by $\omega_1 \le 2.456151, \omega_2 \le 2.539392$, $\omega_3 \le 2.624703$, and $\omega_4 \le 2.711707$ respectively. The exponent for $r_0 = 1$ is $\omega_0 = \omega \le 2.372864$ (see [15, 23]).

The algorithm consists of two parts. For a parameter s, that we will fix shortly, the cost of computing $C = A \cdot B^T$ is $O\left(n^{\omega_r}\right)$, where ω_r is a shorthand notation for $\omega(1, r, 1)$, and where $n^r = dn/s$, and the cost of computing E is $O(nds) = O\left(s^2 n^r\right)$. Dropping the constants of proportionality, and equating the two expressions, we choose

$$s = n^{(\omega_r - r)/2}, \qquad \text{that is,} \qquad d = sn^{r-1} = n^{(\omega_r + r)/2 - 1} = n^{\zeta_r},$$

for $\zeta_r = (\omega_r + r)/2 - 1$. Put $\zeta_i = \zeta_{r_i}$, for the values $r_0, \ldots, r_4$ mentioned earlier; see Table 1.

Now if we are lucky and $d = n^{\zeta_i}$, for $i = 0, 1, 2, 3, 4$, then the overall cost of the algorithm is $O(n^{\omega_i})$. For in-between values of d, we need to interpolate, using the following bound, which is derived in the earlier studies (see, e.g., Huang and Pan [13]), and which asserts that, for $a \le r \le b$, we have

$$\omega_r \le \frac{(b - r)\omega_a + (r - a)\omega_b}{b - a}. \tag{1}$$

That is, given $d = n^\zeta$, where $\zeta_i \le \zeta \le \zeta_{i+1}$, for some $i \in \{0, 1, 2, 3\}$, the cost of the algorithm will be $O\left(n^{\omega_r}\right)$, where r satisfies

$$\zeta = \zeta_r = \frac{\omega_r + r}{2} - 1.$$

Table 2 The time bound for computing dominance product for n points in dimension $n^{\zeta_{\min}} \leq d \leq n^{\zeta_{\max}}$ is $O\left(d^u n^v\right)$.

$\zeta_{\min}$	$\zeta_{\max}$	u	v
0.687	0.87	0.909	1.75
0.87	0.963	0.921	1.739
0.963	1.056	0.931	1.73

Substituting the bound for ω_r from (1), with $a = r_i$ and $b = r_{i+1}$, we have

$$\frac{(r_{i+1} - r)\omega_i + (r - r_i)\omega_{i+1}}{r_{i+1} - r_i} + r = 2(\zeta + 1).$$

Eliminating r, we get

$$r = \frac{2(\zeta + 1)(r_{i+1} - r_i) - r_{i+1}w_i + r_i w_{i+1}}{w_{i+1} + r_{i+1} - w_i - r_i}, \tag{2}$$

and the cost of the algorithm will be $O\left(n^{\omega_r}\right)$, where

$$\omega_r \leq \frac{(r_{i+1} - r)\omega_i + (r - r_i)\omega_{i+1}}{r_{i+1} - r_i}. \tag{3}$$

Note that r is a linear function of ζ, and so is ω_r. Writing $\omega_r = u\zeta + v$, the cost is

$$O\left(n^{\omega_r}\right) = O\left(n^{u\zeta+v}\right) = O\left(d^u n^v\right).$$

The values of u and v for each of our intervals are given in Table 2. (The first row covers the two intervals $1.0 \leq r \leq 1.1$ and $1.1 \leq r \leq 1.2$, as the bounds happen to coincide there.) See also Theorem 4 in Section 1.2. We have provided explicit expressions for $DP(n, d)$ only for $d \leq n^{\zeta_4} = n^{1.056}$, which includes the range $d \leq n$, which is the range one expects in practice. Nevertheless, the recipe that we provide can also be applied to larger values of d, using larger entries from Le Gall's table [12, 16]. As mentioned earlier, the exponents we obtained for $DP(n, d)$ can be even slightly further improved by approximately 0.01, by plugging into our analysis the very recent new bounds for rectangular matrix multiplication of Le Gall and Urrutia [17] (see Table 3 in [17]).

References

1 Nir Ailon and Bernard Chazelle. The fast Johnson-Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.

2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

3 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proc. of the 15th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 80–86, 1983.

4 Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

5 Jon Louis Bentley and Michael Ian Shamos. Divide-and-conquer in multidimensional space. In *Proc. of the 8th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 220–230, 1976.

6 Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999.

7 Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proc. of the 27th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

8 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. of the 20th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 384–391, 2009.

9 Steve Fortune and John Hopcroft. A note on Rabin's nearest-neighbor algorithm. *Inform. Process. Lett.*, 8(1):20–23, 1979.

10 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in x + y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2):197 – 208, 1982.

11 Michael L. Fredman. How good is the information theory bound in sorting? *Theoret. Comput. Sci*, 1(4):355–361, 1976.

12 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 514–523. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.80.

13 Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

14 Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 782–792. Springer, 2004. doi:10.1007/978-3-540-27836-8_66.

15 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Sympos. on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

16 François Le Gall. Faster algorithms for rectangular matrix multiplication. *CoRR*, abs/1204.1111, 2012.

17 François Le Gall and Florent Urrutia. Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor. *ArXiv e-prints*, August 2017. arXiv:1708.05622.

18 Jiří Matoušek. Computing dominances in E^n. *Inform. Process. Lett.*, 38(5):277–278, 1991.

19 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag New York, NY, 1985.

20 Michael Rabin. Probabilistic algorithms. In *Algorithms and Complexity, Recent Results and New Directions*, Academic Press, pages 21–39, 1976.

21 Michael Ian Shamos. Geometric complexity. In *Proc. of 7th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 224–233, 1975.

22 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

23 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

24 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 950–957. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496873.

25 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

Orthogonal Vectors Indexing*

Isaac Goldstein[†1], **Moshe Lewenstein**[‡2], **and Ely Porat**[§3]

1 **Bar-Ilan University, Ramat Gan, Israel**
goldshi@cs.biu.ac.il

2 **Bar-Ilan University, Ramat Gan, Israel**
moshe@cs.biu.ac.il

3 **Bar-Ilan University, Ramat Gan, Israel**
poratelv@cs.biu.ac.il

Abstract

In the recent years, intensive research work has been dedicated to prove conditional lower bounds in order to reveal the inner structure of the class P. These conditional lower bounds are based on many popular conjectures on well-studied problems. One of the most heavily used conjectures is the celebrated Strong Exponential Time Hypothesis (SETH). It turns out that conditional hardness proved based on SETH goes, in many cases, through an intermediate problem - the Orthogonal Vectors (OV) problem.

Almost all research work regarding conditional lower bound was concentrated on time complexity. Very little attention was directed toward space complexity. In a recent work, Goldstein et al. [17] set the stage for proving conditional lower bounds regarding space and its interplay with time. In this spirit, it is tempting to investigate the space complexity of a data structure variant of OV which is called *OV indexing*. In this problem n boolean vectors of size $c \log n$ are given for preprocessing. As a query, a vector v is given and we are required to verify if there is an input vector that is orthogonal to it or not.

This OV indexing problem is interesting in its own, but it also likely to have strong implications on problems known to be conditionally hard, in terms of time complexity, based on OV. Having this in mind, we study OV indexing in this paper from many aspects. We give some space-efficient algorithms for the problem, show a tradeoff between space and query time, describe how to solve its reporting variant, shed light on an interesting connection between this problem and the well-studied SetDisjointness problem and demonstrate how it can be solved more efficiently on random input.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases SETH, orthogonal vectors, space complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.40

1 Introduction

Recently, there is an intensive research work aimed at understanding the complexity within the class P (decision problems that are solved by polynomial time algorithms). Specifically, many conditional lower bounds have been proven on many polynomial algorithmic problems. These lower bounds are based on some conjectures on well-studied problems, especially

[*] A full version of the paper is available at https://arxiv.org/abs/1710.00586.

[†] This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities

[‡] This work was partially supported by an ISF grant #1278/16

[§] This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 40; pp. 40:1–40:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

notable are 3SUM, APSP and SETH. The Strong Exponential Time Hypothesis (SETH) [18, 19] states the following:

▶ **Conjecture 1** (Strong Exponential Time Hypothesis). *There is no $\epsilon > 0$ such that kSAT can be solved in $O(2^{(1-\epsilon)n})$ for all k.*

Many conditional lower bounds for both polynomial and exponential time solvable problems are based on this conjecture. A partial list includes [25, 21, 14, 28, 4, 6, 8, 16, 7, 2, 9, 1, 5, 22]. For polynomial time solvable problems many of the conditional lower bounds are proven through the use of an intermediate problem called *Orthogonal Vectors* (OV) which is defined as follows.

▶ **Definition 1** (Orthogonal Vectors). Given a set S of n input vectors from $\{0,1\}^d$, decide if there are $u, v \in S$ such that u is orthogonal to v.

If SETH is true then there is no $O(n^{2-\epsilon})$ solution for OV for any $\epsilon > 0$ (see [29, 30]). This conditional lower bound on OV was heavily used to obtain conditional lower bounds on the time complexity of a long list of algorithmic problems. This includes graph problems [28, 22], dynamic problems [4], string problems [6, 7, 2, 9] and many other important problems from a variety of research fields.

A recent work by Goldstein et al. [17] set the stage for proving conditional lower bounds on space-time tradeoffs. Specifically, it was suggested that we can achieve space lower bounds by considering a data structure variant of SAT. Given a formula ϕ in a CNF format and a list of variables L from ϕ, we need to preprocess ϕ and L and create a data structure to support the following queries. Given an assignment to all variables not in L we are required to answer if this assignment can be completed to a full assignment that satisfies ϕ. A closely related problem is *Orthogonal Vectors Indexing* (OV Indexing) that is defined as follows.

▶ **Definition 2** (Orthogonal Vectors Indexing). Given a set S that contains n d-length boolean vectors, preprocess S and answer queries of the following form: given d-length boolean vector v, is there a vector in S which is orthogonal to v.

SETH can be reduced to OV indexing (see the details in the full version of this paper). As a consequence of this reduction there is no polynomial time preprocessing algorithm for OV indexing that achieves truly sublinear query time.

The main question that we consider is what the space requirements of OV indexing are. In this paper we examine this question in detail from various aspects for the case that $d = c \log n$ for some constant $c > 1$ (if c is non-constant its seems hard to achieve any improvement due to the connection to SETH). On one hand, solving OV indexing for input vectors of length $c \log n$ can be done easily using a lookup table of size n^c. Using this table, queries can be answered in constant time. On the other hand, without any preprocessing queries can be answered in linear time. It is interesting to figure out what can be done in between these two extremes. Can we achieve truly sublinear query time with less than n^c space? Is there a clear tradeoff between time and space? What can we say about the reporting version of this problem? In this paper we investigate all these questions and more.

Understanding the space requirements of OV indexing is interesting in its own right, but it can have many implications on other problems. Along the lines of Goldstein et al. [17] OV indexing can serve as a basis for proving conditional hardness in terms of space for other algorithmic problems. Specifically, as OV is a standard tool in demonstrating conditional hardness of problems in terms of time it is likely that understanding the space hardness of its data structure variant - OV indexing - can be applied to many problems shown to be hard based on OV. In the work by Goldstein et al. [17] there was an attempt to state a general hardness conjecture for OV indexing. However, as no solution to neither OV indexing nor the data structure variant of SAT was suggested in [17] (other than the trivial ones), a more fine grained conjecture was out of reach. One major motivation for this paper is to state such a conjecture based on improved upper bounds for OV indexing (see more detailed discussion in the last section of this paper).

Related Work. The Partial Match problem and its variants were extensively studied for decades. These problems are related to our OV indexing problem (see, for example, [3]). One of the first works regarding Partial Match is by Rivest [26, 27]. However, his work focused on the average case analysis of several solutions for the problem that in the worst case do not achieve an improvement over the trivial solution, unless the number of "don't cares" symbols (corresponding to the zeroes in the OV indexing problem) in the query is not too large. Many works on the Partial Match problem and its variants focus on improving the time complexity rather than the space complexity which is the main concern of this paper. Other works that do consider space complexity deal with the case of very large dimension d that can be even linear in n [10, 13, 20]. This case admits very different behaviour from the case we handle in this paper in which $d = \Theta(\log n)$.

Our Results. In this paper we present the following results regarding OV indexing. We suggest 3 algorithms that solve OV indexing with truly less that n^c space and truly sublinear query time. We show how to use the second and third algorithms we present to get a tradeoff between space and query time. A variant of the first algorithm is used to prove the connection between OV indexing and SetDisjointness, a problem which was considered by several papers as the basis for showing space conditional hardness. We also solve the reporting variant of OV indexing in which we need to report all input vectors that are orthogonal to our query vector. Finally, we show that, on random input vectors, OV indexing can be solved more efficiently in terms of space.

2 DivideByOnes: First Space-Efficient Solution for OV indexing

Our goal is to achieve an algorithm that has truly sublinear query time and requires $O(n^{c-\epsilon})$ space for some $\epsilon > 0$. This is an improvement over the trivial algorithm that uses n^c space. We note that in this solution and throughout this paper the notations $\tilde{O}$ and $\tilde{\Omega}$ (almost always) suppress not just polylogarithmic factors as usual, but also all factors that are smaller than n^ϵ for any $\epsilon > 0$.

2.1 DivideByOnes Algorithm

Preprocessing. The first step is to save a set S_1 of all vectors from S with at most $c_1 \log n$ ones for some constant $0 < c_1 \leq c/2$. There are at most $\sum_{k=0}^{c_1 \log n} \binom{c \log n}{k} \leq c_1 \log n \binom{c \log n}{c_1 \log n}$ vectors in S_1. We have that $\binom{c \log n}{c_1 \log n} \approx n^{c \log c - c_1 \log c_1 - (c-c_1) \log (c-c_1)}$. We choose the largest c_1 such that the number of vectors in S_1 will be $\tilde{O}(n^{1-\epsilon})$ for some $\epsilon > 0$.

Let S_2 be the set of vectors from S with more than $c_1 \log n$ ones. Assume that c is an integer. We split each vector in S_2 into c parts each of length $\log n$ bits. As all the vectors in S_2 have at least $c_1 \log n$ ones, we are guaranteed that at least one of the c parts of each vector has at least $\frac{c_1}{c} \log n$ ones.

We have n possible vectors of size $\log n$, so we create c arrays $A_1, A_2, ..., A_c$ of length n each, such that the ith entry in each array represents the $\log n$-length boolean vector that its numerical value is i. In the ith entry of an array A_j we create a list that contains each vector $v \in S_2$ such that: (i) The number of ones it has in its jth part is the maximum among all its parts (ties are broken arbitrarily). (ii) The value of its bits in its jth part is orthogonal to the value of the $\log n$-length vector whose numerical value is i (the numerical value of an m-length vector is the value of this boolean vector that is parsed as an m-length boolean number).

To analyse the space consumed by these arrays one should notice that each vector $v \in S_2$ appears only in one array. Moreover, as v appears only in the array that represents the part in which v has the maximum number of ones, the number of lists in this array that contain v is at most $\frac{n}{2^{\frac{c_1}{c}\log n}} = n^{1-\frac{c_1}{c}}$. Therefore, the total size of all arrays is no more than $n \cdot n^{1-\frac{c_1}{c}} = n^{2-\frac{c_1}{c}}$ which is truly subquadratic.

Query. When we get a query vector u we first check in S_1 if there is a vector that is orthogonal to u. Then, we partition u to c equal parts. For each part j if the numerical value of all bits in this part is i we check all the vectors in the ith list of A_j and verify if one of them is indeed orthogonal to u. The problem with this process is that the length of the list we check may be $\tilde{\Omega}(n)$, so our query time will be $O(n)$ which is trivial. To overcome this and obtain a constant query time for long lists, we need to treat lists whose length is $\tilde{\Omega}(n)$ differently in the preprocessing phase.

Additional Preprocessing. For each entry i in some array that the length of the vectors list in it is not truly sublinear, we store a bitmap that tells for all possible values of the other $(c-1)\log n$ bits whether there is a vector in S that is orthogonal to these bits and the $\log n$ bits represented by i. The size of the bitmap is $2^{(c-1)\log n} = n^{c-1}$. As calculated before, the total number of vectors in all lists of the array is $n^{2-\frac{c_1}{c}}$. Consequently, the number of lists that have $\tilde{\Omega}(n)$ vectors in them is no more than $n^{1-\frac{c_1}{c}}$. Therefore, the space needed for all bitmaps is $n^{c-\frac{c_1}{c}}$.

2.1.1 Generalization to klogn

We can generalize the above solution by partitioning the vectors to parts whose size is $k\log n$ for some $k > 0$. First we consider the case that k divides c. In this case, the algorithm continues in same way as for the case that $k = 1$. The number of lists in each array is n^k. Each input vector v has at least $\frac{c_1}{c}k$ ones in the part with the largest number of ones. Consequently, each input vector v occurs in $n^{k-\frac{c_1}{c}k}$ lists in the array corresponding to the part with most ones in v. The total size of all arrays and lists is $O(n^{k+1-\frac{c_1}{c}k})$. The number of long lists is at most $O(n^{k-\frac{c_1}{c}k})$. Each bitmap has size n^{c-k}. Therefore, the space usage for handling long lists is $O(n^{c-\frac{c_1}{c}k})$. The total space of the data structure is $O(n^{k+1-\frac{c_1}{c}k} + n^{c-\frac{c_1}{c}k})$. By setting $k = c-1$ (if possible, otherwise see the next paragraph) we get the lowest space complexity, which is $O(n^{c-c_1(1-\frac{1}{c})})$.

In case k does not divide c, we can partition each vector to $\lfloor\frac{c}{k}\rfloor$ parts of length $k\log n$. However, we are left with one part P whose length is smaller than $k\log n$. It can be the case that for some input vector v the number of ones in each of the parts of length $k\log n$ is smaller than $\frac{c_1}{c}k$, as there can be many ones in P. In order to solve this problem we can do the following. Let $c_1' = c_1 - \epsilon$ for any $\epsilon > 0$. We define $k' = \lfloor\frac{k}{\epsilon}\rfloor\epsilon$ and $c' = \lfloor\frac{c}{\epsilon}\rfloor\epsilon$. It is clear that $k' > k - \epsilon$ and $c' > c - \epsilon$. Each input vector v can be partitioned to $m_1 = \lfloor\frac{c}{\epsilon}\rfloor$ parts

$P_1, P_2, ..., P_{m_1}$ whose length is ϵ and another optional part P whose length is less than ϵ. If we ignore the bits of any vector in P, we are still guaranteed that there are at least c_1' ones in the rest of the vector. We can choose $m_2 = \lfloor \frac{k}{\epsilon} \rfloor$ parts from the m_1 parts $P_1, P_2, ..., P_{m_1}$. This will give us exactly $k' \log n$ bits. There are $m_3 = \binom{m_1}{m_2}$ options of how to choose m_2 parts out of the m_1 parts. The number m_3 is constant as k, c and ϵ are all constants. Therefore, we can create m_3 arrays $A_1, A_2, ..., A_{m_3}$ each one of them represents some $k' \log n$ bits from our input vectors. We handle these arrays as in the regular case explained above. The crucial point one should observe is that for each input vector v there must be $k' \log n$ bits among these m_3 options that contains at least $\frac{c_1'}{c} k$ of the ones in v. Let A_i be the array representing $k' \log n$ bits out of the m_3 options that contains the maximum number of ones in v. We are guaranteed that v will appear in at most $n^{k' - \frac{c_1'}{c} k'}$ lists in A_i. We continue the solution as in the regular case. Following the analysis of the regular case, we have that the total space of the data structure will be $O(n^{k'+1-\frac{c_1'}{c}k'} + n^{c-\frac{c_1'}{c}k'})$. As $k - \epsilon < k' \leq k$ and $c_1' = c_1 - \epsilon$, we get that the total space is $O(n^{k+1-\frac{c_1-\epsilon}{c}(k-\epsilon)} + n^{c-\frac{c_1-\epsilon}{c}(k-\epsilon)})$. Setting $k = c-1$ as before, we get that the space is $O(n^{c-\frac{c_1-\epsilon}{c}(c-1-\epsilon)})$. We can make this space complexity as close as we wish to the space complexity for the case k divides c by choosing ϵ whose value is very close to 0. Consequently, we have the following result (c_1 is the largest number that satisfies $\binom{c \log n}{c_1 \log n} = \tilde{O}(n^{1-\delta})$ for some $\delta > 0$):

▶ **Theorem 3.** *For every $\epsilon > 0$ the DivideByOnes algorithm solves OV indexing with truly sublinear query time using $O(n^{c-\frac{c_1-\epsilon}{c}(c-1-\epsilon)})$ space.*

3 TopLevelsQueryGraph: Second Space-Efficient Solution for OV indexing

There are two problems with the previous solution. The first one is the sharp separation between long lists (having $\tilde{O}(n)$ vectors) and short lists. For long lists we use a large amount of space and answer queries very quickly in constant time, while for short lists we just save the vectors in the lists and spend time in the query stage. The second problem is that each input vector is saved many times in different lists.

3.1 Query Graph

In order to improve the space requirements for sublinear query time we introduce the notion of a *query graph*. The idea of the query graph is to create a tradeoff between query time and space and save each vector just once. We are now ready to define the query graph. A query graph is a directed acyclic graph $G = (V, E)$ such that each vertex v_i in V represents a boolean vector α_i of length $k \log n$. There is an edge $(v_i, v_j) \in E$ if the vectors α_i and α_j differ on exactly one element which is 0 in α_i and 1 in α_j. Following this definition the query graph can be viewed as a layered graph with $k \log n$ layers. The jth layer in this graph contains all the nodes v_i such that the number of ones in α_i is exactly j. All the edges from the vertices in the jth layer are directed to vertices in the $(j+1)$th layer. We call the layers for small values of j *top* layers and the layers with high values of j *bottom* layers.

Let W be a set of indices such that $W \subseteq [c \log n]$ and $|W| = k \log n$. We want each vertex v_i that represents a vector α_i to contain a list L_i of input vectors such that their elements in the indices specified by W are orthogonal to α_i. This is the same as we did in the previous construction as each entry in an array contains all input vectors that are orthogonal to the value of this entry in indices of the relevant part. However, instead of saving all input vectors that are orthogonal to α_i in the indices specified by W, we just pick all the input vectors that their elements in the indices specified by W are exactly the complements of the elements in α_i. All these vectors are saved in the list L_i in vertex v_i. Using these lists, we have the following easy observation:

▶ **Observation 1.** *Given a set $W \subseteq [c \log n]$ such that $|W| = k \log n$, the complete list of input vectors such that their values in the indices specified by W are orthogonal to some α_i can be recovered by concatenating all lists of vectors in the vertices that are reachable from vertex v_i in the query graph G.*

3.2 TopLevelsQueryGraph Algorithm

We start the preprocessing phase by constructing a query graph G. Now, following the last observation, instead of saving each vector many times in all the lists that their index is orthogonal to our query in the relevant indices (as suggested by the previous solution), we can save each vector in just one list and recover the original list by traversing G. We start the traversal from the vertex v_i such that the values of the query vector in the indices specified by W are equal to α_i. We can use any standard graph traversal algorithm to obtain all the input vectors that are orthogonal to the query vector in the indices specified by W. The number of vertices that we visit in the traversal of the query graph for a query vector q that have $k' \log n$ ones in the indices specified by W is $2^{k \log n - k' \log n} = n^{k-k'}$.

We can identify two types of nodes in the query graph. A node v_i that has an empty list L_i is considered a *black node*, otherwise it is considered a *white node.* We note that the number of white nodes is at most n and it can be $O(n)$ if the input vectors are split between many lists. In order to achieve a truly sublinear query time we would like the number of nodes we visit during the traversal in the query graph to be truly sublinear. Moreover, as the number of white nodes can be $\Theta(n)$ we need to make sure that the total number of white nodes we visit is truly sublinear even if we know how to avoid black nodes. As mentioned before, the number of nodes we visit during our traversal is $n^{k-k'}$ which is truly sublinear if we set $k - k' < 1$. This means that we need to handle queries that match some vertex v_i in the top levels of the query graph differently. For all vertices v_i in the x top levels of the graph we create a list L'_i of *all* input vectors that are orthogonal to α_i. Then, for each list L'_i we create a bitmap to quickly identify if there is a vector in the list that is orthogonal to our query. The size of each bitmap is n^{c-k}. The total number of bitmaps we create is $\tilde{O}(\binom{k \log n}{x \log n})$ for $x \leq k/2$ as the number of vectors in the jth level of the query graph is $\binom{k \log n}{j \log n}$ (we choose $j \log n$ positions for the ones in α_i out of $k \log n$ positions). Moreover, the number of layers is logarithmic in n. Thus, the total required space for handling the top layers of the query graph is $\tilde{O}(n^{c-k}\binom{k \log n}{x \log n})$. The binomial coefficient $\binom{k \log n}{x \log n}$ can be approximated by $n^{k \log k - x \log x - (k-x) \log (k-x)}$ using Stirling's approximation. So, the total space for the top layers is approximately $\tilde{O}(n^{c-k+k \log k - x \log x - (k-x) \log (k-x)})$.

Now, a query vector q that matches a vertex v_i in the x top levels can be answered in constant time by just looking at the proper entry in the bitmap of v_i. Otherwise, the number of vertices we need to traverse in the query graph will be at most n^{k-x} which is truly sublinear if $k - x < 1$. The problem is that the total number of vectors in the lists of these vertices can be $\theta(n)$. To overcome this problem, we change the way we handle any list L_i in the $(k - x) \log n$ bottom levels according to the number of elements in it. If the number of elements in the list is $O(n^{1-k+x})$ we do nothing - the elements are kept in the list with no special treatment. Otherwise, we save a bitmap over all the possibilities of the other bits in the query vector. The size of the bitmap, as before, is n^{c-k}. The number of lists that have more than $O(n^{1-k+x})$ elements is at most n^{k-x}. Therefore, the space for all the bitmaps of the long lists is n^{c-x}. We have that the total space of our data structure is $\tilde{O}(n^{c-k+k\log k - x\log x - (k-x)\log(k-x)} + n^{c-x})$. To obtain the best space complexity (while preserving the truly sublinear query time), we set k very close to 1.3 and x to 0.3. The space complexity of this solution using these values is approximately $\tilde{O}(n^{c-0.3})$. To conclude, we obtain the following result:

▶ **Theorem 4.** *The TopLevelsQueryGraph algorithm solves OV indexing with truly sublinear query time using approximately $\tilde{O}(n^{c-0.3})$ space.*

4 BottomLevelsQueryGraph: Third Space-Efficient Solution for OV indexing

We can use the query graph to obtain another solution to the OV indexing problem. This time we focus on the x bottom levels of the query graph. For each vertex v_i in the x bottom levels of the query graph we save a bitmap to quickly identify if there is an input vector such that (a) Its bits in the indices specified by W are the complements of α_i and (b) It is orthogonal to our query vector. The space we invest in these bitmaps is $\tilde{O}(n^{c-k}\binom{k\log n}{x\log n})$. Then, for every vertex v_i which is not in the x bottom levels of the query graph we save in its list L_i all the input vectors that are *orthogonal* to α_i, but *do not appear* in the any of the lists of the vertices in the x bottom. For every list L_i that its length is $\tilde{\theta}(n)$ we save a bitmap to get the answer in $\tilde{O}(1)$ time. Because we do not include in any list L_i vectors from the lists in the x bottom levels, we are guaranteed that each input vector appears in at most n^{k-x} lists. In our view of the query graph, this means that if an input vector appears in the list L_i of some vertex v_i it will be duplicated in the lists of all vertices that v_i is reachable from them. Consequently, the total number of vectors in all lists above the x bottom levels is at most n^{1+k-x}. Therefore, the number of bitmaps we will save for lists of size $\tilde{\theta}(n)$ is at most n^{k-x}. Each bitmap is of n^{c-k} space, so the size of all bitmaps is n^{c-x}. The total size of the data structure is again $\tilde{O}(n^{c-k+k\log k - x\log x - (k-x)\log(k-x)} + n^{c-x})$.

Upon receiving a query vector q, if it matches a vertex in one of the x bottom levels, we immediately get the answer by looking at the right entry in the bitmap in that vertex. Otherwise, we need to look not just at the bitmap of the vertex that matches our query, but rather we have to go over all the vertices v_i in the $(k-x)\log n$ level (the top level of the x bottom levels) such that α_i is orthogonal to q in the positions specified by W. In all these vertices we check in their bitmap if there is an input vector that is orthogonal to q. If $k - x < 1$ we ensure that the query time is sublinear in n. All in all, we obtain a solution that has the same query time and space complexities as the previous one using a different approach, as summarized in the following theorem:

▶ **Theorem 5.** *The BottomLevelsQueryGraph algorithm solves OV indexing with truly sublinear query time using approximately $\tilde{O}(n^{c-0.3})$ space.*

5 Space and Query Time Tradeoff for Solving OV indexing

In all the solutions we presented so far we tried to minimize the space usage and still achieve a sublinear query time. However, obtaining a tradeoff between the space and query time would be of utmost interest. We know how to obtain constant query time by using n^c space. But can we obtain, for example, $O(\sqrt{n})$ query time using just $n^{c-\epsilon}$ space for some $\epsilon > 0$? In the first method we have suggested there is an inherent problem to achieve this as all lists can have more than $O(\sqrt{n})$ vectors. In the second and third solutions we can improve the query time by choosing larger x. However, as x becomes $k/2$ the space of the data structure becomes $\tilde{O}(n^c)$. The following theorem demonstrates how to obtain any polynomial query time while consuming $O(n^{c-\gamma})$ space for some $\gamma > 0$.

▶ **Theorem 6.** *For any $\epsilon > 0$ there is a solution to OV indexing that its query time is $O(n^\epsilon)$ and the space complexity is $O(n^{c-\gamma})$ for $\gamma > 0$.*

Proof. The idea is to combine the second and third solutions. We can save bitmaps for both the x top levels and the x bottom levels of the query graph using $\tilde{O}(n^{c-k}\binom{k\log n}{x\log n})$ space. Then, for every vertex that is not in the x top or bottom levels we do the same as in the second solution - save a bitmap for every node whose list is of length $\theta(n^\delta)$ or more for some $\delta > 0$. The total cost of these bitmaps is $\tilde{O}(n^{c-k+1-\delta})$. When we get a query vector q that matches a vertex v_i in our query graph. If v_i is on the x top or bottom levels, we just check the right entry in the bitmap of v_i. Otherwise, we start a traversal from v_i to all the vertices that are reachable from it except those in the x bottom levels. The number of vertices we visit is at most $\binom{(k-x)\log n}{x\log n}$ if $k/3 < x$. This is approximately $\tilde{O}(n^{(k-x)\log k-x-x\log x-(k-2x)\log(k-2x)})$. It is easy to verify that as x gets close to $k/2$ the exponent of this expression is very close to 0. Therefore, the total query time is $\tilde{O}(n^{\delta+(k-x)\log(k-x)-x\log x-(k-2x)\log(k-2x)})$ as the query time in each vertex we visit is at most n^δ. By choosing suitable value of $k \geq 1$, $x < k/2$ and $\delta > 0$, we can obtain a query time of $\tilde{O}(n^\epsilon)$ for any $\epsilon > 0$ using a data structure that consumes $\tilde{O}(n^{c-\gamma})$ space for some constant $\gamma > 0$. ◀

6 The Reporting Version of OV indexing

In the reporting version of OV indexing, given a query vector q we are required not just to decide if there is a vector in S that is orthogonal to q, but rather we are required to report all input vectors in S that are orthogonal to q.

To solve this version we can use the same methods as we have described for the decision version. However, the only part of these solutions that does not support reporting is the use of bitmaps. Using a bitmap we can answer the query quickly if there is an input vector that is orthogonal to our query vector, but we are unable to discover the list of input vectors that are orthogonal to the query if there are such vectors. The following lemma demonstrates how to construct a data structure that uses almost the same space as a bitmap, but supports efficient reporting.

▶ **Lemma 7.** *Given n $c\log n$-length boolean vectors, there is a data structure that uses $\tilde{O}(n^c)$ preprocessing time and upon receiving a query vector v report on all t input vectors that are orthogonal to v in time $O(t\log n)$*

6.1 Improving The Query Time

We can remove the dependency on n in the query time as shown by the following theorem[1].

▶ **Theorem 8.** *Given n $c\log n$-length boolean vectors, there is a data structure that uses $\tilde{O}(n^c)$ space and upon receiving a query vector v report on all t input vectors that are orthogonal to v in $O(t)$ time.*

[1] All missing proofs appear in: https://arxiv.org/abs/1710.00586

We can plug in the data structure from the previous theorem into any of the three solutions for OV indexing and get solutions for the reporting version of OV indexing that have the same space usage (up to logarithmic factors) and just an additive $O(t)$ to the query time.

7 Reducing OV indexing to SetDisjointness

In this section we present a connection between OV indexing and the problem of SetDisjointness. In the problem of SetDisjointness we are given m sets $S_1, S_2, ..., S_m$ such that the total number of elements in all sets is N and after preprocessing them we need to answer queries of the following form: given a pair of indices (i, j), decide whether $S_i \cap S_j$ is empty or not. The problem can be generalized to k-SetDisjointness in which we are given as a query a k-tuple $(i_1, i_2, ..., i_k)$ and we are required to answer if the intersection $S_{i_1} \cap S_{i_2} \cap ... \cap S_{i_k}$ is empty or not. The SetDisjointness problem was the first problem used to show conditional lower bounds on space complexity(see [12, 24, 15, 23]). Therefore, it should be interesting to see the connection between our OV indexing problem and the fundamental problem of SetDisjointness. Other problems connected to SetDisjointness are discussed in [17]. Currently, the best known space-query time tradeoff for k-SetDisjointness is $S \times T^k = O(N^k)$, where S is the space complexity and T is the query time [11, 17].

We begin by presenting a simple reduction from OV indexing to k-SetDisjointness for $k = c \log n$. Given an instance of OV indexing with n $c \log n$-length boolean input vectors we can create an instance of k-SetDisjointness in the following way. We create $c \log n$ sets. The set S_i contains all the vectors that have 0 in their ith element. Then, given a query vector q that has ones in the elements whose indices are $(i_1, i_2, ..., i_k)$ all that we need in order to answer this query is to verify if the intersection $S_{i_1} \cap S_{i_2} \cap ... \cap S_{i_k}$ is empty or not. If it is empty then we know that there is no input vector that has zeroes in all the position of the ones in q, which means that no input vector is orthogonal to q. Otherwise, there is an input vector which is orthogonal to q.

We would like to show this reduction to other values of k, especially small and constant. The idea is to use the first solution that we have suggested to obtain the following result:

▶ **Theorem 9.** *There is a reduction from OV indexing to k-SetDisjointness that can be used to solve OV indexing with truly sublinear query time and $O(n^{c-\gamma})$ space for some $0 < \gamma < 1$.*

8 OV indexing for Random Input

The solution to OV indexing that we have described in Section 3 is limited by the tradeoff between the bitmaps for the lists in the top levels of the query graphs (lists in vertices v_i such that α_i has a small number of ones) and the bitmaps for long lists in the bottom levels of the query graph. Therefore, we may improve the solution by making the lists in the bottom levels short, as for short lists we only save the elements themselves. We also note that we can also benefit from making the lists in the bottom levels very long, since their number is small. Consequently, the costly lists are those that are not too short and not too long.

In the solution we have presented in Section 3, we pick a set W of the indices for the query graph. Our solution works for any choice of W, but the question is whether there is a choice of W that will make the list shorter or longer, so we can utilize it for a more compact solution to OV indexing. In the following lemma we show that for random input vectors that are uniformly distributed the probability for choosing W such that there are lists that are not short is small.

▶ **Lemma 10.** *The size of every list L_i in the query graph of the second solution of OV indexing is at most logarithmic w.h.p. for any choice of W (the set of $k \log n$ indices) on random input vectors, where $k \geq 1$.*

The last lemma guarantees that on random input vectors for any choice of W the length of all lists in the query graph are supposed to be of length at most $4c \log n$ w.h.p. Therefore, instead of saving bitmaps for both lists in the top levels of the query graph and long lists in the bottom levels of the query graph, we need to save bitmaps just for the former as the latter do not exist. Consequently, for $c \log n$-length input vectors we just create the query graph with nodes representing $\log n$-length vectors and save bitmaps for the top $\delta \log n$ levels, for some $\delta > 0$. The space required by these bitmaps is $n^{c-1+\epsilon}$, for some $\epsilon > 0$ that can be as small as possible by choosing appropriate small value for δ. To conclude, we have obtained the following result:

▶ **Theorem 11.** *OV indexing on random input vectors can be solved in expected truly sublinear query time using $O(n^{c-1+\epsilon})$ space, for any $\epsilon > 0$.*

This improved space complexity for random input makes it tempting to think that the same property holds even for worst case input. More specifically, it is enough to have just one W that will map all input vectors to short or long lists. It turns out that for worst case input this cannot be achieved. In the following lemma we show how to create a worst case input vectors such that many lists in the query graph are neither too short nor too long.

▶ **Lemma 12.** *There exist n $c \log n$-length input vectors such that for all $W \subseteq [k \log n]$ there are $\Theta(n)$ vectors that are mapped by h_W to lists of size between $n^{1/6}$ and $n^{2/3}$*

In the last lemma we can obtain values other than $n^{1/6}$ and $n^{2/3}$ by changing the basic block size from $0.5 \log n$ to some other $r \log n$ for $r > 0$.

This demonstrates that for worst-case input vectors, as opposed to random input vectors, there can be $O(n)$ vectors that are mapped to lists that are neither too short nor too long. The exact size can be controlled by proper choices of block and group size.

9 Further Research

In this paper we presented several algorithms to solve OV indexing that obtain truly sublinear query time and require $O(n^{c-\gamma})$ space for some constant $0 < \gamma < 1$. For random input vectors we demonstrated in Section 8 how to obtain sublinear query time solution to OV indexing using $O(n^{c-\gamma})$ for *any* $0 < \gamma < 1$. We note that the preprocessing time of all algorithms is polynomial in n.

The main question regarding OV indexing, following this paper, is can one obtain a sublinear query time solution to OV indexing that requires only $O(n^{c-1})$ space. This question is interesting even if we allow an unlimited preprocessing time. We conjecture that there is no such solution to OV indexing:

▶ **Conjecture 2.** *There is no truly sublinear query time solution to OV indexing that requires only $O(n^{c-1})$ space.*

Even if that conjecture is false, it is of utmost interest to find the exact lower bound on the space requirements of OV indexing for both unlimited and polynomial preprocessing time. Finding the exact space requirements can be used to obtain conditional lower bounds on the *space* complexity of many problems known to be conditionally hard in terms of time based on OV.

References

1 Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1256–1271, 2016.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

3 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230, 2015.

4 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

5 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

6 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014.

7 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.

8 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014.

9 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015.

10 Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 451–462, 2002.

11 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010.

12 Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. *CoRR*, abs/1006.1117, 2010.

13 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100, 2004.

14 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for EDGE CLIQUE COVER are probably optimal. In *Proceedings of the Twenty-Fourth Annual*

ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, pages 1044–1053, 2013.

15 Pooya Davoodi, Michiel H. M. Smid, and Freek van Walderveen. Two-dimensional range diameter queries. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium, Arequipa, Peru, April 16-20, 2012. Proceedings*, pages 219–230, 2012.

16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

17 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *To appear in WADS 2017*, 2017.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

20 Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2182–2189, 2017.

21 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

22 Daniel Moeller, Ramamohan Paturi, and Stefan Schneider. Subquadratic algorithms for succinct stable matching. In *Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, pages 294–308, 2016.

23 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014.

24 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 738–747, 2012.

25 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075, 2010.

26 Ronald L. Rivest. *Analysis of Associative Retrieval Algorithm.* PhD thesis, Stanford University, 1974.

27 Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM J. Comput.*, 5(1):19–50, 1976.

28 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524, 2013.

29 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

30 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1867–1877, 2014.

Non-approximability and Polylogarithmic Approximations of the Single-Sink Unsplittable and Confluent Dynamic Flow Problems*†

Mordecai J. Golin[1], Hadi Khodabande[2], and Bo Qin[3]

1 CSE Department, The Hong Kong University of Science and Technology, Hong Kong
golin@cse.ust.hk

2 CE Department, Sharif University of Technology, Teheran, Iran
khodabande@ce.sharif.edu

1 CSE Department, The Hong Kong University of Science and Technology, Hong Kong
bqin@cse.ust.hk

Abstract

Dynamic Flows were introduced by Ford and Fulkerson in 1958 to model flows over time. They define edge *capacities* to be the total amount of flow that can enter an edge *in one time unit.* Each edge also has a *length*, representing the time needed to traverse it. Dynamic Flows have been used to model many problems including traffic congestion, hop-routing of packets and evacuation protocols in buildings. While the basic problem of moving the maximal amount of supplies from sources to sinks is polynomial time solvable, natural minor modifications can make it NP-hard. One such modification is that flows be *confluent,* i.e., all flows leaving a vertex must leave along the same edge. This corresponds to natural conditions in, e.g., evacuation planning and hop routing.

We investigate the *single-sink Confluent Quickest Flow* problem. The input is a graph with edge capacities and lengths, sources with supplies and a sink. The problem is to find a confluent flow minimizing the time required to send supplies to the sink. Our main results include:

- *Logarithmic Non-Approximability.* Directed Confluent Quickest Flows cannot be approximated in polynomial time with an $O(\log n)$ approximation factor, unless $P = NP$.
- *Polylogarithmic Bicriteria Approximations.* Polynomial time $(O(\log^8 n), O(\log^2 \kappa))$ bicriteria approximation algorithms for the Confluent Quickest Flow problem where κ is the number of sinks, in both directed and undirected graphs.

Corresponding results are also developed for the *Confluent Maximum Flow over time* problem. The techniques developed also improve recent approximation algorithms for *static* confluent flows.

1998 ACM Subject Classification G.1.6, Optimization, G.2.1, Combinatorics, G.2.2 Graph Theory

Keywords and phrases Optimization, Approximation, Dynamic Flow, Confluent Flow

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.41

[*] The work of all three authors was partially supported by RGC Hong Kong CERG grant 16208415.
[†] A full version of the paper is available at [7], https://arxiv.org/abs/1709.10307.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 41; pp. 41:1–41:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Note: Due to space considerations, this extended abstract is missing many of the proofs of the theorems and lemmas stated. For complete proofs and accompanying diagrams, please see the full version of this paper at [7].

Network Flow problems are very well known. Their input is a graph network with *capacities* $c(e)$ on its edges. $c(e)$ is the maximum *flow* that can be pushed through e. The problem is usually to maximize the amount of flow that can be pushed through the network. By contrast, *Dynamic* network flows, introduced by Ford and Fulkerson [5] in 1958, around the same time as regular network flows, are not as well known. In Dynamic Flows, $c(e)$ becomes the amount of flow *that can enter e in one time unit* while edge *length* $\ell(e)$ is the time that it takes for a unit of flow to traverse e. Dynamic Flow problems need to consider the additional problem of *congestion*, which may arise while flow waits to enter an edge.

Dynamic flows have been used to model problems as diverse as traffic movement, evacuation protocols and hop-routing of packets. The *(Dynamic) Maximum Flow Over Time* problem is to find the maximum amount of flow that can be pushed from sources to sinks in a given amount of time. The *(Dynamic) Quickest Flow* problem is to find the minimum time in which a fixed amount of flow can be pushed from sources to sinks. In addition, there are *multicommodity-flow* versions which require specific amounts of flow between given source-sink pairs and *transshipment problems* versions which do not restrict which source's demands are pushed to which sinks. It is known that the *Quickest Multicommodity Flow Over Time* problem is NP-Hard [9] while the *Quickest Transshipment* problem can be solved in polynomial time [11, 12]. Good surveys on Dynamic Flow problems and an introduction to its basic literature can be found in [15, 19, 23].

In basic (static) network flow problems, *splittable* flow is permitted, i.e., flow between a source and sink can be divided into multiple parts with each being routed over a different path. *Unsplittable flows* require that all flow between a particular source and sink be routed over only one path. *Confluent flows* require that all flow passing through a vertex must leave that vertex on the same edge[1] [2, 22]. Very recent work [21] has shown that, for the static single-sink case, unless $P = NP$, optimal unsplittable flows and optimal confluent flows do not have polynomial time constant-factor approximation[2] algorithms and, in fact, confluent flows can not be approximated to within a factor of $O(m^{1/2-\epsilon})$.

Confluent flows were introduced by [3], with applications including Internet routing [1], evacuation problems [17], and traffic coordination [15]. Several works have studied confluent flows that minimize the maximum *congestion* in routing networks e.g., [3, 2, 22]. However, these works usually do not take into consideration the transit time (or edge length) required for a packet to traverse a single link, though this parameter is usually considered in general network analyses (see, e.g., [10]). This immediately raises the Confluent Quickest Flow problem: Does there exists any routing scheme that minimizes the total time for sending all packets via a feasible (congestion bounded) confluent flow?

Another scenario in which *confluent dynamic flows* arise naturally is in modelling evacuation protocols. Let vertices represent locations to be evacuated and edges represent paths between vertices. A vertex's original supply is the number of people to be evacuated from it and a sink corresponds to an emergency exit. $\ell(e)$ is the time required to traverse path e; $c(e)$ is the number of people that can enter e in parallel, i.e., its width. The Confluent Flow

[1] Thus, confluent flows partition flows into edge disjoint in-trees, with the root of each tree being a sink.

[2] The objectives studied in [21] are the total *amount* of flow that can be confluently routed or the number of demands that can be confluently satisfied in the *static* flow.

restriction states that all people passing through a vertex must leave by the same edge, i.e., following a sign pointing "This way out". The Quickest Flow problem corresponds to placing the exit signs so as to minimize the time required to evacuate all people. The Maximum Flow Over Time problem corresponds to placing the signs so as to maximize the number of people that can be evacuated in a given amount of time.

The single-source single-sink version of the Confluent Quickest Flow problem is the polynomial-time solvable [19] *Quickest-Path Problem.* The Confluent Flow version of the multiple-source multiple-sink Quickest Transshipment problem was known to be polynomial-time solvable when G is a tree [17]. It was also known that, for general graphs, the single-sink Confluent Quickest Transshipment problem is NP-Hard [13]. But no other hardness complexity results, and in particular, non-approximability results, were known for general G.

Our first results are that *Confluent Dynamic Flow* problems on directed graphs, both the Quickest Flow and Max Flow Over Time versions, cannot be approximated to within $O(\log n)$ (n being the number of vertices in G) unless $P = NP$. Our results hold even when the graph has *a single sink.* Since, Multicommodity Flow and Transshipment are equivalent in the single-sink case we write "Quickest Flow" instead of "Quickest Multicommodity Flow" or "Quickest Transshipment".

In the other direction, we present *polylogarithmic bicriteria approximation* algorithms for both the *single-sink* Confluent Quickest Flow and Confluent Maximum Flow Over Time problems, in both directed and undirected networks. Note that known approximation algorithms for confluent flows are restricted to static networks in [3, 2, 22], and known optimal algorithms for dynamic confluent flows are restricted to special graphs, e.g., trees [17]. To the best of our knowledge, our algorithm is the first polylogarithmic approximation for these problems in general networks. These results are presented in Tables 1-2.

1.1 Single-Sink Dynamic Unsplittable/Confluent Flow Problems

The input to the problems is a dynamic flow network, i.e., a graph $G = (V, E)$ with n nodes and m edges, where edge e has capacity $c(e)$ and length $\ell(e)$. Also specified are a collection of sources $\{s_1, ..., s_k\} \subset V$ and a sink $t \in V$. The problems studied are:

- Quickest Flow Problem: Provides additional inputs $\{d_1, ..., d_\kappa\}$. d_i is the supply at source s_i. The problem is to find a flow minimizing the time it takes to send all of the d_i units of supply to sink t.
- Maximum Flow Over Time Problem: Provides additional input of time horizon T. The problem is to find a flow maximizing the amount of supply sent to the sink t within time horizon T. Supply at the s_i is unlimited.

We treat two different types of flow restrictions:

- *Unsplittable Flow:* All flow from s_i to t must pass along the same path P_i from s_i to t.
- *Confluent Flow:* Any two supplies that meet at a node must traverse an identical path to the sink t. In particular, at most one edge out of each node v is allowed to carry flow. Consequently, the support of the flow is a tree with all paths in the tree terminating at t.

1.2 Our Results

Section 3.1 presents a simple proof that, unless $P = NP$, $\forall \epsilon > 0$, it is impossible to construct a polynomial-time $3/2 - \epsilon$ approximation algorithm for the single-sink Quickest Flow problem when flows are restricted to be either unsplittable or confluent. This result holds for both directed and undirected graphs and even when the graph is restricted to have only one sink.

Table 1 Hardness or lower bounds on approx. ratio for the single-sink Quickest Flow problem.

Flow	Dynamic Network	Hardness or LB on Approx. Ratio
Confluent	Trees	Polynomial-Time Solvable [17]
Confluent	Directed/Undirected	NP-Hard [13]
Unsplittable	Directed/Undirected	$3/2-\epsilon$ (Thm. 6)
Confluent	Directed	$\Omega(\log n)$ (Thm. 7)*
Unsplittable/Confluent	Directed/Undirected	No $(\frac{15}{14}-\epsilon, 1+\alpha)$-Approx. (Thm. 10)*

* Corresponding results also hold for the single-sink Maximum Flow Over Time problem (Thms 8, 9, 11).

Table 2 Upper bounds on approximation ratio for variations of the Fixed-Sink Confluent Flow problem. The first three items are for uncapacitated problems but are included here because they serve as the internal building blocks for the approximation algorithms for the capacitated problems.

Network	Capacity	Objective	Sources	Sinks	UB on Approx. Ratio
Static	Uncapacitated	Min Congestion*	n	k $(k \le n)$	$O(\log^3 n)$ [3]†
Static	Uncapacitated	Min Congestion*	n	k $(k \le n)$	$1+\ln k$ [2]
Static	Uncapacitated	Min Congestion*	κ $(\kappa \le n)$	k $(k \le n)$	$O(\log^3 \kappa)$ (Thm. 13)†
Static	Node	Max Demand	n	1	$O(\log^6 n)$ with NBA[4] [22]
Static	Edge/Node	Max Demand	κ $(\kappa \le n)$	1	$O(\log^{10} \kappa)$ with NBA[4] (Thm. 22)†
Dynamic	Edge	Max Flow Over Time	κ $(\kappa \le n)$	1	$(O(\log^2 \kappa), O(\log^8 n))$ (Thm. 21)†
Dynamic	Edge	Quickest Flow	κ $(\kappa \le n)$	1	$(O(\log^8 n), O(\log^2 \kappa))$ (Thm. 20)†

* Minimize the maximum node congestion in a network that admits a feasible splittable flow satisfying all supplies.

† These results hold with high probability, or more precisely, with probability $1-n^{-c}$, where c is a constant.

Section 3.2 proves, for the confluent directed graph case, the much stronger result that unless $P = NP$, it is impossible to construct a polynomial-time $O(\log n)$ approximation algorithm for the single-sink Quickest Flow problem. The major tool used is a modification of a grid graph construction from [21] which was an extension of one pioneered by [8]. We note that our reduction is not the same as that in [21]. There, the objective function was the maximum *amount* of static flow that could be pushed. Here, the objective function is the minimum amount of *time* required to push the supplies. Our proof works by deriving new properties of the grid-graph. Section 3.3 extends the analysis to the Maximum Flow Over Time problem with our lower bounds on the approximation ratio being summarized in Table 1.

We also note that it might seem intuitive that, because confluent flows are "harder" than static flows, the non-approximability of confluent static flows, e.g., the result from [21], should immediately imply the non-approximability of confluent dynamic flows. This is not true, though. The two problems are trying to optimize very different things, making them incomparable. More specifically, in the static case, the goal is *Demand Maximization*, i.e., to find a subset of the demands of maximum total value that can be confluently routed. In the dynamic case, the goal is to find a confluent routing of ALL demands in minimal time. To appreciate the distinction it is instructive to examine confluent routing on *trees* where the static problem is NP-Hard [4] but the dynamic case is *polynomial-time* solvable [17].

Despite the non-approximability shown above for confluent dynamic flows, one might hope to create *bicriteria* (α, β) *approximations*[3]. However, in Section 4, we demonstrate that, for both directed and undirected graphs, there exists a constant $\alpha > 0$ such that, for any $\epsilon > 0$, there is no polynomial-time $(\frac{15}{14} - \epsilon, 1 + \alpha)$-approximation for the Unsplittable/ Confluent Quickest Flow problem, unless $P = NP$. Similar results are obtained for the Unsplittable/Confluent Max Flow Over Time problem. Our proof utilizes a reduction from the Bounded Occurrence 3-Dimensional Matching problem.

In contrast to the above we show, in Section 5, how to construct a $(O(\log^8 n), O(\log^2 \kappa))$-approximation for the Confluent Quickest Flow problem, where κ is now the number of sources, in polynomial time. To this end, we use the idea of routing a confluent flow in a static *monotonic network*, i.e., one in which each vertex is given an additional *vertex capacity* that satisfies that all edges go from a low-capacity node to a high-capacity one, which was introduced in [22]. Recall that in our original confluent flow problem the support of the flow is a tree. In that tree, a parent node never supports less flow than its child. So, intuitively, a feasible confluent flow requires its tree support to be monotonic. We develop new techniques (Theorem 16) that permit constructing, in polynomial time, a confluent flow that routes all supplies in a given monotonic network, while bounding both *node congestion* and *flow length*.

Via this monotonic technique, we build a novel multi-layer monotonic network and construct a confluent static flow on it which is finally re-routed to produce a confluent dynamic flow for our original graph problem. Our method guarantees that a *dynamic flow* can be found such that the total transit time is at most polylogarithmic factor times the optimal. Similarly, this also lets us develop a polynomial-time $(O(\log^2 \kappa), O(\log^8 n))$-approximation of the Confluent Maximum Flow Over Time problem.

Our technique mainly differs from that in [22] in constructing *length-bounded* confluent flows in *static* networks (which might be of independent interest). It also permits us to improve their approximation algorithms when not all vertices are sources. More specifically, recall that [22] gives an $O(\log^6 n)$ approximation algorithm for the demand maximization confluent flow problem, with the no-bottleneck assumption (NBA)[4]. If restricted to static networks, our technique can give an $O(\log^{10} \kappa)$ approximation for the same problem. If κ is bounded, for example, this gives a *constant* approximation, which is nearly optimal.

Our improvement to the approximation ratio comes through a combination of (i) a novel construction of the multi-layer network, and (ii) a new building block inside our monotonic network technique—a better routing approach for *uncapacitated* networks (Theorem 13). This will be discussed in more detail in Section 5.

Our Theorem 13 enables us to route confluent flows in uncapacitated monotonic sub-networks with congestion bounded by $\text{poly}(\log \kappa)$ instead of $\text{poly}(\log n)$. While this might look weak compared to the $1 + \ln k$ (k being the number of sinks) bound from [2] this is only used as a subroutine. In fact, the internal constructions of both [22] and our proofs for approximating the *capacitated* static problem build uncapacitated sub-networks which can have $\Theta(n)$ induced sources and sinks. Plugging in the bound of [2] would give a $\text{poly}(\log n)$ bound. We develop a new combinatorial argument that, combined with our new $\text{poly}(\log \kappa)$ bounds for uncapacitated monotonic sub-networks, gives a $\text{poly}(\log \kappa)$ bound for the capacitated one as well, yielding our Theorem 17. This leads us to the final improvement.

A chart presenting previously known results and our new ones is given in Table 2.

[3] These will be formally introduced in Definition 1.

[4] In node-/edge-capacitated networks, the NBA is that $\max_{v \in V} d(v) \leq \min_{v \in V} c(v)$, and $\max_{v \in V} d(v) \leq \min_{e \in E} c(e)$, respectively.

2 Preliminaries: Definitions and NP-Hard Problems

Let I be some input to an optimization problem, $OPT(I)$ be the *optimum* value to the given problem on I and $|I|$ be its *size*. As examples, I could be a dynamic flow problem on a graph with n vertices and m edges. We could have just as easily defined $|I| = m + n$.

We now define *bicriteria approximation*s for the two-objective optimization problem.

▶ **Definition 1** (Bicriteria Approximation). For any $\alpha, \beta > 0$, an (α, β)-approximation algorithm $\mathcal{A}$ for the two-objective optimization problem is a function that takes as input any parameter k and any instance I, and outputs a solution x such that

1. $\alpha f(x) \geq f(x^*)$, $g(x) \leq \beta k$, if the optimization problem is to find a solution x maximizing the cost function $f(x)$ subject to another cost function $g(x) \leq k$,
2. $f(x) \leq \alpha f(x^*)$, $\beta g(x) \geq k$, if the optimization problem is to find a solution x minimizing the cost function $f(x)$ subject to another cost function $g(x) \geq k$,

where x^* is the optimal solution for the input I and k.

We can actually define two different types of confluent flows:

▶ **Definition 2.** A flow in G is *node-confluent* if, for every vertex v, all flow leaving v leaves along the same edge. A flow in G is *edge-confluent* if, for every edge $e = (u, v)$ if all flow that passes through e must leave v through the same edge (v, w).

In this paper the term "confluent", when used alone, will denote node-confluence. When edge-confluence is needed (in some proofs) it will be explicitly specified.

Finally we will use the following NP-hard problems in our reductions:

▶ **Definition 3** (The Two-Disjoint Paths (Uncapacitated) Problem). Given a graph G and node pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$, decide if G contains paths P_1 from x_1 to y_1 and P_2 from x_2 to y_2 such that they are disjoint.

In undirected graphs the Two-Disjoint Paths (Uncapacitated) problem, for both edge-disjoint and node-disjoint paths, is polynomial-time solvable [20]. However, in directed graphs, the problem is NP-hard for both edge-disjoint and node-disjoint paths [6].

▶ **Definition 4** (The Two-Disjoint Paths (Capacitated) Problem). Let G be a (static) graph whose edges are labelled either α or β with $\beta \geq \alpha$. These labels are the *capacities* of the edges. Given node pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$, decide whether G contains paths P_1 from x_1 to y_1 and P_2 from x_2 to y_2 such that:

i. P_1 and P_2 are disjoint (node-disjoint or edge-disjoint);
ii. P_2 may only use edges of capacity β (P_1 may use both capacity α and capacity β edges).

The version of node-disjoint paths was proven to be NP-hard for undirected graphs by [8]. The version of edge-disjoint paths was proven to be NP-hard by [18].

▶ **Definition 5** (The Bounded Occurrence 3-Dimensional Matching Problem (BO3DM)). Suppose there are three disjoint sets $A = \{a_1, ..., a_n\}$, $B = \{b_1, ..., b_n\}$ and $C = \{c_1, .., c_n\}$, and a set $T = \{T_\mu \in A \times B \times C : \mu \in [m]\}$ such that each element of A, B, C occurs in the same constant number M of triples in T. The goal is to find the largest subset $T' \subset T$ such that all triples in T' are disjoint, i.e., no two elements of T' contain the same element of A, B, C.

As shown in [14], there exists an $\epsilon_0 > 0$ such that it is NP-hard to decide whether there exist n disjoint triples in T (*satisfiable* instance) or there exist at most $(1 - \epsilon_0)n$ disjoint triples in T (ϵ_0-*unsatisfied* instance).

Dynamic Flows. We first describe the mechanics of flow over one edge $e = (u, v)$ with capacity c and length ℓ. Suppose there are d units of supply on node u. Assume the discrete case in which d, c, ℓ are all integral and all d need to be moved from u to v. Items move in groups of size at most c, with one group entering e each time unit. Thus, the items are transported in $\lceil d/c \rceil$ groups. It takes ℓ time units for the first group to arrive at v. Since the groups left u at consecutive time units they arrive at v in consecutive time units. Thus, it requires $\lceil d/c \rceil - 1 + \ell$ time to move all items from u to v over e. Also, in both cases, if other items arrived at u wanting to enter e they would have to wait until all items already at u had departed before entering e.

Finally, we introduce some notations. A flow f is *feasible* if $\forall e \in E$, $f(e) \leq c(e)$. For any $e \in E$, we define its *edge congestion* as $EC(e) := f(e)/c(e)$. Under certain circumstance, we may introduce the *node capacity* $c(v)$ of $v \in V$, and define its *node congestion* $NC(v) := f^{out}(v)/c(v)$, where $f^{out}(v)$ is the total flow out of v. For a flow f, we let its edge congestion $EC(f) := \max_{e \in E} EC(e)$ and node congestion $NC(f) := \max_{v \in V \setminus \{t_1,...,t_k\}} NC(v)$, where $t_1, ..., t_k$ are sinks.

A static flow f can be specified by a collection of source-sink paths $\mathcal{P} = (P_1, ..., P_K)$ and corresponding flow values $f_1, ..., f_K$. We define the *length* of flow f as $L(f) := \max_{i \in [k]} L(P_i)$, where $L(P_i) := \sum_{e \in P_i} \ell(e)$ is the length of P_i. f is called as *L-length-bounded* for some $L \in \mathbb{R}^+$ if $L(f) \leq L$, i.e., no path in $\mathcal{P}$ has path length longer than L. Also, if all f_i's are identical, we call f as *uniform*.

3 Approximation Hardness for Unsplittable/Confluent Dynamic Flows

3.1 Constant Approximation Hardness of the Quickest Flows Problem

This section gives a simple proof that a polynomial-time constant approximation algorithm for the single-sink Unsplittable/Confluent Quickest Flow problem would imply $P = NP$.

▶ **Theorem 6.** *The single-sink Unsplittable/Confluent Quickest Flow problem in both directed and undirected graphs cannot be approximated to within a factor $3/2 - \epsilon$, for any $\epsilon > 0$, unless $P = NP$.*

3.2 Logarithmic Approximation Hardness of Confluent Quickest Flows

For the single-sink directed Confluent Quickest Flow problem we now derive a much stronger result than in the previous section. That is, it is NP-hard to even get a $O(\log n)$ approximation to the optimal solution.

To prove the logarithmic approximation hardness, we construct the following instance.

Hard instance. Before building the desired hard instance, we describe the dynamic half-grid network G_N. It can be viewed as an extension of the static half-grid graph in [21]. There are N rows (numbered from bottom to top) and N columns (numbered from right to left). All the edges in the i-th row and all the edges in the i-th column have capacity $1/i$. The i-th row extends as far as the i-th column and vice versa. The sink t, located at the bottom of the half-grid, is connected with the bottom node t_i of the i-th column by an edge of capacity $1/i$. Also, at the leftmost node of the i-th row, there is a source s_i with supply M^2/i, where M is a sufficiently large constant. We set all edge lengths as 1, and always enforce edge directions to be downwards and to the right.

Suppose we are now given an instance $\mathcal{I}$ of the directed node-disjoint version of the Two-Disjoint Paths (Uncapacitated) problem. We replace each 4-degree node in the half-grid by a copy of $\mathcal{I}$. Inside the copy, all edges have length 1. Consider the copy of $\mathcal{I}$ at the intersection of the i-th column and j-th row (with $j > i$) in G_N. That instance is incident to two edges of capacity $1/i$ and two edges of capacity $1/j$. Inside that $\mathcal{I}$, we let the edges of capacity $1/j$ be incident to x_1 and y_1, and the edges of capacity $1/i$ be incident to x_2 and y_2; we set all edge capacities to $1/i$. This completes the hard instance of directed confluent dynamic flows. Denote the constructed network as $\mathcal{G}$.

Utilizing $\mathcal{G}$, we obtain the logarithmic approximation hardness for the Confluent Quickest Flow problem. The proof works by showing that if we could get a logarithmic approximation, we could solve $\mathcal{I}$.

▶ **Theorem 7.** *The single-sink Confluent Quickest Flow problem in directed graphs cannot be approximated to a factor within $O(\log n)$, unless $P = NP$.*

3.3 Approximation Hardness of the Max Flow Over Time Problem

This section discusses the approximation hardness of the single-sink Unsplittable and Confluent Maximum Flow Over Time problem.

To derive the approximation hardness of the Unsplittable Maximum Flow Over Time problem, we will again reduce from the directed/undirected edge-disjoint version of Two-Disjoint Paths (Capacitated) problem. We construct the same network as in Section 3.1 and utilizing this constructed network, we show

▶ **Theorem 8.** *The single-sink Unsplittable Maximum Flow Over Time problem in both directed and undirected graphs cannot be approximated to a factor within $3/2 - \epsilon$, for any $\epsilon > 0$, unless $P = NP$.*

Although the above hard instance applies to the confluent flow, we present a stronger lower bound for the Confluent Maximum Flow Over Time in directed graphs.

▶ **Theorem 9.** *The single-sink Confluent Maximum Flow Over Time problem in directed graphs cannot be approximated to a factor within $O(\log n)$, unless $P = NP$.*

4 Constant Bicriteria Approximation Hardness of Dynamic Flows

This section first proves the NP-hardness of constant bicriteria approximations for the Unsplittable and Confluent Maximum Flow Over Time problems.

Our proof uses reductions from the BO3DM problem. Inspired by the reduction[5] presented in [8, 16], given an instance of BO3DM, we construct the following corresponding hard instance for the Unsplittable/Confluent Maximum Flow Over Time problem in undirected graphs. Note that the directed case is similar, except that we enforce all edge directions to point right. Suppose we are given an instance $\mathcal{I}$ of Bounded Occurrence 3-Dimensional Matching problem. Denote the μ-th triple T_μ as $(a_{p_\mu}, b_{q_\mu}, c_{r_\mu})$, where $p_\mu, q_\mu, r_\mu \in [n]$. We build an

[5] Even though we are reducing to the same problem note that our goal differs from [8], which aims at finding a maximum number of length-bounded edge-disjoint paths. For technical reasons, this requires us to develop a totally different bounding technique.

undirected graph $G = (V, E)$ where

$$\begin{aligned} V &= \{s, t\} \cup \{a_{il} : i \in [n], l \in [M-1]\} \cup \{s_i, b_i, c_i : i \in [n]\} \cup \{s'_\mu, x_\mu, y_\mu : \mu \in [m]\}, \\ E &= \{(s_i, s), (s, b_i), (c_i, t), (a_{il}, t) : i \in [n], l \in [M-1]\} \\ &\quad \cup \{(s'_\mu, s), (s, x_\mu), (y_\mu, a_{p_\mu l}) : \mu \in [m], l \in [M-1]\} \\ &\quad \cup \{(b_{q_\mu}, x_\mu), (x_\mu, y_\mu), (y_\mu, c_{r_\mu}) : \mu \in [m]\}. \end{aligned}$$

Hereby, G contains a vertex representing each element in the sets B and C, and $(M-1)$ copies of each element in A. Also, G contains a sink t, and sources s_i ($i \in [n]$), s'_μ ($\mu \in [m]$) as well as one more node s (s is removed when considering confluent flows). Meanwhile, for each triple T_μ in T, there are two vertices x_μ, y_μ to represent it. We connect s_i with s, and s with b_i for each $i \in [n]$; we also connect s'_μ with s, and s with x_μ for each $\mu \in [m]$. Similarly, we connect t with a_{il}, c_i for each $i \in [n]$ and $l \in [M-1]$. For each tuple, $T_\mu = (a_{p_\mu}, b_{q_\mu}, c_{r_\mu})$, we connect x_μ with b_{q_μ}, and y_μ with c_{r_μ} as well as $(M-1)$ copies of a_{p_μ}.

Edge capacities and lengths. All edge capacities are set as 1. Let each (s'_μ, x_μ) have length 5 (red edges), and each (y_μ, c_{r_μ}) have length 4 (green edges), and each (a_{il}, t) have length 3 (blue edges), and all other edges have length 2 (black edges). Finally, we set the time horizon $T = 14$ in the constructed graphs for the Unsplittable/Confluent Maximum Flow Over Time problems. Based on the constructed instance, we have

▶ **Theorem 10.** *There exists a constant $\alpha > 0$ such that, for any $\epsilon > 0$, there is no polynomial-time $(1+\alpha, \frac{15}{14} - \epsilon)$-approximation for the Unsplittable/Confluent Maximum Flow Over Time problem in both directed and undirected graphs, unless $P = NP$.*

To show the hardness of the Unsplittable/Confluent Quickest Flow problem, we construct an instance similar to Theorem 10, except that we let each source have supply 1, and have

▶ **Theorem 11.** *There exists a constant $\alpha > 0$ such that, for any $\epsilon > 0$, there is no polynomial-time $(\frac{15}{14} - \epsilon, 1+\alpha)$-approximation for the Unsplittable/Confluent Quickest Flow problem in both directed and undirected graphs, unless $P = NP$.*

5 Polylogarithmic Approximation for Confluent Dynamic Flows

5.1 Static Confluent Flows in Uncapacitated Networks with κ Sources

We now develop techniques for routing confluent flows in uncapacitated networks with $\kappa \leq n$ sources. Through Section 5.3, unless otherwise specified, the flow discussed is *static*.

▶ **Definition 12** (β-Satisfiable)**.** For any $\beta \in [0, 1]$, a supply d_i is *β-satisfiable* in flow f if at least a β faction of d_i can be sent to the sink via f. A flow f is β-satisfiable if all supplies are β-satisfiable in f.

Again, suppose $G = (V, A)$ is a static directed graph with supply $d(v)$ located at each $v \in V$. There exists a collection of sinks $\{t_1, ..., t_k\} \subset V$. We let κ be the number of non-zero supplies, and let all edge and node capacities be 1. We present

▶ **Theorem 13.** *In the directed uncapacitated network with κ uniform non-zero supplies, given a (splittable) 1-satisfiable flow f, there exists a randomized algorithm for finding a multi-sink confluent flow f' with the node congestion bounded by $O((NC(f))^2 \log^3 \kappa)$ whp*[6]*.*

[6] Throughout the paper, we use *whp* to mean with high probability, or more precisely, with probability $1 - n^{-c}$, where n is the number of nodes in the network and c is a constant.

Note that if κ is bounded and f is feasible, Theorem 13 can provide confluent flows with constant congestion.

Also, the support of the resulting flow is a collection of trees rooting at those sinks $t_1, ..., t_k$. We guarantee that the height of those trees can be bounded as below.

▶ **Lemma 14.** *Whp, the height of any tree constructed in the randomized algorithm is at most $O(NC(f) \log n)$.*

5.2 Static Length-Bounded Confluent Flows in Monotonic Networks

This section gives an algorithm for constructing a length-bounded confluent flow in *monotonic networks*, utilizing techniques developed in Section 5.1. A monotonic network is a special (static) directed graph with vertex capacities and no edges pointing in the direction of decreasing capacity. Formally,

▶ **Definition 15** (Monotonic Network)**.** A directed graph $G = (V, A)$ with node capacity $c(v)$ for each $v \in V$ is a *monotonic network* iff $c(u) \leq c(v)$ for every arc (u, v).

The network $G = (V, A)$ is the same as Section 5.1 except that here each node has capacity $c(v)$ and each edge has capacity 1. Our first step is to prove

▶ **Theorem 16.** *Let $G = (V, A)$ be a monotone network. Given a 1-satisfiable flow f with node congestion at most 1, one can, in polynomial time, construct a confluent 1-satisfiable flow with node congestion $O(\log^8 n)$ and flow length $O\left(L(f) \log n \log c_{\max} / \log \log n\right)$ whp, even without the no-bottleneck assumption.*

The idea is to first decompose the monotonic network into several sub-networks, and in each, construct length-bounded confluent flows with small node congestion. Connecting all confluent flows in those sub-networks, we can construct a confluent flow in the original network as desired. Our monotonic network technique incorporates a new parameter, namely the edge length, and, more importantly, our objective is to construct a *bicriteria* confluent flow, namely bounding *both* node congestion and length (note that in [22], only node congestion can be bounded). The main difference from [22] lies in that we embed our new algorithms for uncapacitated networks into the monotonic network routing.

Our technique can be further improved if we remove the length-bounded constraint. The key observation is that the sources in each sub-network are only induced by the given (splittable) flow that we would like to re-route into a confluent one. We can guarantee that, if the given flow is unsplittable, at most κ flow paths pass between two sequential sub-networks, inducing at most $O(\kappa)$ sources. This, combined with our new technique for uncapacitated networks, gives the improvement of the congestion from poly$(\log n)$ to poly$(\log \kappa)$.

▶ **Theorem 17.** *Let $G = (V, A)$ be a monotone network with a single sink. If there is 1-satisfiable flow f with node congestion at most 1, one can, in polynomial time, construct a confluent 1-satisfiable flow with node congestion $O(\log^8 \kappa)$ whp, under the NBA.*

5.3 Static Length-Bounded Confluent Flows in General Networks

Via the techniques developed above for monotonic networks, this section develops a polynomial-time algorithm for determining a length-bounded confluent static flow in general networks.

Suppose we are given a directed/undirected *edge-capacitated* network $G(V, E)$ (Section 5.2 dealt with *node* capacitated networks). Each node $v \in V$ has a supply $d(v)$ to be sent to the unique sink t. Our goal is to find *a subset of supplies* of maximum total value that can be routed via a confluent flow, whose flow length and edge congestion are both bounded.

To this end, we need to pre-process the network as follows. First, we ignore those demands of size at most $d_{\max}/2\kappa$, as they contribute at most half of the value of the optimal flow. Meanwhile, we round each supply up to the nearest power of 2, and group those with the same value together, producing $O(\log \kappa)$ groups of distinct supply sizes. To compute an approximation, we will separately route each supply group in G, and output the flow of the maximum value among all groups. Note that, this will lose a $O(\log \kappa)$ factor in the approximation ratio. Hence, we reduce the original problem to the uniform-supply case. Without loss of generality, by scaling, we can assume every supply is 1.

Second, we round each capacity up to the nearest power of 2, and assume all edges have capacity at most $\kappa d_{\max}$, i.e., $c_{\max} \leq \kappa d_{\max}$ as the extra capacity above this value is superfluous. Furthermore, when considering the uniform-supply case, those edges with capacity less than the supply size would never be used, as the supply should be routed confluently. Accordingly, we can assume each edge capacity is in $[1, \kappa]$ as $d_{\max} = 1$ in unit-supply case, and then there exist $O(\log \kappa)$ distinct capacity sizes.

Given a directed/undirected edge-capacitated network $G(V, A)$ with a single sink t, letting $k := \lfloor \log c_{\max} \rfloor + 1$, we construct the directed *k-layer (monotonic) network H*.

- **k layers.** Create k layers and k node sets $V(H_0), V(H_1), ..., V(H_{k-1})$, where $V(H_i) := V(G) \setminus \{t\}$ and the i-th layer contains $V(H_i)$.
- **Induced node capacities.** For the i-th node set $V(H_i)$ $(i = 0, ..., k-1)$, denote by u^i the i-th copy of node u, and let u^i have capacity 2^i.
- **Vertical arcs.** For each edge $(u, v) \in A(G)$, connect two vertical arcs (u^i, v^i) (and (v^i, u^i) if G is undirected) with capacity of 2^i in H, iff the capacity of (u, v) is at least 2^i $(i = 0, ..., k-1)$.
- **Horizontal arcs.** For $0 \leq i \leq k-2$, $\forall u \in V$, connect a horizontal arc (u^i, u^{i+1}) with capacity 2^i.
- **Arc lengths.** Let vertical arcs have the same length as arcs in G, and horizontal arcs have length 0.
- **$H := (V(H), A(H))$.** Set $V(H)$ as the union of $V(H_0), V(H_1), ..., V(H_{k-1}), \{t\}$ plus those dummy sinks, and set $A(H)$ as the collection of those vertical and horizontal arcs.
- **Supplies.** Place the supply of v at its copy v^0 in Layer 0.
- **Dummy sinks.** If there exists an edge (u, t) with capacity of 2^i, then create a copy t_u^j of t in Layer j and let the capacity of t_u^j be 2^j, for each $j = i, ..., k-1$. Connect the vertical arc (u, t_u^i) with capacity of 2^i, and the horizontal arc (t_u^j, t_u^{j+1}) with capacity of 2^j, for each $j = i, ..., k-2$. Finally, connect the arc (t_u^{k-1}, t) with capacity of 2^{k-1}.

Our multi-layer network can be viewed as a new construction enabling our length-bounded routing technique to work in edge-capacitated networks. Applying Theorem 16 yields:

▶ **Theorem 18.** *In the layered network H, given a (splittable) flow f for routing all unit supplies with node congestion at most 1, there exists a polynomial-time algorithm for constructing a 1-satisfiable confluent flow with node congestion $O(\log^8 n)$ and flow length $O\left(L \log^2 n / \log \log n\right)$ whp.*

Thus, via Theorem 18, we can obtain a confluent flow h in the k-layer network H with both node congestion and length being bounded. Nevertheless, since H is constructed from logarithmic copies of nodes in G, the constructed confluent flow h in H may induce a non-confluent flow in G, because some vertices v might contain logarithmic out-flow edges. We then show that there is a polynomial-time scheme for re-routing h into a confluent flow in the original network G. Also, although we bound *node* congestion in H, the original network G is in fact *edge*-capacitated and we are actually interested in the edge congestion. Fortunately, our construction of multi-layer networks can be patched. With the help of the monotonic structure and dummy sinks, we can bound the edge congestion.

Combining everything, we conclude that

▶ **Theorem 19.** *Suppose G is a directed/undirected edge-capacitated network with one sink. If there is an L-length-bounded confluent flow for routing all supplies with edge congestion at most 1 in G, then, there exists a polynomial-time algorithm for finding a confluent flow for routing* a subset of supplies *with value at least $\sum_{i\in[\kappa]} d_i/O(\log^2 \kappa)$, with edge congestion $O(\log^8 n)$ and flow length $O(L \cdot \log^3 n/ \log\log n)$ whp.*

5.4 Polylogarithmic Approximation for the Confluent Dynamic Flows

With the techniques developed the polylogarithmic approximation for the confluent dynamic problem can be shown to immediately follow. We do not use any storage at intermediate nodes.

▶ **Theorem 20.** *In directed/undirected, edge-capacitated dynamic networks, there is a polynomial-time algorithm that constructs an $(O(\log^8 n), O(\log^2 \kappa))$-approximation for the single-sink Confluent Quickest Flow problem whp.*

▶ **Theorem 21.** *In directed/undirected, edge-capacitated dynamic networks, there is a polynomial-time algorithm that constructs an $(O(\log^2 \kappa), O(\log^8 n))$-approximation for the single-sink Confluent Maximum Flow Over Time problem whp.*

Our technique can be restricted to static flows, yielding

▶ **Theorem 22.** *In directed/undirected, edge-/node-capacitated static networks that satisfy the no-bottleneck assumption, there is a polynomial-time algorithm that constructs an $O(\log^{10} \kappa)$-approximation for the single-sink Demand Maximization Confluent Flow problem whp.*

Acknowledgement. We would like to thank the authors of [22] for providing us with a pre-print of the full version of their paper.

References

1 A. Bley. Routing and capacity optimization for IP networks. In *Operations Research Proceedings 2007*, pages 9–16. Springer, 2008.

2 J. Chen, R. D. Kleinberg, L. Lovász, R. Rajaraman, R. Sundaram, and A. Vetta. (Almost) tight bounds and existence theorems for single-commodity confluent flows. *Journal of the ACM*, 54(4):16, 2007.

3 J. Chen, R. Rajaraman, and R. Sundaram. Meet and merge: Approximation algorithms for confluent flows. In *Proceedings of STOC'03*, pages 373–382. ACM, 2003.

4 D. Dressler and M. Strehler. Polynomial-time algorithms for special cases of the maximum confluent flow problem. *Discrete Applied Mathematics*, 163, Part 2:142–154, 2014.

5 L. R. Ford and D. R. Fulkerson. Constructing Maximal Dynamic Flows from Static Flows. *Operations Research*, 6(3):419–433, jun 1958.

6 S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

7 M. Golin, H. Khodabande, and B. Qin. Non-approximability and polylogarithmic approximations of the single-sink unsplittable and confluent dynamic flow problems (full version). *arXiv*, 2017. `arXiv:1709.10307`.

8 V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67(3):473–496, 2003.

9 A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379(3):387–404, 2007.

10 D. G. Harris and A. Srinivasan. Constraint satisfaction, packet routing, and the Lovasz Local Lemma. In *Proceedings of STOC'13*, pages 685–694, New York, NY, USA, 2013. ACM.

11 B. Hoppe and É. Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of SODA'94*, pages 433–441, 1994.

12 B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000.

13 N. Kamiyama. *Studies on Quickest Flow Problems in Dynamic Networks and Arborescence Problems in Directed Graphs.* PhD thesis, Kyoto University, 2009.

14 V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37(1):27–35, 1991.

15 E. Köhler, R.H. Möhring, and M. Skutella. Traffic networks and flows over time. In *Algorithmics of Large and Complex Networks*, pages 166–196. Springer, 2009.

16 S. G. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. In *Proceedings of FOCS'97*, pages 426–436. IEEE, 1997.

17 S. Mamada, T. Uno, K. Makino, and S. Fujishige. A tree partitioning problem arising from an evacuation problem in tree dynamic networks. *Journal of the Operations Research Society of Japan*, 48(3):196–206, 2005.

18 G. Naves, N. Sonnerat, and A. Vetta. Maximum flows on disjoint paths. In *Approximation, Randomization, and Combinatorial Optimization*, pages 326–337. Springer, 2010.

19 M. M. B. Pascoal, M. E. V. Captivo, and J. C. N. Clímaco. A comprehensive survey on the quickest path problem. *Annals of Operations Research*, 147(1):5–21, aug 2006.

20 N. Robertson and P.D. Seymour. Graph minors .XIII. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

21 F. B. Shepherd and A. Vetta. The inapproximability of maximum single-sink unsplittable, priority and confluent flow problems. *ArXiv*, abs/1504.00627, 2015.

22 F. B. Shepherd, A. Vetta, and G. T. Wilfong. Polylogarithmic approximations for the capacitated single-sink confluent flow problem. In *Proceedings of FOCS'15*, pages 748–758, 2015.

23 Martin Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, 2009.

Range-Efficient Consistent Sampling and Locality-Sensitive Hashing for Polygons[*][†]

Joachim Gudmundsson[1] and Rasmus Pagh[2]

[1] University of Sydney, Australia
joachim.gudmundsson@sydney.edu.au

[2] IT University of Copenhagen, Denmark
pagh@itu.dk

Abstract

Locality-sensitive hashing (LSH) is a fundamental technique for similarity search and similarity estimation in high-dimensional spaces. The basic idea is that similar objects should produce hash collisions with probability significantly larger than objects with low similarity. We consider LSH for objects that can be represented as point sets in either one or two dimensions. To make the point sets finite size we consider the subset of points on a grid. Directly applying LSH (e.g. min-wise hashing) to these point sets would require time proportional to the number of points. We seek to achieve time that is much lower than direct approaches.

Technically, we introduce new primitives for *range-efficient* consistent sampling (of independent interest), and show how to turn such samples into LSH values. Another application of our technique is a data structure for quickly estimating the size of the intersection or union of a set of preprocessed polygons. Curiously, our consistent sampling method uses transformation to a geometric problem.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems, Geometrical problems and computations

Keywords and phrases Locality-sensitive hashing, probability distribution, polygon, min-wise hashing, consistent sampling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.42

1 Introduction

Suppose that you would like to search a collection of polygons for a shape resembling a particular query polygon. Or that you have a collection of discrete probability distributions, and would like to search for a distribution that resembles a given query distribution. A framework for addressing this kind of question is *locality-sensitive hashing* (LSH), which seeks to achieve hash collisions between similar objects, while keeping the collision probability low for objects that are not very similar. Arguably the most practically important LSH method is *min-wise* hashing, which works on any type of data where similarity can be expressed in terms of *Jaccard similarity* of sets, i.e., the ratio between the size of the intersection and the size of the union of the sets. Indeed, the seminal papers of Broder et al. introducing min-wise hashing [5, 6] have more than 1000 citations. Independently, Cohen [10] developed

[*] This work was supported under Australian Research Council's Discovery Projects funding scheme (project number DP150101134, Gudmundsson) and the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013 / ERC grant agreement no. 614331, Pagh).

[†] A full version is available at arXiv [13], https://arxiv.org/abs/1701.05290.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 42; pp. 42:1–42:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

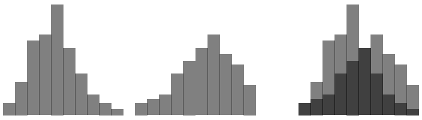


Figure 1 Left: Two probability distributions represented as histogram point sets. Right: The statistical distance can be computed from the Jaccard similarity.

estimation algorithms based on similar ideas (see also [11]). The basic idea behind min-wise hashing is to map a set S to $\text{argmin}_{x \in S} h(x)$, which for a strong enough hash function h gives collision probability equal (or close) to the Jaccard similarity.

If we represent discrete probability distributions by histograms there is a one-to-one relationship between the Jaccard similarity of two histograms and the statistical distance between the corresponding distributions. So a search for close distributions in terms of Jaccard similarity will translate into a search for distributions that are close in statistical distance, see Figure 1.

To make min-wise hashing well-defined on infinite point sets in the plane we may shift to an approximation by considering only those points contained in a finite grid of points. However, for a good approximation these sets must be very large, which means that computing a hash value $h(x)$ for each point $x \in S$, in order to do min-wise hashing, is not attractive.

1.1 Our results

We consider efficient locality-sensitive hashing for objects that can be represented as point sets in either one or two dimensions, and whose similarity is measured as the Jaccard similarity of these point sets. The model of computation considered is a Word RAM with word size at least $\log p$, where p is a prime number. We use integers in $U = \{0, \ldots, p-1\}$ (or equivalently elements in the field $\mathcal{F}_p$ of size p) to represent coordinates of points on the grid. Our first result concerns histograms with n values in U.

▶ **Theorem 1.** *For every constant $\varepsilon > 0$ and every integer N it is possible to choose an explicit hash function $H : U^n \to \mathbb{N}$ that has constant description size, can be evaluated in time $O(n \log p)$, and for which $\Pr[H(\mathbf{x}) = H(\mathbf{y})] \in [J - \varepsilon; J + \varepsilon]$, where $J = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}$ is the weighted Jaccard similarity of vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ of weight $\sum_i x_i = \sum y_i = N$.*

Our construction gives an explicit alternative to existing results on weighted min-wise hashing (see [14, 17]) whose analysis relies on hash functions that are fully random and cannot be described in small space. It was previously shown that a form of priority sampling based on 2-independence can be used to estimate Jaccard similarity of histograms [21], but similarity estimation is less general than locality-sensitive hashing methods such as weighted min-wise hashing.

We proceed to show the generality of our technique by presenting an LSH method for geometric objects. We will use approximation to achieve high performance even for "hard" shapes, and adopt the so-called *fuzzy model* [1]. In a fuzzy polygon, points that are "close" to the boundary (relative to the polygon's diameter) may or may not be included in the polygon. That is, given a polygon P and real value $0 < \phi \leq 1$, define the outer range $P^+ = P^+(w)$ to be the locus of points whose distance from a point interior to P is at most $w = \phi \cdot d(P)$, where $d(P)$ is the diameter of P. The inner range $P^- = P^-(w)$ of P is defined symmetrically.

Using the fuzzy model a *valid answer* to the Jaccard similarity of two polygons P_1 and P_2 w.r.t. ϕ is any value $\frac{X_\cap}{X_\cup}$ such that $A(P_1^- \cap P_2^-) \leq X_\cap \leq A(P_1^+ \cap P_2^+)$ and $A(P_1^- \cup P_2^-) \leq X_\cup \leq A(P_1^+ \cup P_2^+)$, where $A(\cdot)$ denotes the area of the region. To simplify the statement of the theorem we say that a polygon is *α-dense* in a rectangle I if for some value $\alpha > 0$ its area is at least a fraction α of the area of I. We use this to bound the time it takes to generate the sample points.

▶ **Theorem 2.** *For every choice of constants $\varepsilon > 0$, $\phi > 0$ and square $I \subseteq \mathbf{R}^2$ it is possible to choose an explicit random hash function H whose description size is constant, that can be evaluated in time $O((t \log p)/\alpha)$, where t is the time to test if a given point lies inside a polygon, and with the following guarantee on collision probability: Let $P_1, P_2 \subseteq I$ be polygons such that P_1^+ and P_2^+ are α-dense in I. Then $\Pr[H(P_1) = H(P_2)] \in [J - \varepsilon; J + \varepsilon]$, where J is some valid Jaccard similarity of P_1 and P_2 in the fuzzy model with parameter ϕ.*

It is an interesting problem whether the additive error in Theorems 1 and 2 can be improved to a multiplicative $1 + \varepsilon$ error.

In Section 5 we present further applications of our technique and show how a small summary can be constructed for a set $\mathcal{P}$ of polygons such that for any subset $\mathcal{Q}$ of $\mathcal{P}$, an estimate of the area of $\cap\mathcal{Q}$ and $\cup\mathcal{Q}$ can be computed efficiently in the fuzzy model with respect to ϕ.

Techniques. Our main technical contribution lies in methods for *range-efficient min-wise hashing* in one and two dimensions, efficiently implementing min-wise hashing for intervals and rectangles. More specifically, we consider intervals in U and rectangles in $U \times U$. The new technique can be related to earlier methods for *sampling* items with small hash values in one or more dimensions [20, 22]. (In fact, en route we obtain new hash-based sampling algorithms with improved speed, which may be of independent interest.) However, using [20, 22] to sample a single item is not likely to yield a good locality-sensitive hash function. The reason is that the hash functions used in these methods are taken from simple, 2-independent families and, as explained by Thorup [21], min-wise hashing using 2-independence does not in general yield collision probability that is close to (or even a function of) the Jaccard similarity. Instead we use a 2-phase approach: First produce a sample of k elements having the smallest hash values, and then perform standard min-wise hashing on a carefully selected *subset* of the sample using a *different* hash function.

We can combine and filter the samples to handle a variety of point sets that are not intervals or rectangles. To create a sample for a subset of a rectangle we can generate a sample of the rectangle, and then filter away those sample points that are not in the subset. This is efficient if the subset is suitably dense in the rectangle (which we ensure by working in the fuzzy model). To create a sample from the union of two sets, simply take the union of the samples. Theorems 1 and 2 are obtained in this way, and it would be possible to instantiate many other applications.

At the heart of our range-efficient sampling algorithms for one and two dimensions lies a reduction to the problem of finding an integer point (or integer points) in a given interval with small vertical distance to a given line. Such a point can effectively be found by traversing the integer convex hull of the line. Using a result of Charrier and Buzer [9] this can be done in logarithmic time. Thus, geometry shows up in an unexpected way in the solution.

1.2 Comparison with related work

We are not aware of previous work dealing with range-efficient locality-sensitive hashing. The most closely related work is on range-efficient *consistent* (or *coordinated*) sampling, which is a technique for constructing summaries and sketches of large data sets. The technique comes in two flavors: *bottom-k* (or min-wise) sampling, which fixes the sample size, and *consistent sampling* (or *sub-sampling*), which fixes the sampling probability. In both cases the idea is to choose as a sample those elements from a set $S \subseteq U$ that have small hash values under a random hash function $h : U \to [0; 1]$. If the sample size is fixed and some hash values are identical then an arbitrary tie-breaking rule can be used, e.g., selecting the minimum element. To make argmin uniquely defined, which is convenient, we take $\text{argmin}_{x \in I} h(x)$ to be the smallest value $y \in I$ for which $h(y) = \min_{x \in I} h(x)$. To denote the set of the k elements having the smallest hash values (with ties broken in the same way) we use the notation argmin_k. We focus on settings in which U is large and it is infeasible to store a table of all hash values.

In one dimension. Pavan and Tirthapura [20] consider the 2-independent family of linear hash functions in the field of size p, i.e., functions of the form $h(x) = (ax + b) \bmod p$. They show how to find hash values $h(x)$ below a given threshold Δ, where x is restricted to an interval I. (See also [2] for another application of this primitive.) The algorithm of Pavan and Tirthapura uses time $O(\log p + k)$, where k is the number of elements $x \in I$ with $h(x) \leq \Delta$. Using this in connection with doubling search leads to an algorithm finding the minimum hash value in time $O(\log^2 p)$. In this paper we show how to improve the time complexity:

▶ **Lemma 3.** *Let $h(x) = (ax+b)$ mod p, where p is prime and $0 \leq a, b < p$. Given $i_2 > i_1 > 0$ consider the interval $I = \{i_1, \dots, i_2\}$. It is possible to compute $\text{argmin}_{x \in I} h(x)$ (the* min-hash *of I) in time $O(\log |I|)$.*

We will argue in Section 2 that Lemma 3 can be applied repeatedly to subintervals to output the k smallest hash values (and corresponding inputs) in time $O(k \log |I|)$. The possibility of choosing $a = 0$ is included for mathematical convenience (to ensure 2-independence), though in most applications it will be better to choose $a > 0$ (which in addition makes argmin uniquely defined without a tie-breaking rule).

In more than one dimension. Tirthapura and Woodruff [22] consider another class of 2-independent functions, namely linear transformations on vectors over the field $\mathcal{F}_2$. Integers naturally correspond to such vectors, and for a *dyadic* interval I containing all integers that share a certain prefix, the problem of finding elements in I that map to zero is equivalent to solving a linear system of equations. Since an arbitrary interval can be split into a logarithmic number of dyadic intervals they are able to compute all the integers that map to zero in polylogarithmic time. The sampling probability can be chosen as an arbitrary integer power of two. This method generalizes to rectangles in dimension $d \geq 2$.

In this paper we instead consider linear, 2-independent hash functions of the form $(x, y) \mapsto (ax + by + c) \bmod p$. We do not know of a method for efficiently computing a min-hash over a rectangle for such functions, but we are able to efficiently implement consistent sampling with sampling probability $1/p$.

▶ **Lemma 4.** *Let $h(x, y) = (ax + by + c)$ mod p, where p is prime and $0 \leq a, b, c < p$. Given $i_1 < i_2$ and $j_1 < j_2$ consider $I = \{i_1, \dots, i_2\} \times \{j_1, \dots, j_2\}$. It is possible to compute $I' = \{(x, y) \in I \mid h(x, y) = 0\}$ in time $O((|I'| + 1) \log(\min(i_2 - i_1, j_2 - j_1)))$.*

For random a, b, c the expected size of the sample I' is $|I|/p$, and because of 2-independence the distribution of $|I'|$ is concentrated around this value. Compared to the method of [22] ours is faster, but has the disadvantage that the sampling probability cannot be chosen freely. However, as we will see this restriction is not a real limitation to our applications to locality sensitive hashing and size estimation.

From consistent sampling to LSH. Our technique for transforming a consistent sample to an LSH value is of independent interest. Thorup [21] shows that min-wise hashing using 2-independence does not in general yield collision probability that is close to (or even a function of) the Jaccard similarity. On the positive side he shows that bottom-k *samples* of two sets made using a 2-independent hash function can be used to estimate the Jaccard similarity J of the sets with arbitrarily good precision. However, this does not yield a locality-sensitive hash function with collision probability (close to) J, and obvious approaches such as min-wise hashing applied to the samples fails to have the right collision probability. Instead, we use consistent sampling (using a 2-independent family) followed by a stronger hash function for which min-wise hashing has the desired collision probability up to an additive error ε. This transformation yields the first LSH family for Jaccard similarity (with proven guarantees on collision probability) where the function can be:

- evaluated in $O(n + \text{poly}(1/\varepsilon))$ time on a set of size n, and
- described and computed in a constant number of machine words (independent of n).

Previous such functions have used either time per element that grows as ε approaches zero [16], or required description space that is a root of n (see [12]).

1.3 Preliminaries

We will make extensive use of 2-independence:

▶ **Definition 5.** A family of hash functions $\mathcal{H}$ mapping U to U is called *2-independent* if $\forall x_1, x_2, a_1, a_2 \in U$ with $x_1 \neq x_2$ and $h \in \mathcal{H}$ chosen uniformly we have

$$\Pr[h(x_1) = a_1 \wedge h(x_2) = a_2] = 1/|U|^2 \ .$$

It will be convenient to use the notation $x \pm \Delta$ for a number in the interval $[x - \Delta; x + \Delta]$.

Carter and Wegman [7] showed that the family $\mathcal{H}_1 = \{x \mapsto (ax + b) \bmod p \mid a, b \in U\}$ is 2-independent on the set $U = \{0, \ldots, p-1\}$ when p is a prime. Finally, we make use of ε-minwise independent families:

▶ **Definition 6.** A family of hash functions $\mathcal{H}$ mapping U to $\mathbb{N}$ is called *ε-minwise independent* if for every set $S \subset U$, every $y \in S$, and random $h \in \mathcal{H}$: $\Pr[h(y) = \min h(S)] = (1 \pm \varepsilon)/|S|$.

Indyk [16] showed that an efficient ε-minwise independent family mapping to a range of size $O(p/\varepsilon)$ can be constructed by using an $O(\log(1/\varepsilon))$-independent family of functions (e.g. polynomial hash functions). Dahlgaard and Thorup [12] showed that the evaluation time can be made constant, independent of ε, by using space $|U|^{\Omega(1)}$. If we only care about sets of size up to some number $\hat{n}$, this space usage can be improved to $(\hat{n}/\varepsilon)^{\Omega(1)}$.

2 Range-efficient bottom-k sampling in one dimension

The aim of this section is to show Lemma 3 and how it can be used to efficiently compute consistent as well as bottom-k samples. Together with the general transformation presented in Section 4 this will lead to Theorem 1.

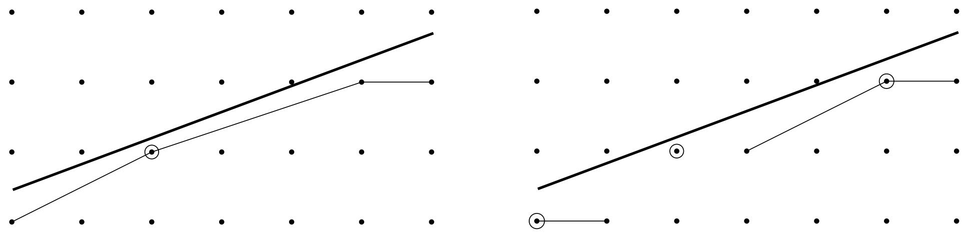


Figure 2 Illustration of reduction to integer convex hull.

Without loss of generality suppose $0 < i_1 < i_2 < p$, consider $I = \{i_1, \ldots, i_2\} \subseteq U$, and let $h \in \mathcal{H}_1 = \{x \mapsto (ax + b) \bmod p \mid a, b \in U\}$. To show Lemma 3 we must prove that $\operatorname{argmin}_{x \in I} h(x)$ can be computed in time $O(\log |I|)$. In case $a = 0$ this is trivial (just output i_1), so we focus on the case $a > 0$. We will show how the problem can be reduced to the problem of finding the integer point at the smallest (vertical) distance below the line segment

$$\ell = \{(x, (ax + b)/p) \mid x \in [i_1; i_2]\}. \tag{1}$$

To see this observe that for $x \in \mathbb{N}$ we have vertical distance $(ax + b)/p - \lfloor (ax + b)/p \rfloor$ between the line and the nearest integer point. Using the equality

$$(ax + b)/p - \lfloor (ax + b)/p \rfloor = ((ax + b) \bmod p)/p$$

we see that minimizing $(ax + b \bmod p)$ is equivalent to minimizing $(ax + b)/p - \lfloor (ax + b)/p \rfloor$, as claimed. Therefore it suffices to search for the point $(x, y) \in D = I \times \mathbb{N}$ below ℓ that is closest to ℓ. Since ℓ is a line, the point (x, y) must lie on the convex hull $CH(\ell)$ of the set of points in D that lie below ℓ, referred to as the "integer convex hull", see Figure 2. Clearly, the closest point will always be on the upper part of the hull, denoted $CH_L(\ell)$. Zolotykh [24] showed that $CH(\ell)$ consists of $O(\log(i_2 - i_1))$ line segments. To find a point on the integer convex hull with the smallest vertical distance to ℓ we will use a result by Charrier and Buzer [9].

▶ **Theorem 7.** *(Charrier and Buzer [9]) Given a line segment ℓ, the upper integer convex hull $CH_L(\ell)$ can be computed in $O(\log(i_2 - i_1))$ time, where i_1 and i_2 are the x-coordinates of the end points of ℓ.*

Charrier and Buzer initially assume that ℓ passes through the origin. However, they note (Section 7 in [9]) that this requirement is not needed. Thus, using their result on the line ℓ defined in (1) we obtain Lemma 3.

We now discuss how to use Lemma 3 to output the k smallest hash values (and corresponding inputs, i.e., the bottom-k sample) in time $O(k \log p)$. First compute $CH_L(\ell)$ and find the point $(x_1, y_1) \in CH_L(\ell)$ with the smallest vertical distance to ℓ. Next, split the problem into two subintervals; one for the part of ℓ in the x-interval $[i_1, x_1 - 1]$ and one for the part of ℓ in the x-interval $[x_1 + 1, i_2]$. Using a heap to find the integer point with smallest vertical distance in the intervals considered, we can repeat this process until k points have been found. To compute a consistent sample rather than the bottom-k sample we simply stop the procedure whenever we see an element with a hash value larger than the threshold.

▶ **Corollary 8.** *Let $h(x) = (ax + b) \bmod p$, where p is prime and $0 \le a, b < p$. Given $0 \le i_1 < i_2 < p$ consider $I = \{i_1, \ldots, i_2\}$. It is possible to compute the bottom-k sample (or the consistent sample of expected size k) from the interval I with respect to h in (expected) time $O(k \log |I|)$.*

It is an interesting problem whether it is possible to improve this bound to $O(k + \log |I|)$.

3 Rectangle-efficient consistent sampling

The aim and structure of this section are similar to those of Section 2, but now addressing the case where we want to do hashing-based sampling in a rectangle $I = \{i_1 \ldots, i_2\} \times \{j_1 \ldots, j_2\}$. Specifically, we prove Lemma 4 and show how one can use it to perform consistent sampling. This will be used in Section 4 to prove Theorem 2 and in Section 5 to construct an efficient data structure for estimating the size of intersections and unions of polygons. Assume without loss of generality that $0 \leq i_1 < i_2 < p$, $0 \leq j_1 < j_2 < p$ and $i_2 - i_1 \leq j_2 - j_1$. Consider the 2-independent family $\mathcal{H}_2 = \{(x, y) \mapsto (ax + by + c) \bmod p \mid a, b, c \in U\}$ and choose $h \in \mathcal{H}_2$. To prove Lemma 4 we have to argue that

$$I' = \{(x, y) \in I \mid h(x, y) = 0\} \tag{2}$$

can be computed in time $O((|I'| + 1) \log(i_2 - i_1))$. Similar to the previous section we will show how the problem can be reduced to the problem of finding all integer points below a line segment ℓ with a small vertical distance to ℓ.

To find all $(x, y) \in I$ for which $h(x, y) = (ax + by + c) \bmod p = 0$, as a first step we translate the function h such that we can consider input $y \in [0, y'_2]$. Specifically, we replace h with $h' : (x, y) \mapsto (ax + by + c') \bmod p$, where $c' = c + bj_1$, and consider inputs with $(x, y) \in [i_1, i_2] \times [0, j'_2]$, $j'_2 = j_2 - j_1$. This is equivalent to the original task since $h(x, y) = h'(x, y - j_1)$. Next note that for $x \in [i_1, i_2]$ and $y \in [0, j'_2]$:

$$(ax + by + c') \bmod p = 0 \quad \Leftrightarrow \quad y \equiv (-b^{-1}ax - b^{-1}c') \bmod p,$$

To simplify the expression set $q = -b^{-1}a$ and $s = -b^{-1}c'$. Then we have a zero hash value when $y = (qx + s) \bmod p = (qx + s) - kp$ for some positive integer k. Dividing by p and substituting $\alpha = q/p$ and $\beta = s/p$ we get $\frac{y}{p} = \alpha x + \beta - k$, where $x \in [i_1, i_2]$ and $y/p \in [0, j'_2/p]$. Now we can express the original problem as finding all $(x, k) \in [i_1, i_2] \times \mathbb{N}$ such that $\alpha x + \beta - k \in [0, j'_2/p]$. Consider the line segment $\ell' = \{(x, \alpha x + \beta) \mid x \in [i_1; i_2]\}$. An integer point (x', y') below ℓ' with $x' \in [i_1, i_2]$ and vertical distance at most j'_2/p to ℓ' corresponds to a point (x', y') such that $h'(x', y') = 0$.

To find all the points (x', y') that fulfill the restrictions we can apply the same technique as in Section 2. That is, compute the integer convex hull $CH_L(\ell')$ using the algorithm by Charrier and Buzer [9]. One difference from the setting of Section 2 is that we are interested in *all* integer points close to ℓ', but $CH_L(\ell')$ is guaranteed only to include one such point if it exists. This is handled by recursing on subintervals in which no points have been reported until we find an interval where the integer convex hull does not contain a point close to ℓ'. Recall that the time to output the integer convex hull is $O(\log(i_2 - i_1))$ by the result of Zolotykh [24], so the cost per point reported is logarithmic. This concludes the proof of Lemma 4.

3.1 Concentration bound

▶ **Definition 9.** An (ε, δ)-estimator for a quantity μ is a randomized procedure that, given parameters $0 < \varepsilon < 1$ and $0 < \delta < 1$, computes an estimate X of μ such that $\Pr[|X - \mu| > \varepsilon\mu] < \delta$.

For some $\alpha > 0$ consider an arbitrary set $S \subseteq I$, and the sample $S' = S \cap I'$ where I' is defined in (2). Let $1/p$ be the sampling probability. We now show that $p\,|S'|$ is concentrated around its expectation $|S|$ when p is not too large.

▶ **Lemma 10.** *For $\varepsilon > 0$, $p\,|S'|$ is an $(\varepsilon, p/(\varepsilon^2\mu))$-estimator for $\mu = |S|$.*

Proof. The proof is a standard application of the second moment bound for 2-independent indicator variables. For each point $q \in S$ let X_q be the indicator variable that equals 1 if $q \in I'$ and 0 otherwise. Clearly we have $|S'| = X$ where $X = \sum_{q\in S} X_i$, so $\mathbf{E}[pX] = p\sum_{q\in S}\mathbf{E}[X_i] = \mu$. By definition of I' the variables are 2-independent, and so $\mathbf{Var}(pX) = p^2\mathbf{Var}[X] \le p^2\mathbf{E}[X] = p\mu$. Now Chebyshev's inequality implies $\Pr[|pX - \mu| > \varepsilon\mu] < \mathbf{Var}(pX)/(\varepsilon\mu)^2 \le p/(\varepsilon^2\mu)$. ◀

To get an (ε, δ)-estimator we thus need $p \le \delta\varepsilon^2\mu$. The expected time for computing I' in Lemma 4 is upper bounded by $O(\mathbf{E}[|I'| + 1]\log p)$ which is $O((|I|/p + 1)\log p)$. If we choose $p = \Omega(\delta\varepsilon^2|S|)$, to get an (ε, δ)-estimator, and let $\alpha = |S|/|I|$ be the fraction of points of I that are also in S, then the expected time simplifies to $O(\log(p)/(\alpha\delta\varepsilon^2))$. That is, the bound independent of the size of S, has logarithmic dependence on p, and linear dependence on $1/\alpha$, $1/\delta$, and $1/\varepsilon$.

4 From consistent sampling to locality-sensitive hashing

We now present a general transformation of methods for 2-independent consistent sampling to locality-sensitive hashing for Jaccard similarity. Together with the consistent sampling methods in Sections 2 and 3 this will yield Theorems 1 and 2.

Thorup [21] observed that min-wise hashing based on a 2-independent family does not give collision probability that is close to (or a function of) Jaccard similarity. He observes a bias for a 2-independent family of hash functions based on multiplication, similar to the ones used in this paper. Thus we take a different route: First produce a consistent sample using 2-independence, and then apply min-wise hashing *to the sample* using a stronger hash function. The expected time per element is constant if we make sure that the sample has expected constant size.

Let constants $\varepsilon > 0$ and $\alpha > 0$ be given. For a point set $S \subseteq I$ with $|S| \ge \alpha|I|$ we produce a 2-independent sample $I' \cap S$ with sampling probability $1/p^*$, where $p^* = \Theta(\varepsilon^3\alpha|I|)$ is a prime number. This is possible assuming $|I| > 1/(\varepsilon^3\alpha)$ because there exists a prime p_i in every interval $\{2^i, \dots, 2^{i+1} - 1\}$, $i = 1, 2, 3, \dots$. Now select f at random from an $\varepsilon/4$-minwise independent family and define the hash value

$$H^*(S) = \operatorname*{argmin}_{x \in I' \cap S} f(x) \ . \tag{3}$$

▶ **Lemma 11.** *For $S, T \subseteq I$ with $|S|, |T| \ge \alpha|I|$ and $|I| > 12\,p/(\varepsilon^3\alpha)$ we have $\Pr[H^*(S) = H^*(T)] = \frac{|S\cap T|}{|S\cup T|} \pm \varepsilon$, where the probability is over the choice of I' and f.*

Proof. Consider the Jaccard similarity of samples $S' = S \cap I'$ and $T' = T \cap I'$:

$$J' = \frac{|S' \cap T'|}{|S' \cup T'|} = \frac{|S'| + |T'| - |S' \cup T'|}{|S' \cup T'|} \ .$$

Conditioned on a fixed I', the collision probability of $H^*(S)$ is $J' \pm \varepsilon/4$ by the choice of f. Thus it suffices to show that J' differs from J by at most $\varepsilon/2$ with probability at least $1 - \varepsilon/4$

By Lemma 10, $p \cdot |S' \cup T'|$ is an $(\varepsilon/8, \varepsilon/12)$-estimator for $|S \cup T|$ since $|S \cup T| \ge \alpha|I|$. Similarly, $p \cdot |S'|$ is an $(\varepsilon/8, \varepsilon/12)$-estimator for $|S|$ and $p \cdot |T'|$ is an $(\varepsilon/8, \varepsilon/12)$-estimator for $|T|$. The probability that all estimators are good is at least $1 - \varepsilon/4$, and in that case

$$J - \varepsilon/2 < \frac{|S \cap T| - (3\varepsilon/8)|S \cup T|}{|S \cup T| + (\varepsilon/8)|S \cup T|} \le J' \le \frac{|S \cap T| + (3\varepsilon/8)|S \cup T|}{|S \cup T| - (\varepsilon/8)|S \cup T|} < J + \varepsilon/2$$

as desired. ◀

We have not specified f. The most obvious choice is to use an $O(\log(1/\varepsilon))$-independent hash function [16]. Another appealing choice is twisted tabulation hashing [12] that yields constant evaluation time, independent of ε. The expected size of $S \cap I'$ is bounded by a function of ε and α. This means that we can combine twisted tabulation with an injective universe reduction step to reduce the domain of twisted tabulation to a (large) constant depending on ε and α.

Proof of Theorem 1. Consider a vector $\mathbf{x} = (x_1, \ldots, x_n) \in U^n$. We follow the folklore approach [14] of conceptually mapping each vector $\mathbf{x}$ to a set $P_\mathbf{x}$, such that the Jaccard similarity of $P_\mathbf{x}$ and $P_\mathbf{y}$ exactly equals the weighted Jaccard similarity of $\mathbf{x}$ and $\mathbf{y}$. In particular, it is easy to verify that this is the case if we let $P_\mathbf{x} = \{(i,j) \mid i = 1, \ldots, n;\ j = 1, \ldots, x_i\}$. Note that $P_\mathbf{x}$ and $P_\mathbf{y}$ both have size N. We will use the following class of hash functions from $U \times U$ to U:

$$H_2 = \{(x,y) \mapsto (ax + by + c) \bmod p \mid a,b,c \in U\} \ . \tag{4}$$

The 2-independence of H_2 follows from the arguments of Carter and Wegman [7]. A proof can be found in the full version of this paper [13]. When restricted to points of the form $(i,\cdot)$ for a fixed i, each function $h \in \mathcal{H}_2$ has a form suitable for Corollary 8 in Section 2. This means we can find the minimum for $P_\mathbf{x}$ restricted to a given column i in time $O(\log x_i)$. Using a heap to keep track of the smallest hash value from each column of $P_\mathbf{x}$ not (yet) reported in the sample, we can output all elements of $P_\mathbf{x}$ with a hash value smaller than any given threshold τ in time $O(\log p)$ per element. The threshold τ is chosen to match the desired sampling probability p^*.

Lemma 11 then says that we get the desired collision probability up to an additive error of ε. The expected time to hash is $O(n \log p)$ (to populate the priority queue) plus $O(\log p)$ times the expected number of samples. The expected number of samples $|S|/p$ is constant for every constant $\varepsilon > 0$, which gives the desired time bound in expectation.

It is possible to turn the expected bound into a worst case bound by stopping the computation if the running time exceeds $1/\delta$ times the expectation, which happens with probability at most δ. If we simply output a constant in this case the collision probability changes by at most δ (which we can compensate for by decreasing ε). ◀

Proof of Theorem 2. The proof is similar to the proof of Theorem 1 but with some added geometric observations. Let P_1 and P_2 be two polygons contained in I. As mentioned in the introduction, a *valid answer* to the Jaccard similarity of polygons P_1 and P_2 with respect to ϕ is any value $\frac{X_\cap}{X_\cup}$ such that $A(P_1^-(w_1) \cap P_2^-(w_2)) \leq X_\cap \leq A(P_1^+(w_1) \cap P_2^+(w_2))$ and $A(P_1^-(w_1) \cap P_2^-(w_2)) \leq X_\cup \leq A(P_1^+(w_1) \cap P_2^+(w_2))$, where $w_i = \phi \cdot d(P_i)$ for $i \in \{1,2\}$.

We now switch to considering the restrictions of $P_1^+(w_1/2)$ and $P_2^+(w_1/2)$ to a p-by-p grid of points whose enclosing rectangle contains I. See [15] for a survey on snapping points to a grid.

The grid points are identified in the natural way with integer coordinates in $[p] \times [p]$. We choose p such that the number of points inside I is $\Theta(p/\alpha)$ times the desired number of samples required for Lemma 11 to hold.

Let $L^+ = [i_1, i_2] \times [j_1, j_2]$ be the minimum bounding box of $I \cap P_1^+(w_1/2)$ and $I \cap P_2^+(w_2/2)$. The consistent sampling will be made on $P_i^+(w_i/2)$, $i \in \{1,2\}$. The reason for this is that

$$|P_1^+(w_1/2) \cap P_2^+(w_2/2)|/|P_1^+(w_1/2) \cup P_2^+(w_2/2)|$$

is a valid answer to the Jaccard similarity of P_1 and P_2 in the fuzzy model with respect to ϕ, which follows immediately from the below two inequalities that are proven in Lemma 13 (Section 5):

$$A(P_1 \cup P_2) \leq |(P_1^+(w_1/2) \cup P_2^+(w_2/2)) \cap I| \leq A(P_1^+(w_1/2) \cup P_2^+(w_2/2)), \text{ and}$$
$$A(P_1 \cap P_2) \leq |(P_1^+(w_1/2) \cap P_2^+(w_2/2)) \cap I| \leq A(P_1^+(w_1/2) \cap P_2^+(w_2/2)) \ .$$

Lemma 11 gives us the desired collision probability up to an additive error of ε. The expected time to hash is $O(\log p)$ plus $O(t \log p)$ times the expected number of samples, where t is the time to test if a given grid point lies inside a polygon. If we assume that P_1 and P_2 are α-dense in I, that is, there exists an $\alpha > 0$ such that $|P_1^+(w_1/2)|, |P_2^+(w_2/2)| > \alpha \cdot |L^+|$, then the expected number of samples is $|L^+|/(\alpha p)$ for any constants ε and ϕ, which gives the desired time bound in expectation. In many natural settings α is a constant, which implies that the expected number of samples is also constant.

5 Estimating union and intersection of polygons

In this section we consider the question: Given a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n preprocessed polygons in the plane, how efficiently can we compute the area of the union or the intersection of a given subset $\mathcal{Q} \subseteq \mathcal{P}$? In contrast to elementary approaches based on global, fully random sampling, our solution allows polygons to be independently preprocessed based on a small amount of shared randomness that specifies a *pseudorandom* sample.

Computing the area of the union of a set of geometric objects is a well-studied problem in computational geometry. One example is the Klee's Measure Problem (KMP). Given n axis-parallel boxes in the d-dimensional space, the problem asks for the measure of their union. In 1977, Victor Klee [18] showed that it can be solved in $O(n \log n)$ time for $d = 1$. This was generalized to $d > 1$ dimensions by Bentley [3] in the same year, and later improved by van Leeuwen and Wood [23], Overmars and Yap [19] and, Chan [8]. In 2010, Bringmann and Friedrich [4] gave an $O(\frac{dn}{\epsilon^2})$ Monte Carlo $(1+\epsilon)$-approximation algorithm for the problem.

A related question is the computation of the area of the intersection of n polygons in d-dimensional space. Bringmann and Friedrich [4] showed that there cannot be a (deterministic or randomized) multiplicative $(2^{d^{1-\epsilon}})$-approximation algorithm in general, unless NP=BPP. They therefore gave an additive ϵ-approximation for a large class of geometric bodies, with a running time of $O(\frac{nd}{\epsilon^2})$ assuming that the following three queries can be approximately answered efficiently: point inside body, volume of body and sample point within a body.

In this section we will approach the problem slightly differently. The approach we suggest is to produce a small summary of the set $\mathcal{P}$, such that given any subset $\mathcal{Q}$ of $\mathcal{P}$ the union and intersection of $\mathcal{Q}$ can be estimated efficiently. Unfortunately, the lower bound arguments by Bringmann and Friedrich [4] defeat any reasonable hope of achieving polynomial running time for arbitrary polygons. To get around the lower bounds we again adopt the approximation model proposed by Arya and Mount [1] (stated in Section 1.1).

Similar to the approach by Bringmann and Friedrich [4] we will also use sampling of the polygons to estimate the size of the union and intersection. However, compared to earlier attempts, the main advantage of our approach is that we generate the sample points (a summary of the input) in a preprocessing step and after that we may discard the polygons. Union and intersection queries are answered using only the summary. Also, we do not impose any restrictions on the input polygons. The drawbacks are that we only consider the case when $d = 2$ and the approximation model [1] we use is somewhat more “forgiving” than previously used models.

For each polygon P_i in $\mathcal{P}$, $1 \leq i \leq n$, let $w_i = \phi \cdot d(P_i)$, where $d(P_i)$ is the diameter of P_i and $0 \leq \phi \leq 1$ is a given constant. Let $\mathcal{Q}$ be the input to a union or intersection query, that is, $\mathcal{Q}$ is a subset of $\mathcal{P}$. To simplify the notations we will write $\cup\mathcal{Q}^+(w) = \cup_{P_i \in \mathcal{Q}} P_i(w_i)$ and $\cup\mathcal{Q}^-(w) = \cup_{P_i \in \mathcal{Q}} P_i(w_i)$. Define $\cap\mathcal{Q}^+(w)$ and $\cap\mathcal{Q}^-(w)$ symmetrically.

Following the above discussion, given a legal answer to a set intersection query $X = \cap\mathcal{Q}$ is any X' such that $\cap\mathcal{Q}^-(w) \subseteq X' \subseteq \cap\mathcal{Q}^+(w)$ and for a union query $X = \cup\mathcal{Q}$ a legal answer is any X' such that $\cup\mathcal{Q}^-(w) \subseteq X' \subseteq \cup\mathcal{Q}^+(w)$. It is immediate from the above definitions that for any polygon P and any $w \geq \sqrt{2}$ we have: $P^-(w) \subset P \subset P^+(w)$. We will use the number of integer coordinates, denoted $|P|$, within a polygon P to estimate the area of the polygon, denoted $A(P)$. Proofs of Lemmas 12, 13 and 15 can be found in the full version of this paper [13].

▶ **Lemma 12.** *For a polygon P having integer coordinates we have $A(P) \leq |P|$.*

To make the queries more efficient we will not estimate the number of integer coordinates in the intersection/union X of a query, instead we will estimate an approximation of $|X|$. We show:

▶ **Lemma 13.** *For any polygon P and $w \geq \sqrt{8}$: $A(P) \leq A(P^+(w/2)) \leq |P^+(w/2)| \leq A(P^+(w))$.*

As an immediate consequence of Lemma 13 we can use the consistent samples in $P_i^+(w_i/2)$, $1 \leq i \leq n$, for our estimates of the intersection and union, provided that $w_i \geq \sqrt{8}$. It remains to show how a summary of $\mathcal{P}$ can be computed and how the summary can be used to answer union and intersection queries.

Constructing a summary. For a given query $\mathcal{Q}$ containing $k \leq n$ polygons, let $P_{\min} = \text{argmin}_{P_i \in \mathcal{Q}} |P_i^+(w_i/2)|$, $P_{\max} = \text{argmax}_{P_i \in \mathcal{Q}} |P_i^+(w_i/2)|$ and let $d_{\min} = \text{argmin}_{P_i \in \mathcal{Q}} d(P_i)$. If $P_i = P_{\max}$ and $P_j = P_{\min}$ then we will write $P_{\max}^+(w) = P_i^+(w_i)$ and $P_{\min}^+(w) = P_j^+(w_i)$, respectively. Before giving the construction of summary and query algorithms we state two lemmas:

▶ **Lemma 14.** $|P_{\max}^+(w/2)| \leq |\cup \mathcal{Q}^+(w/2)| \leq k \cdot |P_{\max}^+(w/2)|$.

▶ **Lemma 15.** *If $\cap Q^-(w) \neq \emptyset$ and $\phi \cdot d_{\min} > \sqrt{8}$ then $\frac{\phi^2}{2} \cdot |P_{\min}^+(w/2)| \leq |\cap \mathcal{Q}^+(w/2)| \leq |P_{\min}^+(w/2)|$.*

We will use the rectangle-efficient consistent sampling technique described in Section 3 to generate a summary of $\mathcal{P}$ to estimate the area of $\cap\mathcal{Q}$ or $\cup\mathcal{Q}$, where $\mathcal{Q}$ is a given subset of $\mathcal{P}$.

The idea of the construction algorithm for the summary is simple. Let $X = \cap\mathcal{Q}^+(w/2)$ or $X = \cup\mathcal{Q}^+(w/2)$ depending on the query and, assume that $\phi \cdot d_{\min} > \sqrt{8}$. In a preprocessing step construct a summary of $\mathcal{P}$, denoted $\mathcal{S}$. The summary $\mathcal{S}$ will contain consistent samples for a number of different sampling rates. To answer a query, pick a minimum sampling rate $1/p$ that guarantees that the expected number of consistent samples in X is small but sufficient to guarantee an (ε, δ)-estimate of $|X|$. If X contains enough unique consistent samples then the algorithm reports an estimate of X, otherwise it iteratively increases the sampling rate with a constant factor until X contains sufficiently many unique consistent samples. From Section 3.1 we know that an (ε, δ)-estimator of X requires the sampling rate to be approximately $1/(\delta \cdot \varepsilon^2 \cdot |X|)$.

From Lemmas 14 and 15 we have that the smallest area that will ever be considered in a query $\mathcal{Q}$ has size at least $f_{\min} = \frac{\phi^2}{2}|P_{\min}^+(w/2)|$ and the largest area is at most $f_{\max} = n \cdot |P_{\max}^+(w/2)|$. To get an (ε, δ)-estimate of $|X|$ at least $1/\delta^2\varepsilon$ unique consistent samples are required to lie within X. As output from the above algorithm we get two data structures:

- $p[\ell]$: Returns a prime number between $[2^{\ell-1}, 2^{\ell}]$.
- $\mathcal{S}[P_i, \ell]$: Returns the set of consistent samples within $P_i^+(w_i/2)$, i.e., points satisfying the equation $(ax + by + c) \bmod p[j] = 0$. If the set is empty it returns FALSE.

Complexity. Consider the total number of consistent samples generated for a polygon P_i. The number of consistent samples is expected to increase with a factor of two in each iteration of the algorithm, that is, the expected total number of consistent samples form an exponentially growing geometric series which sums to $O(\frac{1}{\phi^2\delta^2\varepsilon} \cdot \frac{|P_i|}{|P_{\min}|})$. Summing up over all the polygons, the total number of consistent samples is bounded by $O(\frac{n}{\phi^2\delta^2\varepsilon} \cdot \frac{|P_{\max}|}{|P_{\min}|})$, which is also the expected size of the summary.

For the time complexity we first note that the above procedure can be implemented such that iterations where no consistent samples are expected to be generated are omitted without consideration. Since at least a fraction of $\phi/2$ of all consistent samples in the minimal bounding box of $P_i^+(w_i/2)$ is expected to lie within $P_i^+(w_i/2)$ (can be shown using a similar argument as in the proof of Lemma 15) the total number of generated consistent samples is expected to be at most a factor of $2/\phi$ greater than the number of consistent samples in the summary. Each consistent sample requires at most $O(\log|P_i|)$ time to generate, according to Theorem 4. If we assume that testing if a consistent sample lies inside a polygon can be done in time t then the expected time to build a summary of P is $O(\frac{n}{\phi^2\delta^2\varepsilon} \cdot \frac{|P_{\max}|}{|P_{\min}|} \cdot (t + \log|P_{\max}|))$. A description of union and intersection queries can be found in the full version of this paper [13]. We can now summarize the results in this section:

▶ **Theorem 16.** . *Given a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of polygons and three constants $\varepsilon, \delta > 0$ and $0 < \phi \leq 1$. If $\phi \cdot d(P_i) \geq \sqrt{8}$ for all $P_i \in \mathcal{P}$ then, in the fuzzy model with respect to ϕ, there exists a summary of size $O(\frac{n}{\phi^2\delta^2\varepsilon} \cdot \frac{|P_{\max}|}{|P_{\min}|} \cdot (t + \log|P_{\max}|))$ such that for any subset $\mathcal{Q}$ of $\mathcal{P}$ containing $k \leq n$ polygons an (ε, δ)-estimate of $\cup Q$ can be computed in $O(k/\delta\varepsilon^2)$ expected time and an (ε, δ)-estimate of $\cap Q$ can be computed in $O(\frac{k}{\phi^2\delta\varepsilon^2})$ expected time.*

Acknowledgement. We thank the anonymous reviewers for their useful comments.

References

1 S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry – Theory and Applications*, 17:135–152, 2000.

2 Y. Bachrach and E. Porat. Sketching for big data recommender systems using fast pseudo-random fingerprints. In *Proceedings of 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 459–471. Springer, 2013.

3 J. L. Bentley. Algorithms for Klee's rectangle problems. Unpublished note, Computer Science Department, Carnegie Mellon University, 1977.

4 K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry – Theory and Applications*, 43(6-7):601–610, 2010.

5 A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of International Conference on Compression and Complexity of Sequences (SEQUENCES)*, pages 21–29. IEEE, 1997.

6 A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

7 J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proceedings of 9th ACM Symposium on Theory of Computing (STOC)*, pages 106–112. ACM, 1977.

8 T. M. Chan. Klee's measure problem made easy. In *Proceedings of 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 410–419, 2013.

9 E. Charrier and L. Buzer. Approximating a real number by a rational number with a limited denominator: A geometric approach. *Discrete Applied Mathematics*, 157:3473–3484, 2009.

10 E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comp. Syst. Sci.*, 55(3):441–453, 1997.

11 E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *Proceedings of 26th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 225–234. ACM, 2007.

12 S. Dahlgaard and M. Thorup. Approximately minwise independence with twisted tabulation. In *Proceedings of 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 134–145. Springer International Publishing, 2014.

13 J. Gudmundsson and R. Pagh. Range-efficient consistent sampling and locality-sensitive hashing for polygons. *CoRR*, abs/1701.05290, 2017.

14 B. Haeupler, M. Manasse, and K. Talwar. Consistent weighted sampling made fast, small, and easy. *arXiv:1410.4266*, 2014.

15 J. Hershberger. Stable snap rounding. *Computational Geometry*, 46(4):403—416, 2013.

16 P. Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

17 Sergey Ioffe. Improved consistent sampling, weighted minhash and L1 sketching. In *Proceedings of 10th IEEE International Conference on Data Mining (ICDM)*, pages 246–255, 2010.

18 V. Klee. Can the measure of $\cup[a_i, b_i]$ be computed in less than $o(n \log n)$ steps? *American Mathematical Monthly*, 84:284–285, 1977.

19 M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM Journal on Computing*, 20(6):1034–1045, 1991.

20 A. Pavan and S. Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM Journal on Computing*, 37(2):359–379, 2007.

21 M. Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proceedings of 45th ACM Symposium on Theory of Computing (STOC)*, pages 371–380. ACM, 2013.

22 S. Tirthapura and D. Woodruff. Rectangle-efficient aggregation in spatial data streams. In *Proceedings of 31st Symposium on Principles of Database Systems (PODS)*, pages 283–294. ACM, 2012.

23 J. van Leeuwen and D. Wood. The measure problem for rectangular ranges in d-space. *Journal of Algorithms*, 2(3):282–300, 1981.

24 N. Y. Zolotykh. On the number of vertices in integer linear programming problems. Technical report, University of Nizhni Novograd, 2000.

Maximum Induced Matching Algorithms via Vertex Ordering Characterizations*†

Michel Habib[1] and Lalla Mouatadid[2]

1 IRIF, CNRS & Université Paris Diderot, Paris & INRIA Paris, Gang project, France
Habib@irif.fr

2 Department of Computer Science, University of Toronto, Toronto, ON, Canada
Lalla@cs.toronto.edu

Abstract

We study the maximum induced matching problem on a graph G. Induced matchings correspond to independent sets in $L^2(G)$, the square of the line graph of G. The problem is NP-complete on bipartite graphs. In this work, we show that for a number of graph families with forbidden vertex orderings, almost all forbidden patterns on three vertices are preserved when taking the square of the line graph. These orderings can be computed in linear time in the size of the input graph. In particular, given a graph class $\mathcal{G}$ characterized by a vertex ordering, and a graph $G = (V, E) \in \mathcal{G}$ with a corresponding vertex ordering σ of V, one can produce (in linear time in the size of G) an ordering on the vertices of $L^2(G)$, that shows that $L^2(G) \in \mathcal{G}$ - for a number of graph classes $\mathcal{G}$ - without computing the line graph or the square of the line graph of G. These results generalize and unify previous ones on showing closure under $L^2(\cdot)$ for various graph families. Furthermore, these orderings on $L^2(G)$ can be exploited algorithmically to compute a maximum induced matching on G faster. We illustrate this latter fact in the second half of the paper where we focus on cocomparability graphs, a large graph class that includes interval, permutation, trapezoid graphs, and co-graphs, and we present the first $\mathcal{O}(mn)$ time algorithm to compute a maximum *weighted* induced matching on cocomparability graphs; an improvement from the best known $\mathcal{O}(n^4)$ time algorithm for the *unweighted* case.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Maximum induced matching, Independent set, Vertex ordering characterization, Graph classes, Fast algorithms, Cocomparability graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.43

1 Introduction

A *matching* in a graph $G(V, E)$ is a subset of edges $M \subseteq E$ where no two edges in M have a common endpoint, i.e. every pair of edges in M is at distance at least one in G. An *induced matching* in G is a matching that forms an induced subgraph of G, i.e. every pair of edges in the induced matching is at distance at least two in G. Induced matching was introduced in [33] by Stockmeyer and Vazirani, as an extension of the matching problem (known as the marriage problem) to the "risk-free" marriage problem. Stockmeyer and Vazirani showed that maximum induced matching is NP-complete on bipartite graphs. The same result was also

* This work was partially supported by NSERC.
† The full version is available at [19], https://arxiv.org/abs/1707.01245.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 43; pp. 43:1–43:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

proven by Cameron in [5]. Since its introduction, the problem has been studied extensively. Induced matchings appear in many real-world applications. For instance, the problem can be used to model uninterrupted communications between broadcasters and receivers [17]. In [1], it was used to model the maximum number of concurrent transmissions in wireless ad hoc networks. In [26], it was used to extract and discover storylines from search results. Induced matchings have also been used to capture a number of network problems, see for instance [21, 2, 14] for network scheduling, gathering, and testing.

The problem is NP-complete even on bipartite graphs of degree three, and planar bipartite graphs [27]. It is also hard to approximate to within a factor of $n^{1-\epsilon}$ and $\Delta_G^{1-\epsilon}$ unless P = NP [13], where Δ_G is the maximum degree of the graph G. In [31], it was shown that the problem is W[1]-hard in general, but planar graphs admit a linear size kernel.

On the tractable side, induced matching is polynomially solvable for a number of graph classes, including trees, weakly chordal, asteroidal-triple free, and circular arc graphs, as well as graphs of bounded clique width [5, 6, 7, 16, 17, 8, 22]. We refer the reader to [13], a survey by Duckworth et al. that contains most of the references and complexity results.

Most of the graph classes for which the problem is tractable have well defined intersection models. One of the main techniques used to show the problem is tractable for a graph class $\mathcal{G}$, is to show that given an intersection representation of a graph $G \in \mathcal{G}$, there exists an intersection representation of a graph $H \in \mathcal{G}$, such that $L^2(G) = H$, where $L^2(G)$ is the square of the line graph of G. In other words, one can show that these graph classes are closed under the operation of "taking the square of the line graph" ($L^2(\cdot)$ operation). Since computing a matching (resp. an induced matching) on a graph $G \in \mathcal{G}$ is equivalent to computing an independent set on $L(G)$, the line graph of G, (resp. on $L^2(G)$, the square of $L(G)$), by showing closure under $L^2(\cdot)$, the induced matching problem is tractable on $\mathcal{G}$ if and only if computing an independent set is tractable on $\mathcal{G}$.

A *vertex ordering characterization* is an ordering on the vertices of a graph that satisfies certain properties. A graph class $\mathcal{G}$ has a vertex ordering characterization if every $G \in \mathcal{G}$ has a total ordering of its vertices that satisfies said properties. In this work, we use vertex ordering characterizations to show that certain graph classes are closed under $L^2(\cdot)$. In particular, one can observe that lexicographic orderings on the edges of a given vertex ordering of G produces an ordering on the vertices of $L^2(G)$. Since many graph classes are characterized by vertex orderings, and are closed under the square of the line graph operation, it is natural to ask what these orderings on the edges produce as vertex orderings on $L^2(G)$. In [3], Brandstädt and Hoàng showed how to compute perfect elimination orderings of $L^2(G)$ when G is chordal.

In this work we show that almost all forbidden patterns on three vertices are "preserved" under the $L^2(\cdot)$ operation, under two algorithms that compute orderings on $L^2(G)$. This general theorem shows that graph families with certain vertex ordering characterizations are closed under the $L^2(\cdot)$ operation; and these orderings of $L^2(G)$ can be computed in linear time in the size of G. This property gives, in our opinion, the most natural way to approach this closure operation, and unfies the results on structural graph classes that have relied on geometric intersection models to show closure. Furthermore, being able to compute vertex orderings directly can be exploited algorithmically, since algorithms on the graph classes covered often rely on their vertex ordering characterizations.

Using two different rules ($\star$ and $\bullet$) to compute these orderings on $L^2(G)$, we show that both the $\star$ and the $\bullet$ rules preserve forbidden patterns in the square of the line graph. As a corollary, we get that threshold, interval, and cocomparability graphs – among other classes – are all closed under $L^2(\cdot)$, and their corresponding vertex ordering characterizations are all preserved under $L^2(\cdot)$. One of the classes we focus on is *cocomparability graphs*, a large graph class that includes interval, permutation, and trapezoid graphs.

In the second half of this work, we present a faster algorithm to compute a maximum *weight* induced matching for cocomparability graphs. Induced matching on cocomparability graphs has been studied first by Golumbic and Lewenstein in [17], then by Cameron in [6], where they both gave different proofs to show that cocomparability graphs are closed under the $L^2(\cdot)$ operation. In [17], they showed that this closure holds for k-trapezoid graphs using the intersection representation of k-trapezoid graphs; since cocomparability graphs are the union over all k-trapezoid graphs, the result holds for cocomparability graphs as well. Whereas in [6], Cameron used the intersection model of cocomparability graphs (the intersection of continuous curves between two parallel lines [18]) to conclude the result directly. Cocomparability graphs are characterized by a vertex ordering known as a *cocomparability* or *umbrella-free* ordering [25]. We use cocomparability orderings and the $L^2(\cdot)$ closure to present a $\mathcal{O}(mn)$ time algorithm to compute a maximum *weighted* induced matching for this graph class, which is an improvement over the $\mathcal{O}(n^4)$ time algorithm for the *unweighted* case – a bound one can achieve by computing $L^2(G)$ and running the algorithm in [11] on it.

The paper is organized as follows: In Section 2, we give the necessary background and definitions. In Section 3, we give the general theorem for a number of graph classes closed under the $L^2(\cdot)$ operation. In Section 4, we present the maximum weight induced matching algorithm and its analysis on cocomparability graphs. We conclude with a discussion on methods that fail, as well as future directions in Section 5.

2 Definitions & Preliminaries

We follow standard graph notation in this paper, see for instance [15]. $G = (V, E)$ denotes a *simple graph* (no loops, no multiple edges) on $n = |V|$ vertices and $m = |E|$ edges. $N(v)$ is the *open neighbourhood* of a vertex v. The *degree* of a vertex v is $\deg(v) = |N(v)|$. Δ_G denotes the maximum vertex degree in G. We often refer to an edge (u, v) as uv. The *distance* between a pair of vertices u and v, $\text{dist}_G(u, v)$, is the length of the shortest path between u and v in G. The *distance* between a pair of edges e_1, e_2, denoted $\text{edist}_G(e_1, e_2)$, is the minimum distance over all shortest paths connecting an endpoint of e_1 to an endpoint of e_2. The *square* of a graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ where $uv \in E^2$ if and only if $\text{dist}_G(u, v) \leq 2$. The *chromatic number* of a graph G, $\chi(G)$, is the minimum number of colours required to properly colour G, i.e, to assign colours to V such that adjacent vertices receive different colours. An *induced subgraph* H of G is a graph $H = (V_H, E_H)$ where $V_H \subseteq V$ and for all $u, v \in V_H, uv \in E$ if and only if $uv \in E_H$. A *matching* $M \subseteq E$ is a subset of edges no two of which share an endpoint. An *induced matching* $M^* \subseteq E$ is a matching in G where every pair of edges in M^* forms an induced $2K_2$, or alternatively every pair of edges in M^* is at distance at least two in G. An *independent set* $S \subseteq V$ is a subset of pairwise nonadjacent vertices.

Given a graph $G = (V, E)$, the *line graph* of G, denoted $L(G) = (E, L(E))$, is the graph on m vertices, where every vertex in $L(G)$ represents an edge in G, and two vertices in $L(G)$ are adjacent if and only if their corresponding edges share an endpoint in G. We write $L^2(G) = (E, L^2(E))$ to denote the square of the line graph of G.

It is a well known fact that a matching in G is equivalent to an independent set in $L(G)$ [4]. An induced matching on the other hand is equivalent to an independent set in $L^2(G)$ [5]. Two vertices e_i, e_j in $L^2(G)$ are adjacent, i.e. $e_ie_j \in L^2(E)$, if and only if they have one of the configurations in G and $L(G)$ as shown in Fig. 1. In particular, one can see

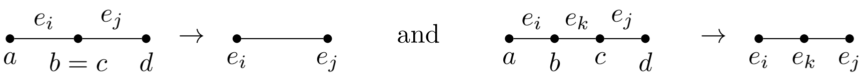


Figure 1 Configurations of $e_i, e_j \in E$ such that $e_i e_j \in L^2(E)$, and their representation in $L(G)$.

that two vertices are not adjacent in $L^2(G)$ if their corresponding edges induce a $2K_2$ in G.

Let $[n] = \{1, 2, \ldots, n\}$. An *ordering* σ of V is a bijection $\sigma : V \to [n]$. We write $\sigma = v_1, v_2, \ldots, v_n$. For a pair of vertices v_i, v_j, where $i, j \in [n]$ and $i < j$, we write $v_i \prec_\sigma v_j$ or $v_i \prec v_j$ if σ is clear in the context.

A *comparability graph* is a graph $G(V, E)$ which admits a transitive orientation of its edges. That is, if two edges $ab, bc \in E$ are oriented $a \to b$ and $b \to c$, then there must exist an edge $ac \in E$ oriented $a \to c$. A *cocomparability graph* is the complement of a comparability graph. Cocomparability graphs are a well studied graph family, see for instance [15]. Given a graph $G = (V, E)$, an ordering σ of G is a cocomparability ordering if and only if for every triple $a \prec b \prec c$, if $ac \in E$ then either $ab \in E$ or $bc \in E$, or both. If both $ab, bc \notin E$, we say that the edge ac forms an *umbrella* over vertex b. It is easy to see that a cocomparability ordering is just a transitive orientation in the complement. We have the following fact:

▶ **Fact 1.** *[25] G is a cocomparability graph iff it admits a cocomparability ordering.*

3 Vertex Orderings in the Square of the Line Graph

Many well-known classes of graphs can be characterized by vertex orderings avoiding some forbidden patterns, see for example the classification studied in [12] and further studied in [20]. Chordal, interval, split, threshold, proper interval, and cocomparability graphs are a few examples of such graph families. In this section, we show that graphs with certain forbidden induced orderings are closed under the $L^2(\cdot)$ operation. In particular, we show that almost all patterns on three vertices are preserved under $L^2(\cdot)$.

To do so, we construct an ordering on the vertices of $L^2(G)$, and thus on the edges of the original graph G, by collecting one edge at a time using different rules; either the $\star$ rule or the $\bullet$ rule. Formally, for a given graph $G = (V, E)$, let $\sigma = v_1, \ldots, v_n$ be a total ordering of V. Using σ, we construct a new ordering $\pi = e_1, \ldots, e_m$ on E as follows: For any two edges $e_i = ab$ and $e_j = uv$ where $a \prec_\sigma b$ and $u \prec_\sigma v$, we place $e_i \prec_\pi e_j$ if:

$$\textbf{Rule } (\bullet): \quad e_i \prec_\pi e_j \iff a \preceq_\sigma u \text{ and } b \preceq_\sigma v$$

$$\textbf{Rule } (\star): \quad e_i \prec_\pi e_j \iff \begin{cases} a \prec_\sigma u & \text{if } a \neq u \\ a = u \text{ and } b \prec_\sigma v & \text{o.w.} \end{cases}$$

We write $\pi^*(\sigma)$ (resp. $\pi^\bullet(\sigma)$) to denote the ordering constructed using the $\star$ (resp. $\bullet$) rule on σ. The ordering $\pi^*(\sigma)$ is the lexicographic ordering of E induced by σ, similar to the one used on chordal graphs in [3]. We will use ϕ^* (resp. $\phi^\bullet$) to denote the ordering $\pi^*(\sigma)$ (resp. $\pi^\bullet(\sigma)$) on $L(G)$, *including* the edges $L(E)$; and use σ^* (resp. $\sigma^\bullet$) to denote the ordering $\pi^*(\sigma)$ (resp. $\pi^\bullet(\sigma)$) on $L^2(G)$, *including* the edges $L^2(E)$.

▶ **Theorem 2.** *Given a graph $G = (V, E)$, its corresponding $L^2(G) = (E, L^2(E))$, and σ an ordering of V, if σ is p_i-free for a pattern p_i in Fig. 2, then $\sigma^\bullet$ is p_i-free as well.*

▶ **Theorem 3.** *Given a graph $G = (V, E)$, its corresponding $L^2(G) = (E, L^2(E))$, and σ an ordering of V, if σ is p_i-free for a pattern p_i in Fig. 2, then σ^* is p_i-free as well.*

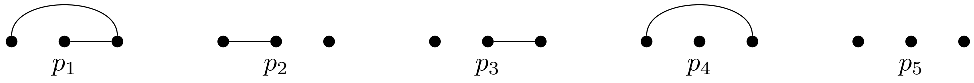


Figure 2 A list of forbidden patterns on three vertices.

Notice that the pattern p_4 forms an umbrella over the middle vertex. Thus the p_4-free orderings are precisely cocomparability orderings. Due to space constraints, and the fact that the $\star$ rule can easily be implemented in linear time, we will provide here the proof of Theorem 3 for the pattern p_4 only, since we will use this result on the second half of the paper. This partial proof of the theorem also gives a bit of intuition as to how the proofs for other patterns go. We refer the reader to [19] for full proofs of both Theorems 2 and 3.

Proof Of Theorem 3 for p_4. The proof is by contradiction, where we show if σ^* has an induced triple that satisfies a given pattern, then σ must also contain such a pattern. Call such a triple $e_1 \prec_{\sigma^*} e_2 \prec_{\sigma^*} e_3$. Let $e_1 = ab, e_2 = cd$, and $e_3 = ef$. Without loss of generality, suppose $a \prec_\sigma b, c \prec_\sigma d$, and $e \prec_\sigma f$. Thus $a \preceq_\sigma c \preceq_\sigma e$.

When a triple of vertices x, y, z induces a pattern p_i, we write $x, y, z \equiv p_i$. For the ordering $\prec_\sigma$ associated with σ, we drop the subscript and use $\prec$ instead, whereas we write $\prec_*$ to refer to the ordering $\prec_{\sigma^*}$. Recall that two vertices in σ^* are not adjacent iff they induce a $2K_2$ in G, and similarly, adjacent vertices in σ^* must have $\text{edist}_G \leq 1$ (Fig. 1).

Let σ be a p_4-free ordering and suppose σ^* is not, i.e. there exist $e_1 \prec_* e_2 \prec_* e_3$ such that $e_1e_3 \in L^2(E)$ and $e_1e_2, e_2e_3 \notin L^2(E)$. This p_4 configuration in σ^* implies the following adjacencies in G:

$$ac, ad \notin E \quad \text{and} \quad bc, bd \notin E \tag{1}$$
$$ce, de \notin E \quad \text{and} \quad cf, df \notin E \tag{2}$$
$$\text{edist}_G(e_1, e_3) \leq 1 \implies ae \vee af \vee be \vee bf \in E \tag{3}$$
$$a \prec c \prec e \tag{4}$$

$e_1e_3 \in L^2(E)$ implies either e_1 and e_3 are incident edges in G or their distance is at most two in $L(G)$, i.e, $\text{edist}_G(e_1, e_3) \leq 1$. Suppose first that e_1, e_3 are incident edges in G. This can happen if $e = b$ or $b = f$ since $a \prec c \prec e$.

If $e = b$, we have: $a \prec c \prec e = b \prec f$; and using (1), this implies $a, c, b \equiv p_4$.
If $b = f$, we have: $a \prec c \prec e \prec b = f$, and once again, $a, c, b \equiv p_4$. Thus, $\text{edist}_G(e_1, e_3) \leq 1$. That is, there exists $\alpha \in \{a, b\}, \beta \in \{e, f\}$ such that $(\alpha, \beta) \in E$.

In an attempt to satisfy (3), let's first suppose that $ae \in E$. By (4), $a \prec c \prec e$. By (1,2), $ae \in E$ would create an umbrella over c. Therefore $ae \notin E$. Suppose next that $af \in E$. Since $a \prec c \prec e \prec f$, it follows (using (1, 2)) that $af \in E$ would imply $a, c, f \equiv p_4$. Thus $af \notin E$. Suppose now that $be \in E$. Given that $a \prec b$ and $d \prec e$, we try to place b with respect to e. If $e \prec b$ then $a \prec c \prec e \prec b$ and $a, c, b \equiv p_4$ by (1). If $b \prec e$ then either $c \prec b$ or $b \prec c$. If $c \prec b$ then $a, c, b \equiv p_4$. If $b \prec c$ then $b \prec c \prec e$ and by assumption $be \in E$. Thus using (1, 2) $b, c, e \equiv p_4$. In all cases, we produce a p_4 if $be \in E$. Therefore $be \notin E$, and to satisfy (3), it remains that $bf \in E$. We place b with respect to f. By the same argument above, it must be that $b \prec f$. In fact, $a \prec b \prec c \prec f$ otherwise $a, c, b \equiv p_4$. But $b \prec c \prec f$ and (1, 2) imply $b, c, f \equiv p_4$. Thus $bf \notin E$. We just showed that in all scenarios, condition (3) cannot be satisfied without creating a p_4 in σ. Therefore if σ^* has a p_4 pattern, then σ must have a p_4 pattern. ◀

Table 1 $G \in \mathcal{G}$ iff $\exists\sigma$ of G that does not have any corresponding induced pattern [12].

$\mathcal{G}$	**Forbidden Patterns**
Threshold	p_1 and p_2
Interval	p_1 and p_4
Split	p_1 and p_3
Cocomparability	p_4
Chordal	p_1

Implementation: Since the $\star$ rule is just a lexicographic ordering on the edges, it is much easier to compute and to store than the $\bullet$ ordering. For this reason, we focus on the $\star$ rule in the remaining of this paper. We begin with the following observation:

▶ **Observation 4.** *$\pi^*(\sigma)$ as computed by the $\star$ rule can be constructed in $\mathcal{O}(m+n)$ time.*

Proof. Since the $(\star)$ rule is just a lexicographic ordering on the edges, it suffices to scan the ordering appropriately recording the endpoints of each edge. Formally, suppose G is given as adjacency lists, and let $\sigma = v_1, v_2, \ldots, v_n$ be a total ordering of G. For every $w \in V$, we sort the adjacency list of w according to σ. That is for every pair $v_i, v_j \in N(w)$, if $v_i \prec_\sigma v_j$ then v_i appears before v_j in $N(w)$. This can be done in $\mathcal{O}(m+n)$ time using standard techniques (see for instance [23]). We next construct the ordering $\pi^*(\sigma)$ on the edges of G as follows: Initially $\pi^*(\sigma)$ is empty. We scan σ from left to right, for every v_i in σ, and every neighbour v_j of v_i such that $i < j$, we append $e_k = v_iv_j$ to $\pi^*(\sigma)$. Adding these edges requires scanning $N(w)$ for every $w \in V$. Thus this process takes $\mathcal{O}(m+n)$ time. It is easy to see that this construction satisfies the $(\star)$ rule. We only append v_iv_j for $i < j$ to avoid inserting the same edge twice. The ordering $\pi^*(\sigma)$ we produce at the end of this process is precisely the ordering of the *vertices* of $\pi^*(\sigma), \phi^*$, and σ^*. Recall that these three orderings differ only in their edge sets and not on the ordering of their vertices. ◀

Therefore if a graph family $\mathcal{G}$ is characterized by the absence of patterns listed in Fig. 2, then if computing an independent set on $G \in \mathcal{G}$ is tractable, and uses the vertex ordering characterization of $\mathcal{G}$, it follows that computing a maximum induced matching is also tractable and reduces to computing an independent set on $L^2(G) \in \mathcal{G}$ using σ^*.

In this paper, we focus on graph families with forbidden patterns on three vertices (as shown in Fig. 2). To illustrate the consequences of Theorem 3, we list in Table 1 a number of graph families characterized by the absence of the patterns listed in Fig. 2 [4], and Corollary 5 follows immediately. For chordal graphs, Brandstädt and Hoàng gave a stronger result where they showed that not only is σ^* a p_2-free ordering, but that it is also a lexicographic breadth first search ordering [3].

▶ **Corollary 5.** *Vertex ordering characterizations of threshold, interval, split, cocomparability, and chordal graphs are all closed under the $L^2(\cdot)$ operation, and computing these orderings of $L^2(\cdot)$ can be done in linear time in the size of G.*

4 Application: Maximum Weight Induced Matching on Cocomparability Graphs

In this section, we focus on cocomparability graphs. We show how to compute a maximum *weight* induced matching on cocomparability graphs in $\mathcal{O}(mn)$ time, an improvement over $\mathcal{O}(n^4)$ time algorithm for the *unweighted* case. To do so, we use a result we presented in [24], where we give a linear time robust algorithm to compute a maximum weight independent set on cocomparability graphs in linear time. We begin by giving an overview of this algorithm, denoted CCWMIS (Cocomparability Maximum Weighted Independent Set), then present the maximum weight induced matching algorithm and its analysis to achieve the $\mathcal{O}(mn)$ runtime. Thus in the remaining of this section, G is a cocomparability graph and σ a cocomparability ordering. By [28], σ can be computed in linear time. By Theorem 3 and Observation 4, cocomparability orderings are closed under $L^2(\cdot)$ and can be computed in $\mathcal{O}(m+n)$ time. In particular, notice that the pattern p_4 is Fig. 2 is precisely the umbrella forbidden in cocomparability orderings. For clarity purposes, we refer the reader to [19] for a full illustration of the algorithm through an example.

4.1 Overview of the CCWMIS Algorithm

Let $G = (V, E, w)$ be a *vertex* weighted cocomparability graph, where $w : V \to \mathbb{R}_{>0}$. We compute a cocomparability ordering of G, $\sigma = v_1, v_2, \ldots, v_n$. For every vertex v_i in σ, we assign a set S_{v_i} of vertices. Initially S_{v_i} is empty for all $i \in [n]$. We write $w(S_{v_i})$ to denote the sum of the weights of the vertices in S_{v_i}: $w(S_{v_i}) = \sum_{z \in S_{v_i}} w(z)$. We use σ to compute a new ordering $\tau = u_1, u_2, \ldots, u_n$ of G, by scanning σ from *left to right* processing one vertex of σ at a time. Initially $\tau_1 = v_1$, and $S_{v_1} = \{v_1\}, w(S_{v_1}) = w(v_1)$. In general, at iteration i, when processing a given vertex v_i in σ, we scan τ_{i-1} from *right to left* looking for the rightmost nonneighbour of v_i in τ_{i-1}. Let u be such a vertex, if it exists. We construct $S_{v_i} = S_u \cup \{v_i\}$ with $w(S_{v_i}) = w(S_u) + w(v_i)$. If no such u exists, then $S_{v_i} = \{v_i\}$, and $w(S_{v_i}) = w(v_i)$. We show in [24] that the sets $\{S_{v_i}\}_{i=1}^{n}$ are independent sets.

We proceed to construct τ_i by inserting v_i into τ_{i-1}. Vertex v_i is inserted into τ_{i-1} so as to maintain an increasing ordering of the weighted sets $\{S_{v_k}\}_{k=1}^{i}$. That is, the vertices are ordered in $\tau = u_1, \ldots, u_n$ such that $w(S_{u_i}) \leq w(S_{u_j}), \forall i < j$. When all the vertices of σ have been processed, $\tau_n = \tau$ is constructed, we return S_{u_n} as a maximum weight independent set. In [24], we prove the following theorem:

▶ **Theorem 6.** *Let $G = (V, E)$ be a cocomparability graph. Algorithm CCWMIS computes a maximum weight independent set of G in $\mathcal{O}(m+n)$ time.*

4.2 The Weighted Maximum Induced Matching Algorithm (CCWMIM)

Now let $G = (V, E, w)$ be an *edge* weighted cocomparability graph where $w : E \to \mathbb{R}_{>0}$. Thus $L^2(G) = (E, L^2(E), w)$ is a *vertex* weighted cocomparability graph by Theorem 3 and [17, 6]. We compute a maximum weight independent set of $L^2(G)$ as shown in Algorithm 2.

By Theorem 6, Algorithm CCWMIS takes $\mathcal{O}(m+n)$ time. Thus, CCWMIS will take $\mathcal{O}(|E| + |L^2(E)|)$ time on $L^2(G)$. When G is dense, CCWMIS on $L^2(G)$ takes $\mathcal{O}(n^4)$ time. We give a careful implementation and analysis to achieve $\mathcal{O}(mn)$ runtime.

4.3 Implementation & Analysis of CCWMIM

Suppose the graph $G = (V, E, w)$, where $w : E \to \mathbb{R}_{>0}$, is given as adjacency lists. We compute $\sigma = v_1, \ldots, v_n$ in $\mathcal{O}(m+n)$ time using the algorithm in [28]. We construct $\pi^*(\sigma)$ in $\mathcal{O}(m+n)$ time using Observation 4.

Notice that we cannot use ϕ as input for the CCWMIS algorithm, since ϕ is not necessarily a cocomparability ordering. In fact, $L(G)$ is not necessarily a cocomparability graph; just consider the line graph of any large clique $K_{p>4}$. Notice also that the square edges in σ^* are necessary for Step 7 of the algorithm, when looking for a rightmost nonneighbour in τ_{i-1}.

Algorithm 1 CCWMIS

```
Input: $G = (V, E, w)$ a weighted cocomparability graph where $w : V \to \mathbb{R}_{>0}$
Output: A maximum weight independent set together with its weight
 1: Compute $\sigma = v_1, v_2, \dots, v_n$ a cocomparability ordering of $G$ [28].
 2: for $i \to 1$ to $n$ do
 3:     $S_{v_i} \leftarrow \{v_i\}$ and $w(S_{v_i}) \leftarrow w(v_i)$
 4: end for
 5: $\tau_1 \leftarrow (v_1)$                                              ▷ Constructing $\tau_i$
 6: for $i \to 2$ to $n$ do
 7:     Choose $u$ to be the rightmost non-neighbour of $v_i$ with respect to $\tau_{i-1}$
 8:     if $u$ exists then
 9:         $S_{v_i} \leftarrow \{v_i\} \cup S_u$ and $w(S_{v_i}) \leftarrow w(v_i) + w(S_u)$
10:     end if
11:     $\tau_i \leftarrow insert(v_i, \tau_{i-1})$      ▷ Insert $v_i$ into $\tau_{i-1}$ s.t. $\tau_i$ remains ordered w.r.t. $w(S.)$
12: end for
13: $z \leftarrow$ the rightmost vertex in $\tau_n$
14: return $S_z$ and $w(S_z)$
```

Algorithm 2 Cocomparability Weighted Maximum Induced Matching (CCWMIM)

```
Input: $G = (V, E, w)$ an edge weighted cocomparability graph where $w : E \to \mathbb{R}_{>0}$
Output: A maximum weight induced matching of $G$
 1: Compute $\sigma = v_1, v_2, \dots, v_n$ a cocomparability ordering of $G$
 2: Compute $\pi^*(\sigma) = e_1, e_2, \dots, e_m$ a cocomparability ordering of $L^2(G)$ using the $(\star)$ rule.
    ▷ The ordering only, not the square edges
 3: Use Algorithm 1 and $\pi^*(\sigma)$ to compute a maximum weight independent set of $L^2(G)$
```

We begin by looking at forbidden configurations of induced $2K_2$s in cocomparability orderings. Let $\sigma = v_1, \dots, v_n$ be a cocomparability ordering. Let $e_i = ab$ and $e_j = uv$ be two edges that induce a $2K_2$ in G. Without loss of generality, suppose $a \prec_\sigma b$ and $u \prec_\sigma v$. Since σ is a cocomparability ordering, the configurations of e_i, e_j that have either $a \prec u \prec b \prec v$ or $a \prec u \prec v \prec b$ as orderings cannot occur in σ, for otherwise σ would have an umbrella. This leaves the following configurations of the edges without umbrellas: $a \prec b \prec u \prec v$ or $u \prec v \prec a \prec b$.

Without loss of generality, suppose $a \prec_\sigma b \prec_\sigma u \prec_\sigma v$. Using the $(\star)$ rule, this configuration always forces $e_i \prec_\pi e_j$, i.e. $ab \prec_\pi uv$. Therefore, when we run Algorithm CCWMIS on $\pi^* = e_1, \dots, e_m$, we process elements of π^* from right to left, and thus we process $e_i = ab$ before processing $e_j = uv$.

Let $\tau = f_1, \dots, f_m$ be the new ordering being constructed by the algorithm CCWMIS using π^* as the ordering computed in Step 1. Initially, as per the algorithm, $\tau_1 = e_1$. In general, at iteration i, let $\tau_{i-1} = f_1, \dots, f_{i-1}$ be the ordering constructed thus far. Suppose e_i is the edge being processed. In Step 7 of Algorithm 1, looking for the rightmost nonneighbour of e_i in τ_{i-1} is equivalent to looking for an edge e that forms an induced $2K_2$ with e_i in σ, such that e is to the left of e_i in σ. When processing vertex e_i in π^*, we scan τ_{i-1} to find the rightmost nonneighbour of e_i in τ_{i-1}. Suppose such a vertex exists, and call it f_j. Since we are working in $L^2(G)$, to check if two vertices in $L^2(G)$ are adjacent, we need to check whether these edges are incident in G, or are at distance at most two in $L(G)$, as shown in Fig. 1. We proceed as follows.

Both σ and π^* are implemented using doubly linked lists. We construct three arrays A, B and F of sizes n, n, m respectively. All three arrays are initialized to zero; $A[t] = B[t] = 0, \forall t \in [n]$ and $F[i] = 0, \forall i \in [m]$.

Every vertex v_t in σ has a pointer to $A[t]$ and $B[t]$. Similarly, every vertex e_i in π^* has a pointer to $F[i]$. We sometimes abuse notation and talk about $A[w]$ to mean the position in array A that vertex w in σ points to. Furthermore, when we talk about vertex $e_i = uv$ in π^*, we always assume that $u \prec_\sigma v$.

For every vertex $e_i = v_t v_k$ in π, its corresponding entry $F[i]$ has four pointers $p_i^t, p_i^k, q_i^t, q_i^k$ that point respectively to $A[t], A[k]$ and $B[t], B[k]$. When processing vertex e_i, where $e_i = ab$, we update A as follows: For every neighbour z of vertex a, we set $A[z] = i$. Similarly, for every neighbour z of vertex b, we set $B[z] = i$. These updates to arrays A and B guarantee that every nonneighbour w of a has $A[w] \neq i$ and every nonneighbour w of b has $B[w] \neq i$. Therefore, for every edge $v_t v_k$ in G that forms an induced $2K_2$ with ab, the following (†) condition holds: $A[t] \neq i \wedge A[k] \neq i \wedge B[t] \neq i \wedge B[k] \neq i$ (†).

Thus, in order to find the rightmost nonneighbour of e_i in τ_{i-1}, we scan τ_{i-1} from right to left, and for every vertex we encounter $f_j = v_t v_k$, we check if one of $A[t], A[k], B[t], B[k]$ is equal to i. We return the first vertex in τ_{i-1} we encounter whose endpoints in G satisfy condition (†) above as the rightmost nonneighbour of e_i in τ_{i-1}. Updating arrays A and B requires $\mathcal{O}(\deg(a) + \deg(b))$ time. When scanning τ_{i-1}, for every vertex $f_j = v_t v_k$ in τ_{i-1}, we use the pointers $p_j^t, p_j^k, q_j^t, q_j^k$ in $F[j]$ to access $A[t], A[k], B[t], B[k]$. Checking these four entries takes constant time using the pointers provided.

It remains to analyze the number of constant checks we do, i.e. how many f_j vertices we check. In particular, this reduces to bounding the degree of e_i in $L^2(G)$.

Let $\deg_1(e_i)$ denote the degree of e_i in $L(G)$, and $\deg_2(e_i)$ denote the degree of e_i in $L^2(G)$. We have the following:

▶ **Claim 7.** *for a given edge $e_i = ab$, we have:* $\deg_2(e_i) \leq \sum_{\substack{v:av\in E\\ v\neq b}} \deg(v) + \sum_{\substack{v:bv\in E\\ v\neq a}} \deg(v)$.

Proof. It is clear that for a given edge $e_i = ab$, $\deg_1(e_i) = \deg(a) + \deg(b) - 2$. On the other hand, when computing $\deg_2(e_i)$, we take into account the degree of any vertex at distance at most two from either a, or b in G. In particular, the following holds:

$$\begin{aligned}\deg_2(e_i) &\leq \deg_1(e_i) + \sum_{\substack{v:av\in E\\ v\neq b}} (\deg(v) - 1) + \sum_{\substack{v:bv\in E\\ v\neq a}} (\deg(v) - 1)\\ &\leq \deg(a) + \deg(b) - 2 + \left[\sum_{\substack{v:av\in E\\ v\neq b}} \deg(v)\right] - \deg(a) + 1 + \left[\sum_{\substack{v:bv\in E\\ v\neq a}} \deg(v)\right] - \deg(b) + 1\\ &\leq \sum_{\substack{v:av\in E\\ v\neq b}} \deg(v) + \sum_{\substack{v:bv\in E\\ v\neq a}} \deg(v)\end{aligned}$$

The first inequality avoids counting edges twice, in particular if a, b, and v form a triangle. The -1s in the first equality is to avoid counting the edge av in $\deg(v)$, for every $v \in N(a)$, similarly for b. The +1s in the second equality is for not counting edge ab for both a and b in $\deg(a)$ and $\deg(b)$. ◀

When scanning τ_{i-1} to find the rightmost nonneighbour of e_i, we check $\mathcal{O}(\deg_2(e_i))$ vertices, each check takes constant time using arrays A, B, and F. Since the weights are positive, $w(S(e_i)) = w(S(f_j)) + w(e_i) > w(S(f_j))$ if such an f_j exists, and thus $f_j \prec_\tau e_i$. Therefore, inserting e_i into τ_{i-1} to create τ_i will also take $\mathcal{O}(\deg_2(e_i))$ time.

Summing over all vertices in π, of which there are $m = |E|$, we have $\mathcal{O}(m + \sum_{e_i} \deg_2(e_i))$. It remains to bound $\sum_{e_i} \deg_2(e_i)$.

▶ **Claim 8.** $\sum_{e_i} \deg_2(e_i) \leq \mathcal{O}(mn)$.

Proof. By Claim 7, we have:

$$\sum_{e_i} \deg_2(e_i) \leq \sum_{e_i=(a,b)} \left[\sum_{\substack{v:av \in E \\ v \neq b}} \deg(v) + \sum_{\substack{v:bv \in E \\ v \neq a}} \deg(v) \right]$$

For a given vertex v, $\deg(v)$ is used $\deg(v)$ time, one for every edge incident to v, thus

$$\begin{aligned} \sum_{e_i} \deg_2(e_i) &= \sum_{e_i=(a,b)} \left[\sum_{\substack{v:av \in E \\ v \neq b}} \deg(v) + \sum_{\substack{v:bv \in E \\ v \neq a}} \deg(v) \right] \\ &\leq \deg(v_1) \cdot \deg(v_1) + \ldots + \deg(v_n) \cdot \deg(v_n) \\ &\leq \deg(v_1) \cdot \Delta_G + \ldots + \deg(v_n) \cdot \Delta_G \leq 2m \cdot \Delta_G \leq \mathcal{O}(mn) \end{aligned}$$ ◀

Therefore, the total running time is $\mathcal{O}(m+mn) = \mathcal{O}(mn)$. The correctness and robustness of the algorithm follows from Theorem 3 as well as the correctness and robustness of Algorithm 1, which we give in [24]. We conclude with the following theorem:

▶ **Theorem 9.** *Let $G = (V, E, w)$ be an edge weighted cocomparability graph, where $w : E \to \mathbb{R}_{>0}$. A maximum weight induced matching on G can be computed in $\mathcal{O}(mn)$ time.*

5 Conclusions and perspectives

In this paper, we give a general theorem that shows that a number of vertex ordering characterizations are closed under the operation of taking the square of the line graph. Using the $\star$ and $\bullet$ rules, we get that chordal, threshold, interval, split, and cocomparability graphs all have vertex orderings closed under the $L^2(\cdot)$ operation. This gives in our opinion a natural way to approach this closure under $L^2(\cdot)$; and unifies the results on structural graph classes that have relied on geometric intersection models to show such closure. Furthermore, being able to compute vertex orderings directly can be exploited algorithmically, since algorithms on the graph classes covered often rely on their vertex ordering characterizations. We also show structural results and properties on cocomparability graphs that allow us to compute a maximum weighted induced matching on this graph class in $\mathcal{O}(mn)$ time, an improvement over the best $\mathcal{O}(n^4)$ time algorithm for the unweighted case. A natural question however is whether one can use the vertex orderings σ^* of the $L^2(G)$ to compute an induced matching more efficiently for other graph classes, similarly to how we did for cocomparability graphs. We note that the graph classes covered in this work are not necessarily the only ones for which the $\star, \bullet$ rules work, thus it's natural to ask what other graph families have this property. In particular, we illustrate our result on graph families with forbidden patterns on three vertices and therefore raise the question of what can be said about forbidden patterns on four or more vertices, but also if other rules exist that preserve orderings in $L^2(G)$.

Another natural question one can raise is whether computing a maximum cardinality induced matching on cocomparability graphs can be done faster than $\mathcal{O}(mn)$ time, especially since computing a maximum cardinality independent set on cocomparability graphs is done with a simple greedy LexDFS based algorithm [11]. LexDFS and LexBFS are graph searching algorithms that have proven powerful on a number of graph families, cocomparability being one of them. We refer the reader to [32, 11, 9, 30, 10, 29] for more on this topic. Unfortunately, one can show that LexDFS cocomparability orderings are not preserved under the $\star$ and $\bullet$ rules, and thus computing such a solution would require computing a LexDFS ordering on $\sigma^*, \sigma^\bullet$. Such an algorithm exists and runs in linear time [23], but it would be linear in the size of $L^2(G)$, thus not in $\mathcal{O}(m+n)$ time. Similarly, LexBFS cocomparability orderings are not preserved under the $\star$ and $\bullet$ rules. We ask the question whether one can come up with a different rule that preserves LexDFS and/or LexBFS cocomparability orderings on $L^2(G)$ without computing the square edges. Such a technique was successfully used with LexBFS on chordal graph in [3].

Lastly, we raise the question of whether $\sigma^*, \sigma^\bullet$ can lead to efficient algorithms to compute a strong edge colouring for these graph classes. Recall that a strong edge colouring is the partitioning of G into induced matchings, and thus the partitioning of $L^2(G)$ into independent sets. The strong chromatic number of G is the size of a minimum strong edge colouring of G. It is thus easy to see that the strong chromatic number of G is just $\chi(L^2(G))$. Since the graph families we presented are perfect, their chromatic number can be computed in polynomial time. In fact for many graph families, it is done in linear time, and it often relies on the vertex ordering characterization of the graph class. Since a vertex ordering of $L^2(G)$ can be computed in linear time given σ, we ask whether $\sigma^*, \sigma^\bullet$ can be used to compute $\chi(L^2(G))$, without computing the edges of $L^2(G)$.

References

1 Hari Balakrishnan, Christopher L. Barrett, V. S. Anil Kumar, Madhav V. Marathe, and Shripad Thite. The distance-2 matching problem and its relationship to the mac-layer capacity of ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1069–1079, 2004.

2 Vincenzo Bonifaci, Peter Korteweg, Alberto Marchetti-Spaccamela, and Leen Stougie. Minimizing flow time in the wireless gathering problem. *ACM Trans. Algorithms*, 7(3):33:1–33:20, 2011.

3 Andreas Brandstädt and Chính T. Hoàng. Maximum induced matchings for chordal graphs in linear time. *Algorithmica*, 52(4):440–447, 2008.

4 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.

5 Kathie Cameron. Induced matchings. *Discrete Applied Mathematics*, 24(1-3):97–102, 1989.

6 Kathie Cameron. Induced matchings in intersection graphs. *Discrete Mathematics*, 278(1-3):1–9, 2004.

7 Kathie Cameron, R. Sritharan, and Yingwen Tang. Finding a maximum induced matching in weakly chordal graphs. *Discrete Mathematics*, 266(1-3):133–142, 2003.

8 Jou-Ming Chang. Induced matchings in asteroidal triple-free graphs. *Discrete Applied Mathematics*, 132(1-3):67–78, 2003.

9 Pierre Charbit, Michel Habib, Lalla Mouatadid, and Reza Naserasr. Towards A unified view of linear structure on graph classes. *CoRR*, abs/1702.02133, 2017.

10 Derek G. Corneil, Barnaby Dalton, and Michel Habib. Ldfs-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM J. Comput.*, 42(3):792–807, 2013.

11 Derek G. Corneil, Jérémie Dusart, Michel Habib, and Ekkehard Köhler. On the power of graph searching for cocomparability graphs. *SIAM J. Discrete Math.*, 30(1):569–591, 2016.

12 Peter Damaschke. *Forbidden Ordered Subgraphs*. Topics in Combinatorics and Graph Theory: Essays in Honour of Gerhard Ringel. Physica-Verlag HD, 1990.

13 William Duckworth, David Manlove, and Michele Zito. On the approximability of the maximum induced matching problem. *J. Discrete Algorithms*, 3(1):79–91, 2005.

14 Shimon Even, Oded Goldreich, Shlomo Moran, and Po Tong. On the np-completeness of certain network testing problems. *Networks*, 14(1):1–24, 1984.

15 Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics)*, volume 57. North-Holland Publishing Co., 2004.

16 Martin Charles Golumbic and Renu C. Laskar. Irredundancy in circular arc graphs. *Discrete Applied Mathematics*, 44(1-3):79–89, 1993.

17 Martin Charles Golumbic and Moshe Lewenstein. New results on induced matchings. *Discrete Applied Mathematics*, 101(1-3):157–165, 2000.

18 Martin Charles Golumbic, Doron Rotem, and Jorge Urrutia. Comparability graphs and intersection graphs. *Discrete Mathematics*, 43(1):37–46, 1983.

19 Michel Habib and Lalla Mouatadid. Maximum induced matching algorithms via vertex ordering characterizations. *CoRR*, abs/1707.01245, 2017.

20 Pavol Hell, Bojan Mohar, and Arash Rafiey. Ordering without forbidden patterns. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 554–565, 2014.

21 Changhee Joo, Gaurav Sharma, Ness B. Shroff, and Ravi R. Mazumdar. On the complexity of scheduling in wireless networks. *EURASIP J. Wireless Comm. and Networking*, 2010, 2010.

22 Daniel Kobler and Udi Rotics. Finding maximum induced matchings in subclasses of claw-free and p5-free graphs, and in graphs with matching and induced matching of equal maximum size. *Algorithmica*, 37(4):327–346, 2003.

23 Ekkehard Köhler and Lalla Mouatadid. Linear time lexdfs on cocomparability graphs. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, pages 319–330, 2014.

24 Ekkehard Köhler and Lalla Mouatadid. A linear time algorithm to compute a maximum weighted independent set on cocomparability graphs. *Inf. Process. Lett.*, 116(6):391–395, 2016.

25 Dieter Kratsch and Lorna Stewart. Domination on cocomparability graphs. *SIAM J. Discrete Math.*, 6(3):400–417, 1993.

26 Ravi Kumar, Uma Mahadevan, and D. Sivakumar. A graph-theoretic approach to extract storylines from search results. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 216–225, 2004.

27 Vadim V. Lozin. On maximum induced matchings in bipartite graphs. *Inf. Process. Lett.*, 81(1):7–11, 2002.

28 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

29 George B. Mertzios and Derek G. Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *SIAM J. Discrete Math.*, 26(3):940–963, 2012.

30 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *CoRR*, abs/1703.05598, 2017.

31 Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem in planar graphs. In *Frontiers in Algorithmics, First Annual International Workshop, FAW 2007, Lanzhou, China, August 1-3, 2007, Proceedings*, pages 325–336, 2007.

32 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.

33 Larry J. Stockmeyer and Vijay V. Vazirani. Np-completeness of some generalizations of the maximum matching problem. *Inf. Process. Lett.*, 15(1):14–19, 1982.

On-the-Fly Array Initialization in Less Space*

Torben Hagerup[1] **and Frank Kammer**[2]

1 **Institut für Informatik, Universität Augsburg, Augsburg, Germany**
`hagerup@informatik.uni-augsburg.de`

2 **MNI, Technische Hochschule Mittelhessen, Gießen, Germany**
`frank.kammer@mni.thm.de`

Abstract

We show that for all given $n, t, w \in \{1, 2, \ldots\}$ with $n < 2^w$, an array of n entries of w bits each can be represented on a word RAM with a word length of w bits in at most $nw + \lceil n(t/(2w))^t \rceil$ bits of uninitialized memory to support constant-time initialization of the whole array and $O(t)$-time reading and writing of individual array entries. At one end of this tradeoff, we achieve initialization and access (i.e., reading and writing) in constant time with $nw + \lceil n/w^t \rceil$ bits for arbitrary fixed t, to be compared with $nw + \Theta(n)$ bits for the best previous solution, and at the opposite end, still with constant-time initialization, we support $O(\log n)$-time access with just $nw + 1$ bits, which is optimal for arbitrary access times if the initialization executes fewer than n steps.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Data structures, space efficiency, constant-time initialization, arrays

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.44

1 Introduction

Whereas the space used by an algorithm (measured in "memory units" such as words) is usually bounded by its running time, there may be exceptions if the memory offers random access, and it is occasionally useful to employ large arrays of which only a small part will ever be accessed. A case in point are adjacency matrices, which are a convenient representation of graphs if the algorithms to be executed issue adjacency queries (e.g., "does G contain an edge from u to v?") in an irregular pattern that cannot be served efficiently using adjacency lists. Even if one can afford the space needed by an adjacency matrix, it may be prohibitively expensive to clear all those entries in the matrix that do not correspond to edges in the graph. The problem does not occur if the memory cells allocated to hold the adjacency matrix can be assumed to be already initialized to some particular value (that can be taken to signify "no edge"), but in general this is not a realistic assumption. Therefore the problem of simulating an initialized array in an uninitialized memory has been considered since the early days of computing.

Additional motivation for our work comes from the fact that certain modern programming languages such as Java, VHDL and D stipulate that memory be initialized (e.g., cleared to zero) before it is allocated to application programs [8, 12] or have this as the default behavior [2]. The initialization is carried out for security reasons and to ease debugging by making faulty programs more deterministic. If it can be ensured that application programs access memory only through a well-defined interface, one may hope to let the interface provide conceptually cleared memory while avoiding the overhead of clearing the memory physically.

* A fuller version of this paper is available as [11], `https://arxiv.org/abs/1709.10477`.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 44; pp. 44:1–44:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For some $w \in \mathbb{N} = \{1, 2, \ldots\}$, our model of computation is a word RAM [3, 9] with a word length of w bits, where we assume that w is large enough to allow all memory words in use to be addressed. As part of ensuring this, in the context of an array of size n we always assume that $n < 2^w$. The word RAM has constant-time operations for addition, subtraction and multiplication modulo 2^w, division with truncation ($(x, y) \mapsto \lfloor x/y \rfloor$ for $y > 0$), left shift modulo 2^w ($(x, y) \mapsto (x \ll y) \bmod 2^w$, where $x \ll y = x \cdot 2^y$), right shift ($(x, y) \mapsto x \gg y = \lfloor x/2^y \rfloor$), and bitwise Boolean operations (AND, OR and XOR (exclusive or)). We also assume a constant-time operation to load an integer that deviates from $\sqrt{w}$ by at most a constant factor – this enables the proof of Lemma 3. The problem of central concern to us is to realize a clearable word array, defined as follows:

▶ **Definition 1.** A *clearable word array* is a data structure that can be initialized with an integer $n \in \mathbb{N}$ and subsequently maintains an element of $\{0, \ldots, 2^w - 1\}^n$, called its *client sequence* and initially $(0, 0, \ldots, 0)$, under the following operations:

read(ℓ) ($\ell \in \{0, \ldots, n-1\}$): If the client sequence before the call is $(x_0, \ldots, x_{n-1})$, returns x_ℓ without changing the client sequence.

write(ℓ, x) ($\ell \in \{0, \ldots, n-1\}$ and $x \in \{0, \ldots, 2^w - 1\}$): If the client sequence before the call is $(x_0, \ldots, x_{n-1})$, changes the client sequence to be $(x_0, \ldots, x_{\ell-1}, x, x_{\ell+1}, \ldots, x_{n-1})$.

The clearable word array is a special case of the *initializable array* of Navarro [15]. There are two differences. First, the data structure of Navarro is more general in that the initialization, in addition to n, receives a second parameter v that is taken to be the initial value of the array entries, i.e., the initial value of the client sequence is $(v, v, \ldots, v)$ rather than $(0, 0, \ldots, 0)$. As is easy to see and will be discussed in Section 3, however, the more general data structure reduces easily to the more restricted one. Second, Navarro does not specify the nature of the array entries, which is of no relevance to his approach, whereas we fix the array entries to be *words*, i.e., elements of $\{0, \ldots, 2^w - 1\}$. Again, this will turn out to be a restriction of little consequence.

Following the initialization of a clearable word array with an integer n, we call n the *universe size* of the data structure. We shall have occasion to consider restricted clearable word arrays that can be initialized only for certain specific universe sizes. Because the connection between the client sequence of an initializable array and an array used to hold it is often very close, it is easy to confuse the two. We may view the client sequence as an array, but then use the letter 'a' to denote this abstract array (which is initialized) and 'A' to denote the corresponding physical array (which is not initialized).

2 Previous Work

Fredriksson and Kilpeläinen [7] give a detailed overview of the known approaches to array initialization and compare them experimentally. In the discussion of their work, we assume that the task is to realize an initializable array of n entries of $b \le w$ bits each. Define the *redundancy* of a data structure that solves this problem and occupies N bits to be $N - nb$, i.e., the number of bits used beyond the minimum of nb bits needed even without the requirement of initializability.

A number of the methods described by Fredriksson and Kilpeläinen can be viewed as special cases of a general *trie method*. Ignoring rounding issues, the trie method is parameterized by an integer $h \in \mathbb{N}$ and a *degree sequence* $(d_1, d_2, \ldots, d_h)$ of h positive integers with $\prod_{i=1}^{h} d_i = nb$. It uses a tree T of height h in which all nodes of height i have d_i children, for $i = 1, \ldots, h$. Each node in T has an associated bit, the bits of each maximal group of siblings are stored compactly, w bits to a word, and the nb bits at the leaves are identified with the nb bits of the abstract array a.

Let *processing* an inner node u in T be the following: If the bit associated with u has the value 0 (informally, u has been initialized, but its children have not), initialize the bits of all children of u, to 0 if the children are inner nodes and to the prescribed initial value v – within groups of b siblings in the obvious manner – if they are leaves. If u has d children, this can be done in $O(\lceil d/w \rceil)$ time. Finally set the bit associated with u to 1. If the value of that bit is 1 already prior to the processing of u, the processing of u terminates immediately after discovering this fact.

To initialize T, set the bit at its root to 0. In addition, it is permissible, as part of the initialization, to process the inner nodes in an upper part of T in a top-down fashion, i.e., so that no nonroot node is processed before its parent. We will say that such nodes are *preprocessed*. To read the ℓth entry of a, descend in T towards the ℓth group of b leaves. If an inner node is encountered whose associated bit has the value 0, return v. If not, return the value found in the ℓth group of b leaves. To write the ℓth entry of a, descend in the same manner towards the ℓth group of b leaves, process every inner node encountered on the way, and finally store the appropriate value in the bits of the ℓth group of b leaves. The total number of bits used by the data structure is the number of nodes in T that are not preprocessed, the initialization takes constant time plus time proportional to the sum of $\lceil d/w \rceil$ over all degrees d of preprocessed nodes, the worst-case time of *read* is $\Theta(h)$, and the worst-case time of *write* is the maximum over all leaves v in T of $\Theta(h + \sum_i \lceil d_i/w \rceil)$, where the sum ranges over those values of $i \in \{1, \ldots, h\}$ for which the ancestor of v of height i is not preprocessed.

Fredriksson and Kilpeläinen consider the following special cases of the trie method: Degree sequence (nb), preprocess the root (Plain); degree sequence (b, n), preprocess the root (Simple); degree sequence $(b, w, w, \ldots, w)$ (Hierarchic); degree sequence $(b, n/w, w)$ (Simple-H); and degree sequence $(b, w, n/w)$, preprocess the root (SHV). The redundancy is 0 for Plain and close to n (i.e., the number of nodes in T of height 1) for the other methods. The initialization time is $\Theta(1 + nb/w)$ for Plain, $\Theta(1 + n/w)$ for Simple, $\Theta(1 + n/w^2)$ for SHV and $\Theta(1)$ for the other methods. The worst-case time for *read* is $\Theta(1 + \log_w n)$ for Hierarchic and $\Theta(1)$ for the other methods. The worst-case time for *write*, finally, is $\Theta(1 + \log_w n)$ for Hierarchic, $\Theta(1 + n/w^2)$ for Simple-H and $\Theta(1)$ for the other methods.

None of the methods discussed above combines constant initialization time with constant access time, and it is easy to see that this is true of every instance of the trie method. Constant time for every operation is achieved by a folklore method that goes back at least to the early 1970s (see [1, Exercise 2.12]). The folklore method uses a physical array A with the index set $\{0, \ldots, n-1\}$ and assigns the codes $0, 1, \ldots$ to the indices of the abstract array a in the order in which the indices are first used in calls of *write*, x_ℓ is stored in $A[f(\ell)]$, where $f(\ell)$ is the code of ℓ, two tables are used to keep track of the encoding function f and its inverse f^{-1}, and finally the data structure remembers the number k of codes assigned. To access x_ℓ, first $f(\ell)$ is looked up in the table of f. Because the table is not initialized, the purported code j may not be correct, but j is the code of ℓ exactly if $0 \le j < k$ and the entry of j in the table of f^{-1} is ℓ. If not, the default initial value v is returned in the case of a read operation, and the next available code is assigned to ℓ in the case of a write operation. The remainder of the access is simply a reading or writing of $A[f(\ell)]$. The structure is initialized by setting k to 0. In addition to the space needed to hold the actual data in A, it needs space for the tables of f and f^{-1} and the counter k, so that its redundancy is $2n\lceil \log_2 n \rceil + \lceil \log_2(n+1) \rceil$.

A family of methods due to Navarro [15] combines the Hierarchic method above with the folklore method. The idea is, starting from Hierarchic, to replace the nodes of height $\ge h+2$, for some $h \ge 0$, by an instance of the folklore data structure. This achieves the same effect as processing the nodes that were removed and obviates the need to descend through these nodes during an access. The initialization time is constant, the worst-case access time is $\Theta(h+1)$, and the redundancy is approximately $3n$ for $h = 0$ and approximately n for $h \geq 1$.

3 Our Contribution

We give an upper-bound tradeoff that spans the entire range from minimal time to minimal space. Our main result is the following:

▶ **Theorem 2.** *There is a clearable word array that, for all given $n, t \in \mathbb{N}$, can be initialized for universe size n in constant time and subsequently occupies at most $nw + \lceil n(t/(2w))^t \rceil$ bits and supports read and write in $O(t)$ time.*

If w and hence (by assumption) n are bounded by constants, it is trivial to realize a clearable word array with constant initialization and access times and zero redundancy (initialize the array explicitly, i.e., use the Plain method of Fredriksson and Kilpeläinen). Given a constant $t \in \mathbb{N}$, we can therefore assume without loss of generality that $w \geq t^2$. Then $(t/w)^2 \leq 1/w$ and hence $(2t/(2w))^{2t} \leq 1/w^t$. Theorem 2 (used with t doubled) thus implies that for all constant $t \in \mathbb{N}$, there is a clearable word array that can be initialized in constant time, executes accesses in constant time and has redundancy $\lceil n/w^t \rceil$. The best previous constant-time solution, due to Navarro [15] and discussed above, has redundancy $n + o(n)$.

At the other end of the time-space tradeoff, for $t = \lceil \log_2 n \rceil$, the redundancy of Theorem 2 is 1, i.e., the constant-time initialization costs only a single bit and accesses are still supported in logarithmic time. If an initialization time of $\Theta(n)$ is acceptable, a clearable word array with constant-time access can obviously be realized with zero redundancy – this is again the Plain method of Fredriksson and Kilpeläinen. On the other hand, the redundancy cannot be reduced below our bound of 1 for any access times unless the initialization writes to at least n words, which needs at least n steps. To see this, assume that a clearable word array with universe size n is represented in N bits for some $N \in \mathbb{N}$. Because the client sequence can be in any one of 2^{nw} states, any two of which can be distinguished through *read* operations, whereas its representation can be in only 2^N states, we must have $N \geq nw$, irrespectively of all operation times. Moreover, if $N = nw$, every state of the client sequence is represented by exactly one bit pattern of its representation. Since the client sequence is in a well-defined state immediately after the initialization, this is impossible unless each of the nw bits of its representation is forced to one specific value during the initialization, i.e., unless the initialization writes to at least n words.

Note that it is a responsibility of the user of a clearable word array initialized for universe size n to ensure that $\ell < n$ in all calls of the form *read*(ℓ) or *write*(ℓ, x) issued to the data structure. Whereas the data structure can easily check the conditions $\ell \geq 0$ and $0 \leq x < 2^w$, when operated close to its minimum space it cannot afford to store the integer n. Thus illegal calls of its operations may go undetected and may lead to attempted accesses to memory words outside of the area assigned to the data structure.

Our result can be seen as a second application of the *light-path technique*, which was introduced (but not named) in [10] and used there to construct space-efficient nonsystematic choice dictionaries. From a technical perspective, the situation is simpler here, as there is no need to store data in a particular *compact representation* and to provide conversion to and from the compact representation. This gives us an opportunity to illustrate the light-path technique in a purer setting. At a more abstract level, the fundamental idea is to upset the structure of a simple table slightly in order to accommodate additional information in the table. Whereas this principle has been used before [4, 5, 14], curiously, it has not so far been employed in the setting of initializable arrays even though it seems particularly natural there. It may be noted that the c-color choice dictionaries of [10] could be used directly as initializable arrays, but efficiently so only for arrays whose elements are drawn from a very small range $\{0,\ldots,2^b-1\}$. This is because each element of that range would be considered a separate color, i.e., we would have $c=2^b$.

Given the clearable word array of Theorem 2, it is easy to derive a more general data structure that, for some integer b with $1\le b\le w$, maintains a client sequence in $\{0,\ldots,2^b-1\}^n$, initially $(0,0,\ldots,0)$, under reading and writing of individual elements of the sequence. Simply pack the n elements of the client sequence tightly in $\lceil nb/w\rceil$ words of w bits each, initialize the used part of the last word to 0, maintain the other words in a clearable word array, inspect a b-bit element of the client sequence by reading the at most two words over which the b bits spread, picking out the relevant pieces of the words and concatenating the pieces, and update a b-bit element of the client sequence correspondingly by splitting the new value into at most two pieces and storing each piece appropriately in a word without disturbing the rest of the word. The execution times are within a constant factor of those of the clearable word array, and the number of bits needed is at most $nb+\lceil n(t/(2w))^t\rceil$.

We can also easily derive a data structure more general than that of Theorem 2 in that the client sequence is initialized to $(g(0),\ldots,g(n-1))$, where $g:\{0,\ldots,n-1\}\to\{0,\ldots,2^w-1\}$ is some function, rather than to $(0,0,\ldots,0)$. The simple idea is to swap the representations of the "internal" and "external" initial values. Both $\mathit{read}(\ell)$ and $\mathit{write}(\ell,x)$ then begin by evaluating $g(\ell)$. If reading the value associated with ℓ in a normal clearable word array yields the value 0, $\mathit{read}(\ell)$ returns $g(\ell)$. If the value read is $g(\ell)$, $\mathit{read}(\ell)$ returns 0, and every other value read is returned as it is. Similarly, if $x=g(\ell)$, $\mathit{write}(\ell,x)$ actually writes the value 0 to the normal clearable word array, $x=0$ causes the value $g(\ell)$ to be written, and every other value of x is written as it is. The initialization and access times are those of Theorem 2 plus whatever time is needed to initialize g and to evaluate it on one argument, respectively, and the space requirements are those of Theorem 2 plus those of g. It is easy to see that the generalizations described in this and the previous paragraph can be combined.

Very recently, giving a clever twist to the folklore method, Katoh and Goto [13] devised a clearable word array that executes every operation in constant time but, when the universe size is n, uses just $nw+1$ bits.

4 The Construction

In this section we prove Theorem 2. At a very low and technical level, we need the following staple of word-RAM computing.

▶ **Lemma 3** ([6, 10]). *Given a nonzero integer $\sum_{i=0}^{w-1}2^i b_i$, where $b_i\in\{0,1\}$ for $i=0,\ldots,w-1$, constant time suffices to compute* $\max I$ *and* $\min I$*, where $I=\{i\mid 0\le i\le w-1$ and $b_i=1\}$.*

Let a *colored tree* be an ordered outtree, each of whose leaves is either *white* or *black*. Given a colored tree T, we extend the colors at the leaves of T to its inner nodes as follows: If the leaf descendants of an inner node u all have the same color (white or black), then u has that same color. If u has both a white and a black leaf descendant, u is *gray*. Clearly every ancestor of a node v has the same color as v or is gray. In particular, every ancestor of a gray node is gray. Define the *navigation vector* of an inner node to be the sequence of the colors of its children in the order from left to right.

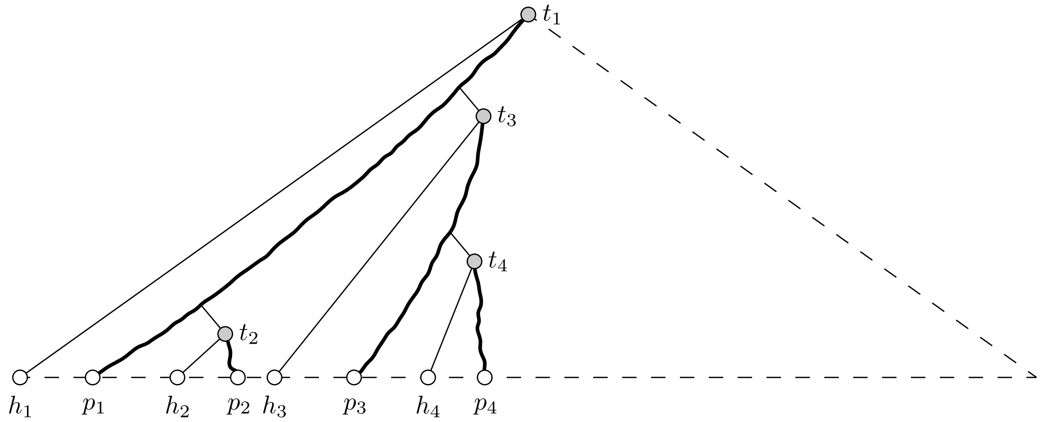


Figure 1 Example light paths (drawn thicker). Top nodes, historians and proxies are labeled "t", "h" and "p", respectively, and a subscript identifies the associated light path.

Recall that the *left spine* of a rooted ordered tree T is the maximal path in T that starts at the root of T and, whenever it contains an inner node u, also contains the leftmost child of u. Define the *preferred child* of a white or gray inner node in a colored tree T to be its leftmost gray child if it has at least one gray child, and its leftmost white child otherwise. Call an edge in T *light* if it leads from a gray inner node to its preferred child or lies on the left spine of a subtree of T whose root is white and has a gray parent of which it is the preferred child. In other words, every gray inner node picks the edge to its preferred child to be light, whereas a white inner node does so only if "prompted" by its parent. The light edges induce a collection of node-disjoint paths called *light paths*, each of which ends at a leaf in T. When P is a light path that starts at a (gray) node u and ends at a (white) leaf v, we call u the *top node*, v the *proxy* and the leftmost leaf descendant of u (that may coincide with v) the *historian* of P and of every node on P. These concepts are illustrated in Fig. 1. A gray node that is not the root of T is a top node exactly if it is not the preferred child of its parent, i.e., if it has at least one gray left sibling. No proper ancestor of a top node u can have a descendant of u as its leftmost leaf descendant, so a leaf is the historian of at most one light path. If h is the historian of a light path P, the top node and the proxy of P are also said to be the top node and the proxy, respectively, of h. A leaf ℓ cannot be the historian of one light path and the proxy of another, since otherwise the two corresponding top nodes would both be ancestors of ℓ and the path between them would contain only gray nodes and be part of a light path, a contradiction. A similar argument shows that in the left-to-right order of the leaves of T, no historian or proxy lies strictly between a historian and its proxy. Define the *history* of a light path that contains the nodes $u_1, \ldots, u_k$, in that order, to be the sequence $(q_1, \ldots, q_{k-1})$, where q_i is the navigation vector of u_i, for $i = 1, \ldots, k-1$ (u_k, as a leaf, has no navigation vector).

The following lemma describes the work-horse of our data structure.

▶ **Lemma 4.** *Let d and t be given positive integers with $2dt \le w$ such that d is a power of 2. Then there is a clearable word array that can be initialized for universe size $n = d^t$ in constant time and subsequently occupies $nw + 2$ bits and, if given access to the parameters d and t, supports read and write in $O(t)$ time.*

Proof. Without loss of generality assume that $d \ge 2$. We use a conceptual colored tree T that is a complete d-ary tree of height t and identify the n leaves of T, in the order from left to right, with the integers $0, \ldots, n-1$. Let r be the root of T and, for each node u in

T, let T_u be the maximal subtree of T rooted at u. We represent a node u of height j in T through the pair (j, k), where k is the number of nodes in T of the same height as u and strictly to its left (in other words, the nodes on each level in T are numbered consecutively in the order from left to right, starting at 0). Then navigating in T is easy: If u is not the root r, its parent is (represented through) $(j+1, \lfloor k/d \rfloor)$, if u is not a leaf, its children are $(j-1, kd), \ldots, (j-1, kd+(d-1))$, u's leftmost leaf descendant is $(0, kd^j)$ (identified with the integer kd^j), and if u is not a leaf and ℓ is a leaf descendant of u, then *viachild*(u, ℓ), the child of u that is an ancestor of ℓ, is $(j-1, \lfloor \ell/d^{j-1} \rfloor)$. The assumption that d is a power of 2 ensures that we can compute the necessary powers of d in constant time by means of multiplication and left shift. This requires the availability of $\log_2 d$, which can be computed from d in constant time according to Lemma 3.

The actual data is stored in a word array A with index set $\{0, \ldots, n-1\}$ and in two additional *root bits*. The three colors white, gray and black are encoded in two bits, the navigation vector of an inner node in T is represented by the $2d$-bit concatenation of the representations of its d (color) elements, and the history of a k-node light path is represented by the $2d(k-1)$-bit concatenation of the representations of its $k-1$ (navigation-vector) elements. The relation $2dt \le w$ ensures that every history fits in a w-bit word. Assume that a history of fewer than w bits is "right-justified" in the word so that the position in the word of the navigation vector of a node depends only on the height of the node.

With the aid of an algorithm of Lemma 3, the preferred child of a given white or gray inner node u in T can be computed in constant time from the navigation vector of u or a history that contains that navigation vector. This may need a couple of bit masks (informally, ones that correspond to all nodes having the same color) that can easily be obtained via multiplication with the integer $1_{dt,2} = (2^{2dt} - 1)/3$, whose $(2dt)$-bit binary representation is $0101 \cdots 0101$. Because 2^{2dt} may not be representable in a w-bit word (namely if $2dt = w$), the computation of $1_{dt,2}$ needs a little care, but is still easy to do in constant time.

The client sequence $(x_0, \ldots, x_{n-1})$ is represented in $A[0], \ldots, A[n-1]$ and the two root bits according to the following storage invariants: First, the two root bits indicate the color of the root r of T. Second, for $\ell = 0, \ldots, n-1$,

- if ℓ is a historian, $A[\ell]$ stores the history of the proxy of ℓ (hence the term "historian"),
- if ℓ is black and not a historian, $A[\ell]$ stores x_ℓ,
- if ℓ is a proxy whose historian h is black, $A[\ell]$ stores x_h (as a "proxy" for h), and
- if ℓ is white and neither a historian nor a proxy whose historian is black, the value of $A[\ell]$ may be arbitrary.

Note that because every proxy is white, for each $\ell \in \{0, \ldots, n-1\}$ exactly one of the four cases above applies. In particular, although a proxy may coincide with its historian, this is not the case if the historian is black. The data structure is initialized by coloring r white (i.e., by setting the root bits accordingly).

In terms of the abstract array a, the leaf colors white and black signify "not yet written to, and therefore still containing the initial value 0" and "written to at least once", respectively. For the actual array A, this translates approximately into white and black meaning "not initialized" and "initialized to a meaningful value", respectively.

The data structure does not explicitly store the color of any node except r. Instead node colors must be deduced from histories. It turns out that the colors of all nodes other than r are implied by the histories of the light paths. A white leaf ℓ offers potential for storing a history (namely in its associated word $A[\ell]$), but we cannot know in advance where to find a white leaf. This motivates the introduction of historians and proxies. We actually need the history of a light path P when, during a descent in T from r to a leaf, we reach the top node of P. The historian of P provides a fixed place (namely at the leftmost leaf descendant) at which to look for the history, but if the historian is black, then its own data must be accommodated somewhere else – this is the role of the (white) proxy. How this works is perhaps best illustrated by the following detailed description of the realization of the operation *read*, which basically carries out a descent in T. The call *leftmostleaf*(u) is assumed to return (the integer identified with) the leftmost leaf descendant of the node u.

```
read(ℓ):
  u := r; (∗ start at the root ∗)
  while u is gray do
    if u is a top node then (∗ switch to a new history ∗)
      h := leftmostleaf(u); (∗ u's historian ∗)
      H := A[h]; (∗ u's history ∗)
    u := viachild(u, ℓ); (∗ continue towards ℓ ∗)
  if u is white then return 0; (∗ the initial value ∗)
  (∗ now ℓ is black ∗)
  if u = r or ℓ ≠ h then return A[ℓ]; (∗ ℓ is neither a historian nor a proxy ∗)
  (∗ now ℓ is a black historian ∗)
  return A[p], where p is the leaf at the end of the light path that contains u's parent;
```

The procedure discovers a white ancestor of ℓ and returns 0, determines that ℓ is black and not a historian and returns $A[\ell]$, or identifies ℓ as a black historian and returns $A[p]$, where p is the proxy of ℓ. In all cases, the return value is correct.

Whenever the color of a node u is queried, either $u = r$, in which case the color of u is given by the root bits, or the color of u can be deduced in constant time from the history stored in H, one of whose elements is the navigation vector of the parent of u. Similarly, if $u \neq r$, we can decide in constant time whether u is a top node by looking at the navigation vector of its parent. The light path that contains u's parent can be followed in constant time per node, again by inspection of H. Thus *read* can be executed in $O(t)$ time.

To execute *write*(ℓ, x), we carry out two phases. The purpose of the first phase is to take the data structure to a legal state in which ℓ is black and all values of the client sequence $(x_0, \ldots, x_{n-1})$ except possibly x_ℓ are correct, i.e., unchanged. The second phase concludes the writing by setting x_ℓ to x. In the description of the two phases, we leave to the reader details such as how to determine the color of a given node; in all cases, one can proceed similarly as in the implementation of *read*.

The first phase begins by following the path P in T from r to ℓ until encountering a node that is not gray. This can be done similarly as in the implementation of *read*: Each node visited is tested for being a top node, and at each top node a new history is fetched and subsequently used. This computation, in particular, can determine the color of ℓ. If ℓ is already black, the first phase terminates without modifying the data structure. Assume in the remaining discussion of the first phase that ℓ is white and consider the consequences of an *update* that changes the color of ℓ from white to black. We will use the terms "old" and "new" to describe the situation before and after the update, respectively.

Because the color of an inner node in T is a function of the colors of its children, only nodes on P can change their color as a result of the update. The first phase proceeds to find the first node v on P (i.e., the node on P of minimal depth) that changes its color. The following observations show that this can be done in a single traversal of P and characterizes the possible scenarios in a useful way. If some proper ancestor of ℓ is white (before the update), all proper ancestors of the first white node $\widetilde{v}$ on P are gray both before and after

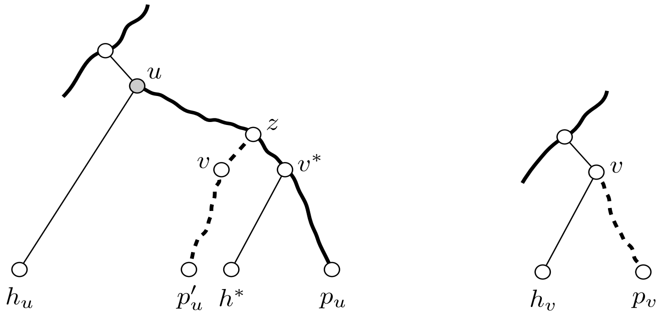


Figure 2 Left: The situation of Case 1 and, after a swap of the labels “p_u” and “p'_u”, also of Case 2. Right: The situation of Cases 4 and 5.

the update, and all descendants of $\widetilde{v}$ on P other than ℓ are white before and gray after the update. Thus $v = \widetilde{v}$. In the opposite case, namely if all proper ancestors of ℓ are gray, let $\overline{v}$ be the last node on P that has a white or gray sibling if there is at least one such node, and take $\overline{v} = r$ otherwise. It is easy to see that all proper ancestors of $\overline{v}$ are gray both before and after the update and, by backwards induction on P, that all descendants of $\overline{v}$, including $\overline{v}$ itself, are black after the update. In this case, therefore, $v = \overline{v}$.

As can be seen from the observations above, no descendant of v has more than one gray child before or after the update under consideration. Therefore the only node in T_v that can be a top node before or after the update is v itself, the only node in T_v that can be a historian before or after the update is the leftmost leaf descendant h_v of v, and before as well as after the update at most one node in T_v is a proxy. Moreover, at most one node in T other than v can become or stop being a top node as a result of the update, and this node, if it exists, must be the leftmost gray sibling of v and to the right of v.

If $v = r$, change the root bits to reflect the new color of the root. Otherwise compute u as the top node of the light path that contains the (gray) parent of v, let h_u be the historian of u (before and after the update) and let p_u and p'_u be the proxies of u before and after the update, respectively, which can be found by following the old and new light paths that start at u. Store the new history of p'_u in $A[h_u]$. In particular, this registers the new color of v. To compute the history, it suffices to record the new navigation vectors encountered on the path in T from u to p'_u. Now consider five cases that together cover all possible situations and do not overlap. Even though every color change is irreversible, Cases 1 and 2 show some aspects of being reverses of each other, and so do Cases 4 and 5. These four cases are illustrated in Fig. 2.

Case 1*:* v has a parent z and is the preferred child of z after the update. Thus v changes its color from white to gray without becoming a top node. If h_u is black before the update, then execute $A[p'_u] := A[p_u]$, which moves x_{h_u} from the old to the new proxy of h_u. This overwrites no relevant information, as p'_u is white and neither a historian nor a proxy before the update unless p'_u coincides with h_u or p_u, in which case the assignment is not carried out or has no effect. Let v^* be the preferred child of z before the update and let h^* be the leftmost leaf descendant of v^*. If v^* is white before the update (this includes the case $v^* = v$), nothing more needs to be done. If v^* is gray (before and after the update), it is a right sibling of v, and it becomes a new top node whose historian h^* and proxy p_u must have their associated information updated accordingly. To this end first execute $A[p_u] := A[h^*]$ and subsequently store in $A[h^*]$ the history of the new light path that starts at v^* and ends at p_u. If $p_u = h^*$, the two assignments write to the same word, but then any relevant information present in $A[h^*]$ before the update was already copied to $A[p'_u]$.

Case 2*:* v has a parent z and is the preferred child of z before, but not after the update. After the update, v is black and no descendant of v is a historian or a proxy, except that h_v may coincide with h_u. Let v^* be the preferred child of z after the update and let h^* be the leftmost leaf descendant of v^*. If v^* is gray, it is a right sibling of v and a top node with historian h^* and proxy p'_u before the update, whereas after the update p'_u is the proxy of u and h^* is neither a historian nor a proxy unless $h^* = p'_u$. If h^* is black, then execute $A[h^*] := A[p'_u]$, which moves x_{h^*} to the correct place and overwrites a history that is no longer useful. Finally, independently of the color of v^* and as in Case 1, if h_u is black, then execute $A[p'_u] := A[p_u]$.

In the remaining cases 3–5 v is a preferred child neither before nor after the update, so there are no changes to light paths outside of T_v (i.e., the set of light edges outside of T_v remains the same). In particular, $p'_u = p_u$. Moreover, v is not a leftmost child.

Case 3*:* v is a leaf with at least one white left sibling. There are no changes to light paths, so nothing needs to be done.

Case 4*:* v is a top node after the update. Before the update, v is white, so no descendant of v is a historian or a proxy at that time (informally, no information is stored below v). Compute the proxy p_v of v after the update and store the new history of p_v in $A[h_v]$. This involves following the new light path that starts at v and recording the new navigation vectors encountered on the way.

Case 5*:* v is a top node before the update. Because v is black after the update, no descendant of v is a historian or a proxy at that time. Before the update, since ℓ is the only white descendant of v, it is its proxy. If h_v is black (i.e., if $h_v \neq \ell$), then copy the value of $A[\ell]$, namely x_{h_v}, to $A[h_v]$. This overwrites an old history that is no longer useful.

The second phase of the execution of *write*(ℓ, x) simulates the execution of *read*(ℓ) until the point when the routine is ready to return as its answer the value of $A[i]$ for some i (that is either ℓ or the proxy of ℓ). Instead of returning $A[i]$, it finishes by storing x in $A[i]$. Since i is not a historian, it is easy to see that a subsequent call of *read*(ℓ) will return x and that the update of $A[i]$ leaves the data structure in a legal state and does not change the value of any elements of the client sequence $(x_0, \ldots, x_{n-1})$ except x_ℓ. ◀

The next lemma and its proof show how to handle the case of an "incomplete tree" elegantly and, following the initialization, without any overhead to test for special cases.

▶ **Lemma 5.** *There is a clearable word array that, for all given $n, d, t \in \mathbb{N}$ with $2dt \leq w$ and $n \leq d^t$ such that d is a power of 2, can be initialized for universe size n in constant time and subsequently occupies $nw + 2$ bits and, if given access to d and t, supports read and write in $O(t)$ time.*

Proof. We use the construction of the previous proof for universe size d^t, but provide for its storage only a word array A with index set $\{0, \ldots, n-1\}$ in addition to two root bits. If $n = d^t$, nothing more needs to be said. If $n < d^t$, before executing any true *write* operation, we change the color of the root from white to gray (of course, by modifying the root bits) and store in $A[0]$ a history that corresponds to the leaves $0, \ldots, n-1$ being white and $n, \ldots, d^t - 1$ being black. Provided that only legal accesses are subsequently attempted, this prevents the data structure from ever choosing a proxy larger than $n-1$, and it will process the operations correctly without ever attempting to access one of the nonexisting array elements $A[n], \ldots, A[d^t - 1]$.

The computational steps just described are conceptually part of the initialization of the data structure, but the computation of the history to be stored in $A[0]$ may take more than constant time. In order to guarantee a constant initialization time, we postpone the steps and execute them as an initial part of the first and only execution of a *write* operation that begins with a white root, until which point we remember n in $A[0]$. Since the steps are easily carried out in $O(t)$ time, the bound of $O(t)$ for the execution time of *write* remains valid. ◀

We now take the step to values of n larger than d^t.

▶ **Lemma 6.** *There is a clearable word array that, for all given $n, t \in \mathbb{N}$, can be initialized for universe size n in constant time and subsequently occupies at most $nw + \lceil n(t/(2w))^t \rceil$ bits and, if given access to n and t, supports read and write in $O(t)$ time.*

Proof. When $c \in \mathbb{N}$ is an arbitrary constant, we can assume without loss of generality that n is a multiple of c. This is because we can initialize up to $c - 1$ "left-over" words in constant time. Moreover, a word RAM with a word length of w bits can simulate one with a word length of cw bits with constant slowdown, i.e., every instruction can be simulated in constant time. By these observations, we can essentially pretend to be working on a word RAM with a word length of cw bits (of course, the values communicated to and from a user of the data structure are still w-bit quantities). In particular, we view A as consisting of n/c *large words* of cw bits each, and the condition $2dt \leq w$ of Lemma 5 can be relaxed to $2dt \leq cw$. We use this with $c = 16$, for which choice the condition becomes $d \leq 8w/t$.

Assume that $t \leq w$. This entails no loss of generality because reducing larger values of t to w does not increase the space bound of the lemma (recall that $w \geq \lceil \log_2 n \rceil$). Compute d as the largest power of 2 no larger than $8w/t$ and note that $d \geq 4w/t \geq 2$. Dividing the universe $\{0, \ldots, n/c - 1\}$ into *ranges* of d^t consecutive elements each, except that the last range may be smaller, we store each subsequence of the client sequence corresponding to a range in an instance of the data structure of Lemma 5, called a *tree*, except that the root bits are handled slightly differently. Altogether we have $N = \lceil n/(cd^t) \rceil \leq \lceil n(t/(2w))^t \rceil$ trees.

If $N = 1$, i.e., if there is only a single tree, we use a single root bit to distinguish between black and nonblack (i.e., white or gray). In order to indicate a white root, in addition to initializing the root bit to the value that denotes a nonblack color, we store in $A[0]$ a value that cannot be the history of a gray root, such as one in which all colors in the navigation vector of the root are white. The total redundancy is $1 = N \leq \lceil n(t/(2w))^t \rceil$.

If $N > 1$, we solve the problem of initializing the N trees differently. Each tree has two root bits, and we must set these to indicate a white root. Assume, for convenience, that the root color white is represented through two bits with a value of zero. Then the task is to clear the $2N$ root bits, i.e., to set them to zero. Pack the $2N$ root bits tightly in $M = \lfloor 2N/w \rfloor$ fully occupied words and at most one partially occupied word. If there is an only partially occupied word, clear it explicitly. As for the M fully occupied words, maintain these, if $M \geq 1$, in a clearable word array implemented with the folklore method discussed near the end of Section 2. In addition to the M words, this needs space for two tables with altogether $2M$ entries and one counter that takes values in $\{0, \ldots, M\}$. Each table entry fits in a w-bit word, and except in the trivial case $w = 1$, the counter can be stored in N bits, so the redundancy is at most $3Mw + N \leq 7N$. Since $N > 1$, we even have $8N \leq 16(n/c)(t/(4w))^t = n(t/(4w))^t \leq \lceil n(t/(2w))^t \rceil$. This slightly stronger bound is irrelevant here, but useful in a proof of Theorem 2. ◀

In order to derive Theorem 2 from Lemma 6 and its proof, we show in [11] how to "hide" the parameters n and t in the data structure essentially without additional space or how to make do without them.

It is interesting to note that we can add an additional operation to our clearable word array, namely an iteration that enumerates all first arguments of past *write* operations (informally, the positions to which writing took place). For this we would iterate over the codes handed out by the folklore method and the associated trees, which is easy, enumerate all leaves of each tree whose root is black, and for each tree whose root is gray carry out a depth-first search (say) of its gray nodes and enumerate all leaf descendants of their black children. The time needed is proportional to the number k of leaves enumerated plus the total number of gray nodes, a quantity that is clearly bounded by $(t+1)k$ and never larger than $2n$. The iteration must be called with an argument that indicates n.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

2 Andrei Alexandrescu. *The D Programming Language.* Addison-Wesley, 2010.

3 D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. Syst. Sci.*, 18(2):155–193, 1979. doi:10.1016/0022-0000(79)90045-X.

4 Amos Fiat, J. Ian Munro, Moni Naor, Alejandro A. Schäffer, Jeanette P. Schmidt, and Alan Siegel. An implicit data structure for searching a multikey table in logarithmic time. *J. Comput. Syst. Sci.*, 43(3):406–424, 1991. doi:10.1016/0022-0000(91)90022-W.

5 Gianni Franceschini and Roberto Grossi. No sorting? Better searching! *ACM Trans. Algorithms*, 4(1):2:1–2:13, 2008. doi:10.1145/1328911.1328913.

6 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

7 Kimmo Fredriksson and Pekka Kilpeläinen. Practically efficient array initialization. *J. Softw. Pract. Exper.*, 46(4):435–467, 2016. doi:10.1002/spe.2314.

8 James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification, Java SE 8 Edition.* Oracle America, 2015.

9 Torben Hagerup. Sorting and searching on the word RAM. In *Proc. 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998)*, volume 1373 of *LNCS*, pages 366–398. Springer, 1998. doi:10.1007/BFb0028575.

10 Torben Hagerup and Frank Kammer. Succinct choice dictionaries. *Computing Research Repository (CoRR)*, arXiv:1604.06058 [cs.DS], 2016. arXiv:1604.06058.

11 Torben Hagerup and Frank Kammer. On-the-fly array initialization in less space. *Computing Research Repository (CoRR)*, arXiv:1709.10477 [cs.DS], 2017. arXiv:1709.10477.

12 IEC/IEEE International Standard; Behavioural languages — Part 1–1: VHDL Language Reference Manual. IEC 61691–1–1:2011(E) IEEE Std 1076-2008, 2011. doi:10.1109/IEEESTD.2011.5967868.

13 Takashi Katoh and Keisuke Goto. In-place initializable arrays. *Computing Research Repository (CoRR)*, arXiv:1709.08900 [cs.DS], 2017. arXiv:1709.08900.

14 J. Ian Munro. An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. *J. Comput. Syst. Sci.*, 33(1):66–74, 1986. doi:10.1016/0022-0000(86)90043-7.

15 Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52:1–52:47, 2014. doi:10.1145/2535933.

On Directed Covering and Domination Problems

Tesshu Hanaka[1], Naomi Nishimura[*2], and Hirotaka Ono[†3]

1 **Department of Economic Engineering, Kyushu University, Fukuoka, Japan**
3EC15004S@s.kyushu-u.ac.jp

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

3 **Department of Mathematical Informatics, Nagoya University, Nagoya, Japan**
ono@i.nagoya-u.ac.jp

Abstract

In this paper, we study covering and domination problems on directed graphs. Although undirected Vertex Cover and Edge Dominating Set are well-studied classical graph problems, the directed versions have not been studied much due to the lack of clear definitions.

We give natural definitions for Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set as directed generations of Vertex Cover and Edge Dominating Set. For these problems, we show that (1) Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set are NP-complete on planar directed acyclic graphs except when $r = 1$ or $(p, q) = (0, 0)$, (2) if $r \geq 2$, Directed r-In (Out) Vertex Cover is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs, (3) if either p or q is greater than 1, Directed (p, q)-Edge Dominating Set is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs, (4) all problems can be solved in polynomial time on trees, and (5) Directed $(0, 1), (1, 0), (1, 1)$-Edge Dominating Set are fixed-parameter tractable in general graphs.

The first result implies that (directed) r-Dominating Set on directed line graphs is NP-complete even if $r = 1$.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases directed graph, vertex cover, dominating set, edge dominating set, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.45

1 Introduction

Covering and domination problems are well-studied problems in theory and in applications of graph algorithms, for example, Vertex Cover [16], Dominating Set [16] and Edge Dominating Set [24]. However, almost all of these problems are studied on undirected graphs. In particular, Vertex Cover and Edge Dominating Set on directed graphs have not been studied although there are some results on directed Dominating Set [11, 7, 21, 15]. This seems surprising, but maybe one reason might be that it is difficult to expand the definition naturally to directed graphs due to the unclear relationship between "direction" and "domination".

In this paper, we study directed versions of Vertex Cover and Edge Dominating Set. First, we give formal definitions of directed Vertex Cover and directed Edge

[*] This work is supported by the Natural Sciences and Engineering Research Council of Canada.
[†] This work is partially supported by KAKENHI, no. 26241031, 26540005, 17H01698 and 17K19960.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 45; pp. 45:1–45:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Dominating Set. In the definitions, we consider several scenarios that reflect how the selected set influences edges via directed edges. It should be noted that the definition follows from r-Dominating Set [8, 12, 21]. These definitions are also motivated by economic network analysis. We mention applications of these problems in Section 1.2.

In a directed graph, vertex v is said to *in-cover* every incoming edge (u, v) and *out-cover* every outgoing edge (v, u) for some u. A vertex v is also said to *r-in-cover* all edges in the directed path to v of length at most r. Similarly, v is said to *r-out-cover* all edges in the directed path from v. Here, for a path $v_1, v_2, \ldots, v_\ell$, the *length* of the path is defined as the number of edges, that is, $\ell - 1$. In particular, if $r = 0$, a vertex is not considered to cover any edge. Then Directed r-In (Out) Vertex Cover is the following problem.

▶ **Definition 1.** Directed r-In (Out) Vertex Cover (r-In (Out) VC) is the problem that given a directed graph $G = (V, E)$ and two positive integers k and r, determines whether there exists a vertex subset $S \subseteq V$ of size at most k such that every edge in E is r-in (out)-covered by S. Such S is called an *r-in (out)-vertex cover.*

Furthermore, we define Directed (p, q)-Edge Dominating Set. An edge $e = (u, v)$ is said to *(p, q)-dominate* itself and all edges that vertex u p-in-covers and vertex v q-out-covers. In particular, edge (u, v) is said to *$(p, 0)$-dominate* (resp., *$(0, q)$-dominate*) itself and all edges p-in-covered by u (resp., q-out-covered by v).

Then Directed (p, q)-Edge Dominating Set is defined as follows.

▶ **Definition 2.** Directed (p, q)-Edge Dominating Set ((p, q)-EDS) is the problem that given a directed graph $G = (V, E)$, one positive integer k, and two non-negative integers p, q, determines whether there exists an edge subset $K \subseteq E$ of size at most k such that every edge is (p, q)-dominated by K. Such K is called a *(p, q)-edge dominating set.*

The undirected Edge Dominating Set problem is Dominating Set on (undirected) line graphs. We can see the same relationship between Directed $(0, 1)$-Edge Dominating Set and Dominating Set on directed line graphs. For a directed graph, a *directed line graph* is defined as follows:

▶ **Definition 3** ([18]). A *directed line graph* of $G = (V, E)$ is $L(G) = (E, E_2)$ such that

$$E_2 = \{((x, y), (z, w)) | (x, y), (z, w) \in E \wedge y = z\}.$$

It is obvious that a directed $(0, 1)$-edge dominating set on a directed graph G corresponds to a (directed) dominating set on the line graph of G. Furthermore, Directed $(1, 1)$-Edge Dominating Set corresponds to undirected Dominating Set on an underlying undirected graph of a directed line graph. These relations imply that our definition of Directed (p, q)-Edge Dominating Set is quite natural from the viewpoint of the line graph operation.

One interesting aspect of directed versions, but not undirected versions, is the asymmetry of the problem structures. For Directed r-In Vertex Cover, a vertex v in-covers only (u, v) when $r = 1$. Thus, a 1-in vertex cover is the set of all vertices whose in-degree is at least one. Therefore, it is trivial that Directed 1-In (Out) Vertex Cover is solvable in linear time, while undirected Vertex Cover is NP-complete. On the other hand, Directed $(1, 1)$-Edge Dominating Set, in a sense, corresponds to (undirected) Edge Dominating Set. For the optimization version, Edge Dominating Set is equivalent to Minimum Maximal Matching [24]. However, Directed $(1, 1)$-Edge Dominating Set does not necessarily correspond to matching on the undirected graphs underlying directed graphs due to the asymmetry of domination.

Table 1 Our results for graph classes. NP-c and $W[2]$-h stand for NP-complete and $W[2]$-hard, respectively.

Graph class	Tree	Planar DAG of bounded degree	DAG	General
1-In (Out) VC	-	-	-	$O(n)$
r-In (Out) VC ($r \geq 2$)	$O(n^4)$	NP-c	$W[2]$-h	$W[2]$-h
$(0,1),(1,0)$-EDS	$O(n^4)$	NP-c	NP-c	$2^{O(k)}n$
$(1,1)$-EDS	$O(n^4)$	NP-c	NP-c	$2^{O(k)}n$
(p,q)-EDS (p or $q \geq 2$)	$O(n^4)$	NP-c	$W[2]$-h	$W[2]$-h

For Directed (p,q)-Edge Dominating Set, there exists another source of asymmetry. That is, we can consider the case in which p and q are different. In the case in which $(p,q) = (0,1)$, edge (u,v) dominates itself and edges out-covered by v. Although Directed $(0,1)$-Edge Dominating Set is similar to Directed 1-Out Vertex Cover, surprisingly, it is NP-complete on directed acyclic graphs.

1.1 Our Contributions

Table 1 shows our results. In this paper, we first give hardness results for Directed r-In (Out) Vertex Cover and Directed (p,q)-Edge Dominating Set on restricted graphs, even on directed acyclic planar graphs of bounded degree. The hardness on directed acyclic graphs implies that we cannot design parameterized algorithms with respect to *directed treewidth* [19] and *DAG-width* [2] unless P=NP. The fact that Directed $(0,q)$-Edge Dominating Set is NP-complete even if $q = 1$ implies that r-Dominating Set on directed line graphs is NP-complete even if $r = 1$. Moreover, we prove that Directed r-In (Out) Vertex Cover is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs when $r \geq 2$, and Directed (p,q)-Edge Dominating Set is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs when either p or q is greater than 1. These results hold even if there are no multiple edges or loops.

On the other hand, we obtain algorithms for certain cases, including algorithms for all problems when restricted to trees, for any values of p, q, and r. The interplay among distance, direction, and domination results in a complex dynamic programming solution, running in $O(n^4)$ time. Because an edge can either dominate or be dominated by edges outside of a subtree depending on how it is directed, at each step of the algorithm we need to maintain extensive information not only about the subtree itself but also potential outside influence.

We show that Directed $(0,1),(1,0),(1,1)$-Edge Dominating Set is fixed-parameter tractable with respect to k. In particular, we give $2^{O(k)}n$-time algorithms. We emphasize that the running time of these algorithms is single exponential in k and linear in n. Moreover, our fixed-parameter algorithms are based on dynamic programming on a tree decomposition. Thus, we also show that Directed $(0,1),(1,0),(1,1)$-Edge Dominating Set can be solved in linear time on graphs whose underlying undirected graphs have bounded treewidth. Note that given a directed graph G and its underlying undirected graph G^*, the directed treewidth of G is no greater than its DAG-width which, in turn, is no greater than the treewidth of G^* [2].

1.2 Motivation and Application

As practical motivation, a number of network models employ directed graphs. For example, directed graphs are used to represent economic networks in which vertices correspond to industries and edges correspond to transactions of money or materials between industries [22, 17].

Recently, economists have used graph algorithms to analyze these economic networks in terms of graph structures in order to find critical industries and transactions [20, 23]. Based on the analyses, they discuss which kinds of economic policies should be adopted, and so on. However, there are some problems. Such analyses in economics are based on undirected graph algorithms instead of directed graph algorithms; they first transform directed graphs to undirected graphs, and then apply undirected graph algorithms to the graphs thus obtained. This is because there are many more results on graph optimization on undirected graphs than on directed graphs. Of course, such substitute algorithms might extract some information from the processed graph, but some important information is definitely lost. For example, when we would like to find a critical transaction in an economic network, the edge direction is clearly essential.

The theoretical motivation is a relationship between directed Dominating Set and Directed (p, q)-Edge Dominating Set. As we mentioned above, Directed $(0, 1)$-Edge Dominating Set is directed Dominating Set on directed line graphs and Directed $(1, 1)$-Edge Dominating Set is undirected Dominating Set on an underlying undirected graph of a directed line graph. Directed line graphs are well-studied for DNA sequencing and have some useful properties and characterizations [18, 3]. As for combinatorial problems on graphs, (directed) Hamiltonian Path on directed line graphs can be solved in time $O(n^2 + m^2)$ [4] while Hamiltonian Path on undirected line graphs is NP-complete [1]. Therefore, some directed problems could be easier than the undirected versions on line graphs. Unfortunately, however, our results show that directed Dominating Set and the distance version, that is, directed r-Dominating Set, remain NP-complete even on directed line graphs.

1.3 Related problems

One of the most famous covering problems is Vertex Cover. This is a classical NP-complete problem on undirected graphs, but known to be fixed-parameter tractable [6]. In terms of graph parameters, the size of the minimum vertex cover of G is called the *vertex cover number* of G. For any graph, it is easily seen that vertex cover number is greater than or equal to the treewidth [14].

Edge Dominating Set is the problem that given an undirected graph $G = (V, E)$ and an integer k, determines whether there exists a set of edges X of size at most k such that any edge in $E \setminus X$ has at least one incident edge in X. This problem is NP-complete even on bipartite, planar, and bounded degree graphs [24], but fixed-parameter tractable in general [13]. As we have seen, the Edge Dominating Set problem is equivalent to Dominating Set on line graphs. Moreover, the (optimization) Edge Dominating Set problem is equivalent to Minimum Maximal Matching [24].

Dominating Set is a classical domination problem. This problem is known to be $\Omega(\log n)$-inapproximable, but $O(\log n)$-approximable by a simple greedy algorithm on general graphs [9]. With respect to parameterized complexity, Dominating Set is $W[2]$-complete, unlike Vertex Cover and Edge Dominating Set [10]. Therefore, this problem is well-studied on restricted graphs. Recently, Dawar et al. [8] and Drange et al. [12] considered fixed-parameter tractability and the existence of problem kernels for some sparse graph classes. Their results include the distance version, that is, r-Dominating Set. This approach was generalized to directed graphs because the directed Dominating Set problem is also $W[2]$-complete [21].

The remainder of this paper is organized as follows. In Section 2, we first give basic terminology, notions, and definitions. In Section 3, we show the hardness results of the problems. In Section 4, we give polynomial-time algorithms on trees and fixed-parameter algorithms on general graphs. We prove many theorems (including lemmas) in this paper, but several of the proofs are omitted due to space limitations.

2 Preliminaries

In this section, we give notation and definitions. Let $G = (V, E)$ be a directed graph where $|V| = n$ and $|E| = m$. A vertex u is called an *in-neighbor* of v if there exists an edge (u, v) and a vertex w is called an *out-neighbor* of v if there exists an edge (v, w). Moreover, the sets of in (out)-neighbors of v are denoted by $N^{in}(v)$ (resp., $N^{out}(v)$). The number of in (out)-neighbor vertices of v is called the *in (out)-degree* and denoted by $indeg(v) := |N^{in}(v)|$ (resp., $outdeg(v) := |N^{out}(v)|$).

For two vertices u, v, the *distance* from u to v is defined as the number of edges in the shortest path from u to v, denoted by $dist(u, v)$. A vertex u such that $dist(u, v)$ is at most r is called an *r-in-neighbor* of v and a vertex w such that $dist(v, w)$ is at most r is called an *r-out-neighbor* of v. The sets of r-in (out)-neighbors of v are denoted by $N^{in}_r(v)$ (resp., $N^{out}_r(v)$). Note that $N^{in}_r(v) = N^{in}(v)$ and $N^{out}_r(v) = N^{out}(v)$ when $r = 1$.

In an undirected graph G^*, a set of edges such that no edges share an endpoint is called a *matching*. Furthermore, a matching is *maximal* if no proper superset is a matching. An edge dominating set is the edge set E' such that every edge in $E \setminus E'$ is adjacent to at least one edge in E'. Therefore, a maximal matching is an edge dominating set. As a typical design tool of parameterized algorithms, we make use of the tree decomposition and treewidth in this paper. We denote the *treewidth* of G^* by $\mathbf{tw}(G^*)$. For formal definitions of treewidth and tree decomposition, see [5], for example.

A directed graph G is called a *directed acyclic graph* (*DAG*) if G has no directed cycle and a *planar graph* if it can be embedded in the plane without any edges crossing. We mention results on such restricted graphs.

3 Hardness results

In this section, we discuss the hardness of Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set.

3.1 Directed (0, 1), (1, 0)-Edge Dominating Set

We first show that Directed $(0, 1), (1, 0)$-Edge Dominating Set is NP-complete. Although Directed $(0, 1)$-Edge Dominating Set is very similar to 1-Out Vertex Cover, there is a large gap in terms of time complexity.

To show this, we introduce a variant of the SAT problem. Let $(X, \mathcal{C})$ be an instance I of SAT, where $X = \{x_1, x_2, \ldots, n\}$ is the set of variables and $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ is the set of clauses. We consider a bipartite graph $G_I = (X \cup \mathcal{C}, E)$, where $E = \{\{x, C\} \mid x \in X, C \in \mathcal{C}$ such that $x \in C$ or $\bar{x} \in C\}$. An instance I of SAT is called *planar* if G_I is planar. Much is known concerning the planar version of SAT. For example, 3SAT is known to be NP-complete even if the instance is restricted to being planar. The restricted version of 3SAT is called Planar 3SAT.

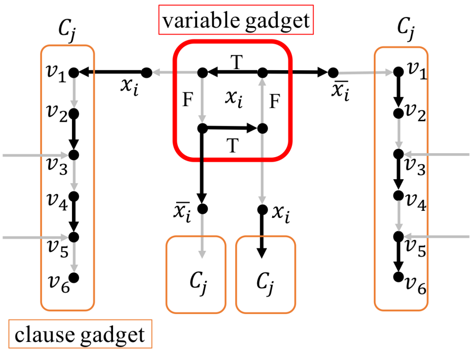


Figure 1 Constructed graph of the reduction from 3SAT to $(0,1)$-EDS

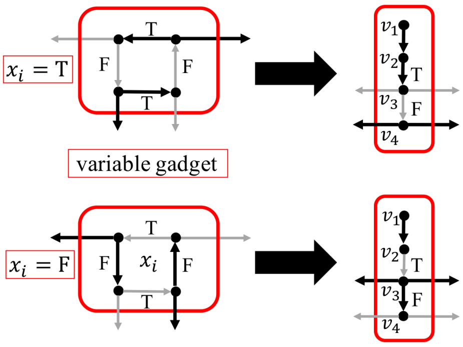


Figure 2 Replacing a cycle by a directed path for a variable's gadget

Here, we consider another restriction of Planar 3SAT. In the restricted instances, each literal appears at most twice, that is, $\forall y \in X \cup \bar{X} : |\{C \in \mathcal{C} \mid y \in C\}| \leq 2$. Instead, the size of each clause is relaxed to be not exactly three but at most three. We call this version Planar At-Most3SAT(L2).

▶ **Lemma 4.** Planar At-Most3SAT(L2) *is NP-complete.*

By using Lemma 4, we can obtain Theorem 5.

▶ **Theorem 5.** Directed $(0,1),(1,0)$-Edge Dominating Set *is NP-complete on directed planar graphs such that* $indeg(v) + outdeg(v) \leq 3$ *holds for any vertex.*

Proof. We only consider Directed $(0,1)$-Edge Dominating Set as the other proof is similar. This problem is clearly in NP. Thus, we show the hardness. The reduction is from Planar At-Most3SAT(L2).

Let n be the number of variables, m be the number of clauses, and l be the number of literals in an input Φ for Planar At-Most3SAT(L2). Then, we construct a graph as in Figure 1. First, we create n cycles of length four corresponding to the variables in Φ and m paths of length five corresponding to the clauses in Φ. For a variable's gadget, if we include the two horizontal edges in the $(0,1)$-edge dominating set, it corresponds to setting the variable to true in Φ. Otherwise, we include the two vertical edges, which corresponds to setting the variable to false. Note that the size of a minimum $(0,1)$-edge dominating set for a cycle of length four is two. In Figure 1, thick lines represent that they are included in the solution (we use the same convention in the other figures).

We connect each clause gadget to the variable gadgets corresponding to the literals in the clause, as follows. For $v_1, v_2, \ldots, v_6$ the vertices in the clause gadget, each of v_1, v_3, and v_5 is connected by a path of length two, called a *linking path*, to one of the vertices in a variable gadget. We can observe that there are l linking paths in the constructed graph. For each variable, there are at most two occurrences of true literals and at most two of false literals. Because the variable gadget has four vertices corresponding to literals, by connecting each vertex in the variable gadget to a clause gadget, for any vertex v in the constructed graph, $indeg(v) + outdeg(v) \leq 3$.

Finally, we conclude this proof by obtaining the following lemma.

▶ **Lemma 6.** *An input* Φ *for* Planar At-Most3SAT(L2) *has a satisfying truth assignment if and only if there exists a* $(0,1)$*-edge dominating set of size* $2n + l + 2m$ *in a constructed graph.* ◀

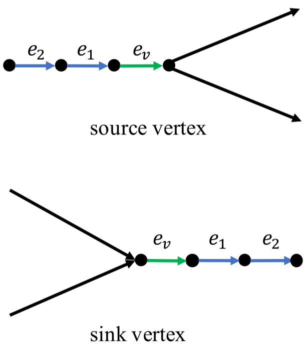


Figure 3 Vertex gadgets for source and sink in the reduction to $(1,1)$-EDS

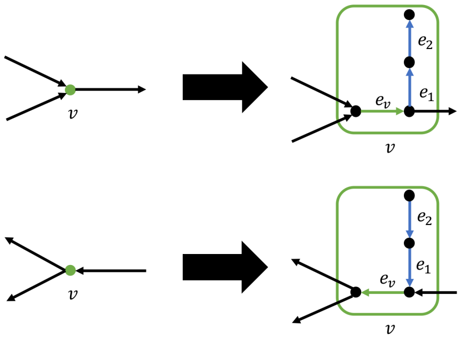


Figure 4 Vertex gadgets for other vertices in the reduction to $(1,1)$-EDS

By replacing each variable gadget by a path v_1, v_2, v_3, v_4 of length three and connecting vertex v_3 and true literals in a clause, and vertex v_4 and false literals (see Figure 2), we can also show that DIRECTED $(0,1)$-EDGE DOMINATING SET is NP-complete on directed acyclic planar graphs of bounded degree. Note that edge (v_1, v_2) is contained in any $(0,1)$-edge dominating set. Moreover, including edge (v_2, v_3) in the $(0,1)$-edge dominating set corresponds to setting the variable to true and including edge (v_3, v_4) corresponds to setting the variable to false.

▶ **Corollary 7.** DIRECTED $(0,1),(1,0)$-EDGE DOMINATING SET *is NP-complete on directed acyclic planar graphs such that* $indeg(v) + outdeg(v) \leq 4$ *holds for any vertex.*

3.2 Directed (1, 1)-Edge Dominating Set

As for DIRECTED $(1,1)$-EDGE DOMINATING SET, we obtain a stronger result in terms of a degree constraint. To show this, we first introduce a variant of planar graphs. A graph is *planar almost cubic* if it is planar, there are exactly two vertices of degree two, and the degree of all other vertices is three. We show that VERTEX COVER remains NP-complete on planar almost cubic graphs.

▶ **Lemma 8.** VERTEX COVER *on planar almost cubic graphs is NP-complete.*

By using Lemma 8, we show the following theorem.

▶ **Theorem 9.** DIRECTED $(1,1)$-EDGE DOMINATING SET *is NP-complete on directed acyclic planar graphs such that* $indeg(v) + outdeg(v) \leq 3$ *holds for any vertex.*

Proof. Since DIRECTED $(1,1)$-EDGE DOMINATING SET clearly belongs to NP, we prove the hardness. We show a reduction from VERTEX COVER on planar almost cubic graphs. Suppose that we are given an instance (G,k) of VERTEX COVER. For an undirected planar almost cubic graph G, we choose two vertices with degree two in G as source and sink vertices. We then arrange each vertex in a horizontal line such that the two vertices of degree two become ends of the line and orient every edge from left to right. Note that there exist exactly one source vertex such that the in-degree is zero and out-degree is two and exactly one sink vertex such that the in-degree is two and out-degree is zero. For other vertices v, it holds that $indeg(v) = 1$ and $outdeg(v) = 2$ or $indeg(v) = 2$ and $outdeg(v) = 1$. Each oriented edge corresponding to an edge in G is called an *original edge*.

Next, we attach paths of length three to the source vertex and the sink vertex as a vertex gadget as in Figure 3. Moreover, we replace any other vertex by a path of length three consisting of e_v, e_1, e_2 as in Figure 4. An edge e_v in G' corresponds to vertex v in G. Let G' be the constructed graph. Since we only replace vertices in G by paths, G' remains planar and acyclic and for any vertex v in G', $indeg(v) + outdeg(v) \leq 3$. Then the following lemma completes the proof.

▶ **Lemma 10.** *An instance (G, k) of* Vertex Cover *is a yes-instance if and only if an instance $(G', n+k)$ of* Directed $(1,1)$-Edge Dominating Set *is a yes-instance.* ◀

We also obtain the following result on the distance-generalized version.

▶ **Corollary 11.** Directed (p, q)-Edge Dominating Set *is NP-complete on directed acyclic planar graphs such that $indeg(v) + outdeg(v) \leq 3$ holds for any vertex when $p, q \geq 1$.*

3.3 Distance generalization

In this subsection, we consider the distance-generalized versions as with Corollary 11. We first show that Directed r-In (Out) Vertex Cover and Directed $(0, q), (p, 0)$-Edge Dominating Set are NP-complete on directed acyclic planar graphs of bounded degree.

▶ **Theorem 12.** *When r, p and q are greater than* 1*,* Directed r-In (Out) Vertex Cover *and* Directed $(0, q), (p, 0)$-Edge Dominating Set *are NP-complete on directed acyclic planar graphs such that $indeg(v) + outdeg(v) \leq 4$ holds for any vertex v.*

From Theorems 5 and 12, we can conclude directed r-Dominating Set on directed line graphs is NP-complete.

▶ **Corollary 13.** *The (directed) r-*Dominating Set *problem is NP-complete on directed line graphs even if $r = 1$.*

Finally, we show that Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set are $W[2]$-hard on directed acyclic graphs by a reduction from Set Cover, which is $W[2]$-complete and $\Omega(\log n)$-inapproximable [10, 9].

▶ **Theorem 14.** Directed r-In (Out) Vertex Cover *is $W[2]$-hard on directed acyclic graphs when $r \geq 2$.* Directed (p, q)-Edge Dominating Set *is $W[2]$-hard on directed acyclic graphs when $p \geq 2$ or $q \geq 2$. For these problems, there is no polynomial-time $c \ln k$-approximation algorithm for any constant $c < 1$ unless P=NP, where k is the size of an optimal solution, though they can be approximated within ratio $O(\log n)$ by a greedy algorithm.*

4 Algorithms

In this section, we give polynomial-time algorithms for Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set on trees and fixed-parameter algorithms for Directed $(0, 1), (1, 0), (1, 1)$-Edge Dominating Set on general graphs.

4.1 Algorithms on Trees

We solve Directed (p, q)-Edge Dominating Set by dynamic programming on a graph G for which the underlying undirected graph is a tree, which we can root at an arbitrary vertex; henceforth we use $\hat{G}$ to denote such a rooted tree. When we use the terms *parent*, *child*, *ancestor*, and *descendant*, we are referring to the relationships between vertices in $\hat{G}$.

We first extend the definition of distance to specify distances between vertices and edges. For an edge $e = (u, v)$ and vertices w and x, we define $dist(w, e)$ to be $dist(w, u)$ and $dist(e, x)$ to be $dist(v, x)$. Moreover, for two edges $e = (u, v)$ and $f = (x, y)$, we define $dist(e, f)$ to be $dist(v, x)$. An edge e *i-in-dominates* (or just *in-dominates*) all edges f such that $dist(f, e) \leq i$ and an edge e *j-out-dominates* (or just *out-dominates*) all edges f such that $dist(e, f) \leq j$. In a directed path containing edges e and f, the edges (not including e and f) traversed along the path are *between* e and f. If there are k edges between e and f, then e $(k + 1)$-out-dominates f and f $(k + 1)$-in-dominates e.

In $\hat{G}$, we use T_v to denote the subtree rooted at the vertex v, and $G[T_v]$ to denote the subgraph of (the directed graph) G induced on the vertices in T_v. We call $G[T_v]$ the *subtree of G rooted at v* and use $conn(v)$ to denote the edge connecting v to its parent, if it has one. We refer to a vertex v as an *out-vertex* if $conn(v)$ is directed from v to its parent and a *in-vertex* if $conn(v)$ is directed from v's parent to v. If v is the root of $\hat{G}$, it is neither an out-vertex nor an in-vertex. We use $same(v)$ and $diff(v)$ to denote the sets of children of v that are out-vertices and in-vertices, respectively, if v is an out-vertex and that are in-vertices and out-vertices, respectively, if v is an in-vertex. Furthermore, we use $ST(v)$ to denote the set of subtrees rooted at vertices in $same(v)$ and $DT(v)$ to denote the set of subtrees rooted at vertices in $diff(v)$; these are considered to be two different *types of subtrees*. In addition, we use C_s to denote the set of edges between v and vertices in $same(v)$, and C_d to denote the set of edges between v and vertices in $diff(v)$; just as there are two types of subtrees, we consider these set to constitute two types of *connecting edges*.

Our dynamic-programming algorithm processes vertices in an order such that a vertex v is processed after all its descendants, where we use information about the subtrees rooted at the children of v to determine how to dominate edges in $G[T_v]$. We store not only the sizes of edge dominating sets, but also the sizes of edge dominating sets defined in terms of their *reach* and *deficit*, which are measures of the impact of edges inside a subtree in the domination of edges outside the subtree and the impact of edges outside a subtree in the domination of edges inside the subtree.

To see how edges in subtrees rooted at children of v can have an impact on each other, suppose v has two children w and x such that w is an out-vertex and x is a in-vertex. Furthermore, consider an edge e_w in $G[T_w]$ such that $dist(e_w, w) = i$ and an edge e_x in $G[T_x]$ such that $dist(x, e_x) = j$. We can form a directed path that starts at e_w and traverses the edges (w, v) and (v, x) to end at e_x. The number of edges between e_w and e_x is $i + j + 2$, which means that e_w $(i + j + 3)$-out-dominates e_x and that e_x $(i + j + 3)$-in-dominates e.

To determine the reach of a set of edges K in $G[T_v]$, we first determine the shortest distance i from an edge in K to v, if v is an out-vertex, or the shortest distance i from v to an edge in K, if v is an in-vertex. When v is an endpoint of an edge in K (that is, $i = 0$), that edge will be able to q-out-dominate an edge outside of $G[T_v]$, if v is an out-vertex, or p-in-dominate an edge outside of $G[T_v]$, if v is an in-vertex. We thus define $maxreach(v) = q$ for each out-vertex v and $maxreach(v) = p$ for each in-vertex v. More generally, we define the *reach of K beyond $G[T_v]$* to be $maxreach(v) - i$.

To measure which edges depend on outside edges for domination, we define the *deficit of K within* $G[T_v]$ to be maximum over $dist(e, v)$ (resp., $dist(v, e)$) over all edges e in $G[T_v]$ not (p, q)-dominated by any edge in K, for v an out-vertex (resp., in-vertex). Since the edge between v and its parent is the outside edge that can cover the largest deficit, we set $maxdeficit(v) = p$ for v an out-vertex and $maxdeficit(v) = q$ for v a in-vertex. We refer to all edges e with $dist(e, v) \leq d$ (resp., $dist(v, e) \leq d$) to be *edges of deficit of most d* in $G[T_v]$, for v an out-vertex (resp., an in-vertex). Should an edge outside a subtree have sufficient reach to dominate all edges of deficit at most d, we will say that the edge *covers the deficit*.

Using these concepts, we say that a set of edges K is a *reach-r-deficit-d edge dominating set* for $G[T_v]$ if the reach of K beyond $G[T_v]$ is r, and K (p,q)-dominates $G[T_v \backslash J]$ where J is the set of edges of deficit at most d in $G[T_v]$. In our algorithm, we use $D[v,r,d]$ to store the minimum number of edges in a reach-r-deficit-d edge dominating set for $G[T_v]$.

When processing a vertex v, we determine $D[v,r,d]$ for values of r and d in the ranges $0 \leq r \leq maxreach(v)$ and $0 \leq d \leq maxdeficit(v)$. For the base cases, for each leaf v in $\hat{G}$, we set $D[v,r,d] = 0$ for all values of r and d. To determine the value of $D[v,r,d]$, we will consider all possible options for adding edges between v and its children to K, a reach-r-deficit-d edge dominating set for $G[T_v]$, as the choice of edges of K in the subtrees rooted at the children of v will be represented by already-computed table entries.

The computation of the table entries depends on the following lemmas.

▶ **Lemma 15.** *The reach of K beyond $G[T_v]$ is $maxreach(v)$ if and only if $K \cap C_s \neq \emptyset$.*

▶ **Lemma 16.** *If $K \cap C_s = \emptyset$, the reach of K beyond $G[T_v]$ is one less than the maximum over all vertices $u \in same(v)$ of the reach of K restricted to $G[T_u]$.*

▶ **Lemma 17.** *For any child u of v, $conn(u)$ covers a deficit of $maxdeficit(u)$ in $G[T_u]$.*

▶ **Lemma 18.** *For any child u of v, if $conn(u)$ is not included in K, then the maximum possible deficit within $G[T_v]$ that can be covered by K is $maxdeficit(u) - 1$.*

▶ **Lemma 19.** *For any child u of v, if $conn(u)$ is not included in K, the deficit in $G[T_v]$ will be covered by any single connecting edge of the opposite type. Thus, if $K \cap C_d \neq \emptyset$, $d = 0$.*

The complete proofs of Theorems 20 and 21 are omitted.

▶ **Theorem 20.** *There is an algorithm that solves* Directed (p,q)-Edge Dominating Set *on trees in $O(n^4)$-time.*

▶ **Theorem 21.** *There is an algorithm that solves* Directed r-In (Out) Vertex Cover *on trees in $O(n^4)$-time.*

4.2 Fixed-Parameter Algorithm for Directed Edge Dominating Set

In this subsection, we give a $2^{O(k)}n$-time algorithm for Directed $(1,1)$-Edge Dominating Set. First, we obtain the following lemmas and theorem.

▶ **Lemma 22.** *Given a directed graph G, let G^* be the underlying undirected graph of G and s be the minimum size of* Directed $(1,1)$-Edge Dominating Set *on G. Then the following inequality holds: $\mathbf{tw}(G^*) \leq 2s$.*

Proof. Let G^* be an undirected graph, $\mathbf{tw}(G^*)$ be the treewidth of G^*, and $\mathbf{vc}(G^*)$ be the size of minimum vertex cover. Then we have $\mathbf{tw}(G^*) \leq \mathbf{vc}(G^*)$ [14]. Let M^* be a minimum maximal matching in G^*. A minimum $(1,1)$-edge dominating set in G is an (not necessarily minimum) edge dominating set in G^*. If not, there is an edge not dominated by the $(1,1)$-edge dominating set in G. Moreover, for any edge dominating set D in undirected graphs, $|D| \geq |M^*|$ holds because a minimum maximal matching is a minimum edge dominating set [24]. Therefore, $s \geq |M^*|$ holds. On the other hand, we have a well-known result that for any maximal matching M, $\mathbf{vc}(G^*) \leq 2|M|$ [16]. Moreover, we already know that $\mathbf{tw}(G^*) \leq \mathbf{vc}(G^*)$ holds. Finally, we can obtain $\mathbf{tw}(G^*) \leq 2s$. ◀

▶ **Lemma 23.** *Given a directed graph G, let G^* be the underlying undirected graph of G. Then given a tree decomposition of G^* of width at most ℓ, there exists an algorithm that solves* Directed $(1,1)$-Edge Dominating Set *in $25^{\ell}\ell^{O(1)}n$-time.*

▶ **Theorem 24** ([5]). *There exists an algorithm that, given an n-vertex graph G and an integer ℓ, in time $2^{O(\ell)}n$ either outputs that the treewidth of G is larger than ℓ, or constructs a tree decomposition of G of width at most $5\ell + 4$.*

Finally, Directed $(1,1)$-Edge Dominating Set can be solved in the following time.

▶ **Theorem 25.** *Given an instance (G,k) of* Directed $(1,1)$-Edge Dominating Set*, it can be solved in $2^{O(k)}n$-time.*

Proof. Given an instance (G,k), we first determine whether the treewidth of G^* is at most $2k$ in $2^{O(k)}n$-time by using Theorem 24. If $\mathbf{tw}(G^*) > 2k$, we conclude that it is a no-instance by Lemma 22. Otherwise, we use the $25^{\ell}\ell^{O(1)}n$-time algorithm based on a tree decomposition of width at most $10k + 4$ obtained by Theorem 24. Therefore, the total running time is $2^{O(k)}n + 25^{10k+4}(10k + 4)^{O(1)}n = 2^{O(k)}n$. ◀

Thus, Directed $(1,1)$-Edge Dominating Set is fixed-parameter tractable with respect to k. We emphasize that the running time of this algorithm is single exponential in k and linear in n. In the same way, we can prove Directed $(0,1),(1,0)$-Edge Dominating Set is fixed-parameter tractable with respect to k.

▶ **Theorem 26.** *Given an instance (G,k) of* Directed $(0,1),(1,0)$-Edge Dominating Set*, it can be solved in $2^{O(k)}n$-time.*

References

1 Alan A. Bertossi. The edge hamiltonian path problem is NP-complete. *Information Processing Letters*, 13(4):157–159, 1981.

2 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The dag-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.

3 Jacek Blazewicz, Alain Hertz, Daniel Kobler, and Dominique de Werra. On some properties of DNA graphs. *Discrete Applied Mathematics*, 98(1):1–19, 1999.

4 Jacek Blazewicz, Marta Kasprzak, Benjamin Leroy-Beaulieu, and Dominique de Werra. Finding hamiltonian circuits in quasi-adjoint graphs. *Discrete Applied Mathematics*, 156(13):2573–2580, 2008.

5 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

6 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P^*. *SIAM Journal on Computing*, 22(3):560–572, 1993.

7 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264–1275, 2008.

8 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *Proceedings of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS2009)*, pages 157–168. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.

9 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing (STOC2014)*, pages 624–633. ACM, 2014.

10 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

11 Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. In *Proceedings of Feasible Mathematics II*, pages 219–244. Birkhäuser Boston, 1995.

12 Pål Grønås Drange, Markus Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *Proceedings of 33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, pages 31:1–31:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

13 Henning Fernau. edge dominating set: Efficient enumeration-based exact algorithms. In *Proceedings of Parameterized and Exact Computation: Second International Workshop (IWPEC2006)*, pages 142–153. Springer Berlin Heidelberg, 2006.

14 Jiří Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, 2011.

15 Robert Ganian, Petr Hliněný, Joachim Kneis, Alexander Langer, Jan Obdržálek, and Peter Rossmanith. Digraph width measures in parameterized algorithmics. *Discrete Applied Mathematics*, 168:88–107, 2014.

16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

17 Tesshu Hanaka, Shigemi Kagawa, Hirotaka Ono, and Keiichiro Kanemoto. Finding environmentally critical transmission sectors, transactions, and paths in global supply chain networks. *Energy Economics*, 2017. (in press).

18 Frank Harary and Robert Z. Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2):161–168, 1960.

19 Thor Johnson, Neil Robertson, P.D. Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

20 Shigemi Kagawa, Sangwon Suh, Klaus Hubacek, Thomas Wiedmann, Keisuke Nansai, and Jan Minx. CO_2 emission clusters within global supply chain networks: Implications for climate change mitigation. *Global Environmental Change*, 35:486–496, 2015.

21 Stephan Kreutzer and Siamak Tazari. Directed nowhere dense classes of graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2012)*, pages 1552–1562. Society for Industrial and Applied Mathematics, 2012.

22 Michael Lahr and Erik Dietzenbacher. *Input-Output Analysis: Frontiers and Extensions.* Palgrave Macmillan UK, 2001.

23 Omar Rifki, Hirotaka Ono, and Shigemi Kagawa. The robustest clusters in the input–output networks: global CO_2 emission clusters. *Journal of Economic Structures*, 6(1):3, 2017.

24 Mihalis Yannakakis and Fanica Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

Settlement Fund Circulation Problem*

Hitoshi Hayakawa[1], Toshimasa Ishii[2], Hirotaka Ono[3], and Yushi Uno[4]

1 **Graduate School of Economics, Hokkaido University, Sapporo, Japan**
hit.hayakawa@econ.hokudai.ac.jp

2 **Graduate School of Economics, Hokkaido University, Sapporo, Japan**
ishii@econ.hokudai.ac.jp

3 **Graduate School of Informatics, Nagoya University, Nagoya, Japan**
ono@nagoya-u.jp

4 **Graduate School of Engineering, Osaka Prefecture University, Sakai, Japan**
uno@cs.osakafu-u.ac.jp

Abstract

In the economic activities, the central bank has an important role to cover payments of banks, when they are short of funds to clear their debts. For this purpose, the central bank timely puts funds so that the economic activities go smooth. Since payments in this mechanism are processed sequentially, the total amount of funds put by the central bank critically depends on the order of the payments. Then an interest goes to the amount to prepare if the order of the payments can be controlled by the central bank, or if it is determined under the worst case scenario. This motivates us to introduce a brand-new problem, which we call the settlement fund circulation problem. The problems are formulated as follows: Let $G = (V, A)$ be a directed multigraph with a vertex set V and an arc set A. Each arc $a \in A$ is endowed debt $d(a) \geq 0$, and the debts are settled sequentially under a sequence π of arcs. Each vertex $v \in V$ is put fund in the amount of $p_\pi(v) \geq 0$ under the sequence. The minimum/maximum settlement fund circulation problem (Min-SFC/Max-SFC) in a given graph G with debts $d : A \to \mathbb{R}_+ \cup \{0\}$ asks to find a bijection $\pi : A \to \{1, 2,\dots, |A|\}$ that minimizes/maximizes the total funds $\sum_{v \in V} p_\pi(v)$. In this paper, we show that both Min-SFC and Max-SFC are NP-hard; in particular, Min-SFC is (I) strongly NP-hard even if G is (i) a multigraph with $|V| = 2$ or (ii) a simple graph with treewidth at most two, and is (II) (not necessarily strongly) NP-hard for simple trees of diameter four, while it is solvable in polynomial time for stars. Also, we identify several polynomial time solvable cases for both problems.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.2 Discrete Mathematics

Keywords and phrases Fund settlement, Algorithm, Digraph, Scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.46

1 Introduction

Background

In the economic activities, when a company borrows money, it owes a debt and the debt is not cleared until the debtor pays its amount. If the debtor fails to prepare cash for the

* This work is partly supported by KAKENHI 15H02965, 16K00001, 17H01698, 17K00017, 26241031, 26280001, 26540005 and by JST CREST Grant Number JPMJCR1402, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 46; pp. 46:1–46:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

payment until the deadline, it will go bankrupt. Such bankruptcy should be avoided when it could cause significant damage to the economy, and it is particularly true for the case of banks since their debts are highly interconnected each other and bankruptcy of a bank may cause chain reaction of bankruptcy. It is one of the reasons that debts among banks are cleared in a special system, called *interbank settlement system*, in which the central bank supports cash management of the banks.

In the system, cash held by the central bank is used as the fund for the payments. When a bank does not have enough funds for clearing its debts, the central bank will lend the necessary amount. Suppose, for example, that there are three banks, say A, B, and C, and they form debts such that A owes 50 to B, and B owes 30 to C, and A and B currently have 10 each on its own. Now if A pays for its debt, then A is short of 40. Therefore, the central bank is requested to put 40 in order to fill the shortage. Once 40 is put on A, it can clear its debt 50 to B, and then B can also clear its debt 30 by using its own funds 10 and a part of the received funds 50. Note that we assume each debt has to be cleared independently and "sequentially", that is, it is not allowed to cancel out payments; A pays 30 directly to C, and the rest 20 to B, for example.[1]

Objective

Now, suppose that B pays before A does. Then, the central bank has to put 20 to B, and in addition, 40 to A. This illustrates, in general, that the total amount of funds put to clear all debts depends on the order of the payments. Since funds in an interbank settlement system is scarce resource in the public interest, the efficient usage is socially desirable. Accordingly, one of the important roles of the central bank is to minimize the total funds put to clear the debts. Then we can consider a problem that finds the minimum total funds put to clear all debts by deciding a sequence of the payments, which we formulate as Min-SFC.

In a different perspective, another role of the central bank is to prepare for the worst case scenario such that it could hardly control the sequence of the payments. It is typical at the time of financial disruption and is crucially important. These observations again motivate us to define a corresponding maximization version of the problem, that is, to estimate the maximum funds that have to be put to clear all given debts, which we formulate as Max-SFC. It is quite significant to obtain insights concerning the desirable sequence of the payments in order to argue relevant policies.

Technically, both problems are formulated as optimization problems on networks. However, the nature of our problems is essentially different from the classic flow problems in the sense that the amount of each "debt" (flow) cannot be split at the time of the payment. On the contrary, such unsplittable flows come to have a feature that once some debt is cleared, then the transferred funds are accumulated in the bank's "account" and they can be split arbitrarily for the subsequent payments.

History and Perspective in Economics

Historically, we can find a primitive concern of fund circulation in the renowned Quesnay's "Economic Table" [10]. Only recently, Rotemberg explicitly discusses the amount of required funds in the context of interbank settlements [12], though he does not give its general formulation. A general formulation to derive each of the minimum and maximum amount of

[1] Sequential clearing is standard in the modern interbank settlement systems, as World Bank documents that 116 of 139 surveyed countries have adopted sequential clearing based systems up to 2010 [14].

required funds is then given by Hayakawa [5] for the purpose of economic analysis.[2] This paper now gives, from the computational aspect, detailed mathematical formulations for these problems as Min-SFC and Max-SFC, and presents a series of algorithmic or complexity results based on solid observations for the first time.

In the wake of the recent world-wide financial crisis, analyzing "dominos" of default comes to have critical importance. Seminal studies in the literature effectively assume "simultaneous" clearing that makes payments cancel out whenever possible, not only bilaterally but also multilaterally, though "sequential" clearing, which we assume, is standard in the modern interbank settlement systems. The assumption of simultaneous clearing lets the relevant analyses be highly tractable [1, 2], however, it could considerably underestimate the amount of funds required to prevent "dominos" of default. In the light of these, we believe that the study in this paper serves as fundamental tools of the estimation and suggests a new methodology in the analyses that is applicable to complex economic situations in reality.

This paper is organized as follows. In Section 2, after giving several terminologies, we formalize our problem of interests and show some examples. Sections 3 and 4 discuss the minimization version of the problem, and show tractable and intractable cases, respectively. Section 5 deals with the maximization version. Finally in Section 6, future work is described.

2 Preliminaries

2.1 Definitions and Terminology

For a positive integer n, let $[n] = \{1, 2, \dots, n\}$. For a finite set V, a family $\mathcal{X}$ of subsets in V is a *partition of* V if $\bigcup_{X \in \mathcal{X}} X = V$ holds and every two distinct sets in $\mathcal{X}$ are disjoint.

A directed graph (digraph) D is an ordered pair of its vertex set $V(D)$ and arc set $A(D)$ and is denoted by $D = (V(D), A(D))$, or simply $D = (V, A)$. An arc, an element of $A(D)$, is an ordered pair of vertices, and is denoted by $a = (u, v)$; this is distinct from (v, u). For an arc $a = (u, v)$, u is its *start vertex* and v is *end vertex*; they are denoted by $s(a)$ and $t(a)$, respectively. A digraph D is *multiple* when $A(D)$ is a multiple set; otherwise it is *simple*.

The *underlying graph* of a digraph D is an undirected graph G_D whose vertex set is $V(D)$ and edge set $E(G_D)$ has an edge $\{u, v\}$ as its element if and only if $(u, v) \in A(D)$ or $(v, u) \in A(D)$. A digraph D is *weakly connected* if its underlying graph G_D is connected. We assume throughout the paper that all digraphs are weakly connected. We usually use n and m to denote the number of vertices and arcs (edges), respectively, of a graph.

The *degree* of v is the number of arcs incident on v. We use $\Delta(D)$ to denote the maximum degree of a digraph D. Let $N_D(v)$ denote the set of vertices u with $(u, v) \in A(D)$ or $(v, u) \in A(D)$. Let $D[V']$ (resp., $D[A']$) denote the subgraph of D induced by a subset $V' \subseteq V(D)$ of vertices (resp., a subset $A' \subseteq A(D)$ of arcs). For a digraph D and a subset $A' \subseteq A(D)$ of arcs, we denote by $D \setminus A'$ the subgraph of D obtained from D by deleting A'.

2.2 Models and Problem Description

In the paper, we describe our problem by a digraph whose nodes are banks and arcs are loan relationship from one bank to another together with debts as arc weights.

Given a digraph $D = (V, A)$, *debt* of arcs is a function $d : A \to \mathbb{R}_+ \cup \{0\}$. For convenience, we sometimes introduce a (virtual) arc $a = (u, v)$ with $d(a) = 0$ if $(v, u) \in A$ and $(u, v) \notin A$.

[2] The relevant chapter of the paper [5] is reorganized as an independent article [6] with additional results.

A debt function d is *uniform* if $d(a) = c$ (constant) for all $a \in A$, otherwise *non-uniform*; it is *unit* if d is uniform and $c = 1$. Debts on a vertex v is *balanced* if $\sum_{(u,v)\in A} d(u,v) = \sum_{(v,w)\in A} d(v,w)$, and debts on a pair of vertices u and v is *symmetric* if $d(u,v) = d(v,u)$.

In our model, debts on arcs are settled individually in a single installment and sequentially. We say that a debt on an arc is *cleared* when it is settled (or, simply *clear* the arc), and we *put* funds on vertices to clear debts on their out-going arcs. When an arc is cleared, the amount for it is accumulated on the end vertex of the arc and can be reused for subsequent settlements. The amount of funds existing on a vertex is called its *residual*. A sequence of arcs, which corresponds to the order of selecting arcs to be cleared, can be represented as a permutation $\pi : A \to \{1, 2, \ldots, |A|\}$. We sometimes refer to this permutation as a *sequence* of A. We denote by $p_\pi(u,i)$ the fund put on u and by $r_\pi(u,i)$ the residual of u, immediately before putting fund $p_\pi(u,i)$ to clear arc $\pi^{-1}(i)$ for all $u \in V$ and for $i = 1, \ldots, |A|$. Then we can clear the debt on arc (u,v) if $r_\pi(u, \pi(u,v)) + p_\pi(u, \pi(u,v)) \geq d(u,v)$. We assume that we always put the minimum amount of funds to clear an arc, that is, $p_\pi(s(a), \pi(a)) = \max\{0, d(a) - r_\pi(s(a), \pi(a))\}$.

Now we define the *minimum settlement fund circulation problem* (Min-SFC) and the corresponding maximization problem (Max-SFC), which are introduced in [5] in the context of the interbank fund settlement systems, as follows.

Min-SFC (Max-SFC)
Instance: a digraph $D = (V, A)$ and debt $d : A \to \mathbb{R}_+ \cup \{0\}$.
Question: minimize (maximize) $\sum_{v\in V} p_\pi(v)$ ($\triangleq p_\pi(V)$)
subject to permutation $\pi : A \to \{1, \ldots, |A|\}$
and $p_\pi(s(a), \pi(a)) + r_\pi(s(a), \pi(a)) \geq d(a)$ for all $a \in A$, where
$p_\pi(u, 0) = 0, r_\pi(u, 0) = 0$ for all $u \in V$,

$$r_\pi(u,i) = \begin{cases} r_\pi(u, i-1), & \pi^{-1}(i-1) \text{ is not incident on } u, \\ r_\pi(u, i-1) + d(\pi^{-1}(i-1)), & \pi^{-1}(i-1) \text{ is incident to } u, \\ \max\{0, r_\pi(u, i-1) - d(\pi^{-1}(i-1))\}, & \pi^{-1}(i-1) \text{ is incident from } u. \end{cases}$$

Here, we define $p_\pi(v) = \sum_{i=1}^{|A|} p(v,i)$, and for notational convenience, we often let $p_\pi(X) = \sum_{v\in X} p_\pi(v)$ for a subset X of $V(D)$ in a digraph D and a sequence π of $A(D)$.

We show examples of Min-SFC and Max-SFC in Figure 1, and see how debts are cleared in detail. In the sequence π_1, 20 funds are put on v_5 to clear the debt $d(v_5, v_6) = 20$ for the first arc $\pi_1^{-1}(1) = (v_5, v_6)$; $p_{\pi_1}(v_5, 1) = 20$. The 20 funds are transferred to and accumulated in v_6; $r_{\pi_1}(v_6, 2) = 20$. The second arc $\pi_1^{-1}(2) = (v_3, v_6)$ is cleared by putting 10 funds on v_3; $p_{\pi_1}(v_3, 2) = 10$. The 10 funds are transferred to v_6, and it turns out that the residual on v_6 becomes $20 + 10 = 30$; $r_{\pi_1}(v_6, 3) = 30$. Now, the residual are used for clearing the third arc $\pi_1^{-1}(3) = (v_6, v_1)$ and no additional fund needs to be put on v_6; $r_{\pi_1}(v_1, 4) = 30$. We remark here again that a debt can only be cleared by a single installment. Also a residual can be split. Next, therefore, a part 20 of the residual 30 of v_1 is used for clearing $\pi_1^{-1}(4) = (v_1, v_2)$, where $d(v_1, v_2) = 20$, and so on.

2.3 Summary of the Results

The results of this paper are summarized in Table 1. To explain the table and for the use throughout the paper, we introduce some additional definitions. For a digraph D, if the underlying graph G_D of D belongs to some class $\mathcal{C}$ of graphs, then we may simply say that D belongs to $\mathcal{C}$ if no confusion occurs. A digraph is called *balanced* if debts on each vertex is

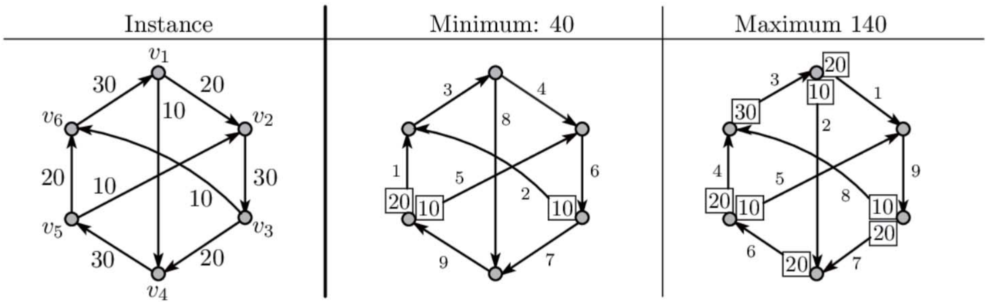


Figure 1 **[Left]** An instance of both Min-SFC and Max-SFC; a digraph $D = (V, A)$ with the debts $d(a)$ beside each arc a. **[Middle, Right]** Sequences π_1 and π_2 of A, respectively. The number beside each arc $a \in A$ indicates $\pi_i(a)$ and the number in the square attached to $s(a)$ indicates the amount of funds put on $s(a)$ for clearing the debt of a in π_i, i.e., $p_{\pi_i}(s(a), \pi_i(a))$ $(i = 1, 2)$. In fact, π_1 and π_2 are optimal solutions of Min-SFC and Max-SFC for D, with $p_{\pi_1}(V) = 40$ and $p_{\pi_2}(V) = 140$, respectively.

balanced. A digraph D is called *symmetric* if debts on each pair of vertices u and v with $\{u, v\} \in E(G_D)$ is symmetric. A digraph is called *uniform* if its debt function is uniform.

We emphasize here that all the results are new. Especially, we can see that those for general and simple graphs show sharp border with respect to the complexity in the sense that it is tractable for stars, but is intractable for trees.

3 Min-SFC: Intractable Cases

In the subsequent two sections (Sections 3 and 4), we discuss about Min-SFC, which is our main interest in the context of analyzing settlements of debts. We first observe in this section that the problem is hard in general, but later in Section 4 we will see that it is tractable in some practical cases. Throughout these two sections, for an instance (D, d) of Min-SFC, we denote by $opt(D, d)$ the minimum amount of funds put on $V(D)$ for clearing all arcs in D, i.e., $opt(D, d) = \min\{p_\pi(V(D)) \mid \pi \text{ is a sequence of } A(D)\}$.

Now let $D = (V, A)$ be a multiple digraph. We show that even if $|V| = 2$ or D is balanced, Min-Sfc with D is strongly NP-hard by a reduction from 3-Partition, which is known to be strongly NP-hard [4, p.224].

3-Partition
Instance: $(\{x_1, x_2, \dots, x_{3m}\}, B)$: A set of $3m$ positive integers $x_1, x_2, \dots, x_{3m}$ and an integer B such that $\sum_{i \in [3m]} x_i = mB$ and $B/4 < x_i < B/2$ for each $i \in [3m]$.
Question: Is there a partition $\{X_1, X_2, \dots, X_m\}$ of $[3m]$ such that $\sum_{i \in X_j} x_i = B$ for each $j \in [m]$?

▶ **Theorem 1.** *For a multiple digraph D,* Min-SFC *is strongly NP-hard even if $|V(D)| = 2$ or D is balanced.*

Proof. Take an instance $I_{\text{3PART}} = (\{x_1, x_2, \dots, x_{3m}\}, B)$ of 3-Partition. From the I_{3PART}, we construct an instance $I_{\text{SFC}} = (D = (V, A), d)$ of Min-SFC as follows. Let $V = \{u, v\}$ and A be the set of arcs consisting of $3m$ multiple arcs from u to v and m multiple arcs from v to u; denote an arc from u to v by a_i, $i \in [3m]$, and an arc from v to u by b_j, $j \in [m]$. Let $d(a_i) = x_i$ for $i \in [3m]$ and $d(b_j) = B$ for $j \in [m]$. Note that D is balanced.

Table 1 Summary of our results in this paper together with their corresponding theorem/lemma numbers; Linear and P stand for linear and polynomial time solvable, respectively, and T, C and L in brackets stand for Theorem, Corollary and Lemma, respectively.

arcs		graphs				
debt	multiplicity	dag	path	star	tree	larger classes
MIN-SFC						
uniform	multiple	Linear [T4]				
symmetric	simple	—	P [T7]			
balanced	multiple	—	strongly NP-hard for two vertices [T1]			
	simple	—	P	P	P [C8]	strongly NP-hard for bipartite or tw≤ 2 [T2]
general	simple	Linear [T6]	P [C12]	P [T10]	NP-hard [T3] FPT wrt. Δ [T11]	
	multiple	Linear [T6]				
MAX-SFC						
uniform		NP-hard [T17]				
general	multiple	Linear [T16]				

We claim that the instance $I_{3\text{PART}}$ is a yes-instance of 3-PARTITION if and only if there exists a sequence π of A with $p_\pi(V) \leq B$. Notice that since I_{SFC} can be constructed from $I_{3\text{PART}}$ in polynomial time, this claim proves the theorem.

First, we show "only-if" part. Assume that $I_{3\text{PART}}$ is a yes-instance of 3-PARTITION; there exists a partition $\{X_1, X_2, \dots, X_m\}$ of $[3m]$ such that $\sum_{i \in X_j} x_i = B$ for each $j \in [m]$. Without loss of generality, let $X_j = \{3j-2, 3j-1, 3j\}$ for $j \in [m]$ (note that $|X_j| = 3$ holds since $B/4 < x_i < B/2$ for each $i \in [3m]$). Then the sequence π of A defined as (b_1, a_1, a_2, a_3, b_2, a_4, a_5, a_6, $b_3, \dots,$ $b_m, a_{3m-2}, a_{3m-1}, a_{3m}$) satisfies $p_\pi(V) = B$. Notice that $p_\pi(v, 1) = B$, $p_\pi(v, \ell) = 0$ for all $\ell \geq 2$, and $p_\pi(u) = 0$.

Next we show "if" part. Assume that there exists a sequence π of A with $p_\pi(V) \leq B$. Since $d(b_j) = B$, we have $opt(D, d) \geq B$ and hence $p_\pi(V) = B$. Without loss of generality, assume that $\pi(b_1) < \pi(b_2) < \cdots < \pi(b_m)$. For $j \in [m-1]$, let X_j be the set of indices $i \in [3m]$ such that $\pi(b_j) < \pi(a_i) < \pi(b_{j+1})$. Since we need funds with amount B for clearing b_1 and $p_\pi(V) = B$ holds, no additional fund is put on V when any arc $a' \in A$ with $\pi(a') > \pi(b_1)$ is cleared. Hence, the total debts of arcs cleared between b_j and b_{j+1} is exactly B, i.e., $\sum_{i \in X_j} x_i = B$ for each $j \in [m-1]$. Furthermore, since $B/4 < x_i < B/2$ for $i \in [3m]$, we have $|X_j| = 3$ for each $j \in [m-1]$. Let $X_m = [3m] \setminus (\bigcup_{i=1}^{m-1} X_j)$. Note that $|X_m| = 3m - \sum_{j=1}^{m-1} |X_j| = 3$ and $\sum_{i \in X_m} x_i = mB - \sum_{j=1}^{m-1} \sum_{i \in X_j} x_i = B$. Thus, the partition $\{X_1, X_2, \dots, X_m\}$ of $[3m]$ shows that $I_{3\text{PART}}$ is a yes-instance of 3-PARTITION. ◀

Let D_1 be the graph obtained from the graph $D = (V, A)$ of I_{SFC} in the proof of Theorem 1 by introducing new vertices w_i, $i \in [3m]$, and w'_j, $j \in [m]$, replacing each arc a_i with two arcs (u, w_i) and (w_i, v) with $d(u, w_i) = d(w_i, v) = x_i$, and replacing each arc b_j with two arcs (v, w'_j) and (w'_j, u) with $d(v, w'_j) = d(w'_j, u) = B$. Notice that D_1 is a simple and balanced graph, and the underlying graph G_{D_1} of D_1 is bipartite and series-parallel. Also we can prove that $I_{3\text{PART}}$ is a yes-instance of 3-PARTITION if and only if there exists a sequence π of $A(D_1)$ with $p_\pi(V(D_1)) \leq B$, in a similar way to the proof of Theorem 1.

▶ **Theorem 2.** *For a simple digraph D,* Min-Sfc *is strongly NP-hard even if D is balanced, G_D is bipartite, or series-parallel (i.e., the treewidth of G_D is at most two).*

Furthermore, we can show that the problem Min-SFC is NP-hard even in the case of trees, while it is open whether it is strongly NP-hard. The proof is given later in Section 4.3.

▶ **Theorem 3.** *For a simple digraph D,* Min-Sfc *is NP-hard even if G_D is a tree of diameter at most four.*

4 Min-SFC: Tractable Cases

In this section, we show that in some practical cases the problem Min-SFC becomes tractable. We assume in this section that D is a simple digraph, unless otherwise mentioned.

4.1 Uniform Digraphs, Acyclic Digraphs, and Symmetric Graphs

In the case of uniform debt, Min-SFC is equivalent to the problem which asks to partition a given graph into a minimum number of directed paths, which is known to be linearly solvable (e.g., see [3, Lemma 2]).

▶ **Theorem 4.** *If each debt is uniform,* Min-SFC *can be solved in linear time.*

Let $D = (V, A)$ be a digraph and $comp(D)$ be the number of components in D. Let A_δ denote the set of arcs a in A with $d(a) \leq \delta$. We denote $\{d(a) \mid a \in A\}$ by $\{\delta_1, \delta_2, \ldots, \delta_q\}$ with $\delta_1 < \delta_2 < \cdots < \delta_q$. Then, we have the following lemma about lower bounds on $opt(D, d)$.

▶ **Lemma 5.**

(i) *For a digraph $D = (V, A)$, $opt(D, d) \geq \sum_{v \in V} \max\{0, \sum_{a \in A_D^+(v)} d(a) - \sum_{a \in A_D^-(v)} d(a)\}$, where $A_D^+(v)$ (resp., $A_D^-(v)$) denotes the set of all arcs incident from v (resp., to v) in D.*

(ii) *For a digraph $D = (V, A)$, we have $opt(D, d) \geq \sum_{i=1}^{q} comp(D[A \setminus A_{\delta_{i-1}}])(\delta_i - \delta_{i-1})$, where we let $\delta_0 = 0$.*

Assume that D is an acyclic digraph and let $\tau : V \to [n]$ be a topological ordering of V. It is not difficult to see that a sequence π of A such that $\pi(a_1) < \pi(a_2)$ if and only if $\tau(s(a_1)) \leq \tau(s(a_2))$ for each $a_1, a_2 \in A$ satisfies $p_\pi(V) = \sum_{v \in V} \max\{0, \sum_{a \in A_D^+(v)} d(a) - \sum_{a \in A_D^-(v)} d(a)\}$; Min-SFC is linearly solvable by Lemma 5(i).

▶ **Theorem 6.** *For an acyclic digraph D,* Min-SFC *can be solved in linear time.*

Assume that D is a symmetric digraph. Then, we can show that a sequence π of A with $p_\pi(V) = \sum_{i=1}^{q} comp(D[A \setminus A_{\delta_{i-1}}])(\delta_i - \delta_{i-1})$ which composes an Eulerian cycle of D can be found in $O(m^2)$ time.

▶ **Theorem 7.** *For a symmetric digraph D,* Min-SFC *can be solved in $O(m^2)$ time.*

For a tree D, if debts on each vertex is balanced, then debts on each pair of two vertices u and v with $\{u, v\} \in E(G_D)$ become symmetric. Therefore, as a corollary of Theorem 7, we can show that Min-SFC with a tree D is polynomially solvable if D is balanced.

▶ **Corollary 8.** *For a balanced tree,* Min-Sfc *can be solved in $O(n^2)$ time.*

4.2 Stars with General Debts

We next consider the case where the underlying graph of $D = (V, A)$ is a star with arbitrary debts. We remark that the interbank network system in Japan was a kind of star structures before 1997 [7]. Throughout this subsection, we assume that for each pair of vertices v and v' in V with $\{v, v'\} \in E(G_D)$, both of (v, v') and (v', v) belong to A; otherwise (say, $(v, v') \notin A$), then we add an arc (v, v') with debt 0 to D and redenote the resulting graph by D (note that the existence of arcs with debt 0 does not affect to $opt(G, d)$).

Now let $D = (V, A)$ be a star with center u. Then, $E(G_D) = \{\{u, v\} \mid v \in V \setminus \{u\}\}$ holds. Let $V^+ = \{v \in V \setminus \{u\} \mid d(v, u) \geq d(u, v)\}$ and $V^- = \{v \in V \setminus \{u\} \mid d(v, u) < d(u, v)\}$. We have the following theorem about an optimal solution.

▶ **Theorem 9.** *Let $D = (V, A)$ be a star with center u. There exists an optimal sequence π of A for* Min-SFC *satisfying the following* (i)–(iv)*:*

(i) $\pi(u, v) = \pi(v, u) - 1$ *for all* $v \in V \setminus \{u\}$.

(ii) $\pi(u, v) < \pi(u, v')$ *for all* $v \in V^+$ *and* $v' \in V^-$.

(iii) $\pi(u, v) < \pi(u, v')$ *if and only if* $d(u, v) \leq d(u, v')$ *for all* $v, v' \in V^+$.

(iv) $\pi(u, v) < \pi(u, v')$ *if and only if* $d(v, u) \geq d(v', u)$ *for all* $v, v' \in V^-$.

This theorem shows that we can obtain an optimal solution of Min-SFC by the following algorithm MinStar(D, d).

Algorithm MinStar(D, d)
Input: A star $D = (V, A)$ with center u and a debt function d.
Output: A sequence π of A such that $p_\pi(V)$ is minimized.
Step 1: Order vertices of V^+ such that $d(u, v_1) \leq d(u, v_2) \leq \cdots \leq d(u, v_{|V^+|})$ and let $\pi(u, v_i) = 2i - 1$ and $\pi(v_i, u) = 2i$ for $i = 1, 2, \ldots, |V^+|$.
Step 2: Order vertices of V^- such that $d(v_{|V^+|+1}, u) \geq d(v_{|V^+|+2}, u) \geq \cdots \geq d(v_{|V^+|+|V^-|}, u)$ and let $\pi(u, v_i) = 2i - 1$ and $\pi(v_i, u) = 2i$ for $i = |V^+| + 1, |V^+| + 2, \ldots, |V^+| + |V^-|$.

It is fairly straightforward to see that the time complexity of this algorithm is $O(n \log n)$, since it is dominated by that of sorting $O(n)$ arcs.

▶ **Theorem 10.** *For a star,* Min-SFC *can be solved in $O(n \log n)$ time.*

4.3 Trees with General Debts

We consider the case where the underlying graph of D is a tree. As shown in Theorem 3 and Corollary 8, the problem is NP-hard even for trees, while it is polynomially solvable if a given tree is balanced. In this subsection, we will show that Min-SFC is fixed-parameter tractable with respect to the maximum degree Δ, and give a hardness proof of Theorem 3.

Throughout this subsection, we assume that for each pair of vertices v and v' in V with $\{v, v'\} \in E(G_D)$, both of (v, v') and (v', v) belong to A.

A Fixed-parameter Algorithm

We first show the following theorem.

▶ **Theorem 11.** *For a tree D,* Min-SFC *can be solved in $O(2^{\Delta(D)} n \log n)$ time.*

As a corollary of this theorem, we can see that Min-SFC with paths is polynomially solvable.

▶ **Corollary 12.** *For a path,* Min-SFC *can be solved in $O(n \log n)$ time.*

Before proving Theorem 11, we prepare some auxiliary lemmas. For a digraph D, a vertex is called a *leaf* if its degree is one in G_D. For a leaf v, *splitting* v is to introduce a new vertex v' and to replace the arc $(u, v) \in A(D)$ incident to v with an arc (u, v') with debt $d(u, v)$. We denote the resulting digraph and its debt function by $D_{v,v'}$ and $d_{v,v'}$, respectively.

▶ **Lemma 13.** *Assume that a digraph $D = (V, A)$ has a leaf v; denote the two arcs incident on v by (u, v) and (v, u). Let π be a sequence of A with $\pi(u, v) > \pi(v, u)$ and π' be the sequence of $A(D_{v,v'})$ such that $\pi'(u, v') = \pi(u, v)$ and $\pi'(a) = \pi(a)$ for all other arcs $a \in A \setminus \{(u, v)\}$. Then, π' is an optimal sequence of* Min-SFC *for $D_{v,v'}$ if and only if π is an optimal sequence for D under the assumption that (v, u) is cleared before (u, v).*

For a vertex u in D, let D_0 be the star induced by $\{u\} \cup N_D(u)$, and $D_1, D_2, \ldots, D_q$ be subtrees in the graph obtained from D by deleting u, where $q = |N_D(u)|$. We denote two arcs connecting u and D_i by (u, v_i) and (v_i, u), where $v_i \in V(D_i)$. The following lemma shows that if we know in advance whether (u, v_i) is cleared after (v_i, u) or not for each $v_i \in N_D(u)$, then the minimum amount $opt(D, d)$ of funds for clearing $A(D)$ follows from optimal solutions for the star D_0, and either trees $D_i + u$ or $(D_i + u)_{u,u'}$ obtained from $D_i + u$ by splitting u, where for a subgraph D' of D and a vertex $u \in V \setminus V(D')$, we denote $(V(D') \cup \{u\}, A(D') \cup \bigcup_{v \in N_D(u) \cap V(D')} \{(u, v), (v, u)\})$ by $D' + u$.

▶ **Lemma 14.** *For a vertex u in a digraph $D = (V, A)$, let v_i, D_0, D_i, and $D_i + u$, $i = 1, 2, \ldots, q$ be defined as above. Let N_1 and N_2 be a partition of $N_D(u)$ (N_1 or N_2 may be empty). Let $opt(D, d, u, N_1, N_2)$ denote the minimum amount of funds put on V for clearing all arcs in A under the assumption that (v, u) is cleared before (u, v) for each $v \in N_1$ and (u, v) is cleared before (v, u) for each $v \in N_2$. Then,*

$$\begin{aligned} opt(D, d, u, N_1, N_2) \quad = \quad & opt((D_0)_{N_1,N_1'}, d_{N_1}) - \textstyle\sum_{v \in N_1} d(v, u) \\ & - \textstyle\sum_{v \in N_2} \max\{0, d(v, u) - d(u, v)\} \\ & + \textstyle\sum_{v_i \in N_1} (opt(D_i + u, d) - \max\{0, d(u, v_i) - d(v_i, u)\}) \\ & + \textstyle\sum_{v_i \in N_2} (opt((D_i + u)_{u,u'}, d_{u,u'}) - d(u, v_i)), \end{aligned}$$

where $(D_0)_{N_1,N_1'}$ denotes the star obtained from the star D_0 by splitting all vertices in N_1, N_1' denotes the set of vertices generated by these splitting operations, and d_{N_1} denotes the resulting debt function on $A((D_0)_{N_1,N_1'})$.

Proof. Let

$$\begin{aligned} f(D, d, u, N_1, N_2) \quad = \quad & opt((D_0)_{N_1,N_1'}, d_{N_1}) - \textstyle\sum_{v \in N_1} d(v, u) \\ & - \textstyle\sum_{v \in N_2} \max\{0, d(v, u) - d(u, v)\} \\ & + \textstyle\sum_{v_i \in N_1} (opt(D_i + u, d) - \max\{0, d(u, v_i) - d(v_i, u)\}) \\ & + \textstyle\sum_{v_i \in N_2} (opt((D_i + u)_{u,u'}, d_{u,u'}) - d(u, v_i)). \end{aligned}$$

Let π be an arbitrary sequence of A such that $\pi(v, u) < \pi(u, v)$ for each $v \in N_1$ and $\pi(u, v) < \pi(v, u)$ for each $v \in N_2$. First we show that $p_\pi(V) \geq f(D, d, u, N_1, N_2)$, from which $opt(D, d, u, N_1, N_2) \geq f(D, d, u, N_1, N_2)$. We will consider lower bounds L_1, L_2, and L_3 on $p_\pi(u)$, $\sum_{v_i \in N_1} p_\pi(V(D_i))$, and $\sum_{v_i \in N_2} p_\pi(V(D_i))$, respectively; $p_\pi(V) \geq L_1 + L_2 + L_3$.

Consider a lower bound on $p_\pi(u)$. Since how much funds need to be put on u depends only on debts of arcs incident from/to u, we consider the minimum amount p^* of funds for clearing all arcs in the star D_0 with center u. By the assumption that (v_i, u) is cleared before (u, v_i) for each $v_i \in N_1$ and (u, v_i) is cleared before (v_i, u) for each $v_i \in N_2$ and Lemma 13 for leaves $v_i \in N_1$ of D_0, we can see that $p^* = opt((D_0)_{N_1,N_1'}, d_{N_1})$. Now we can observe that any sequence π' of $A((D_0)_{N_1,N_1'})$ satisfies $p_{\pi'}(N_{(D_0)_{N_1,N_1'}}(u)) = \sum_{v \in N_1} d(v, u) +$

$\sum_{v\in N_2}\max\{0, d(v,u)-d(u,v)\}$. Hence, the amount $p_\pi(u)$ of funds put on u is at least $\mathit{opt}((D_0)_{N_1,N_1'}, d_{N_1}) - (\sum_{v\in N_1} d(v,u) + \sum_{v\in N_2}\max\{0, d(v,u)-d(u,v)\})$.

Consider a lower bound on $p_\pi(V(D_i))$ for $v_i \in N_1$. Note that how much funds need to be put on $V(D_i)$ depends on debts of $A(D_i)\cup\{(u,v_i),(v_i,u)\}$. By taking into account the assumption that (v_i,u) is cleared before (u,v_i), we can observe that $p_\pi(V(D_i))$ is at least the minimum amount of funds put on $V(D_i)$ among any funds for clearing $A(D_i+u)$ in D_i+u. Here notice that the amount of funds put on u in D_i+u is always $\max\{0, d(u,v_i)-d(v_i,u)\}$. It follows that $p_\pi(V(D_i)) \geq \mathit{opt}(D_i+u,d) - \max\{0, d(u,v_i)-d(v_i,u)\}$. Similarly, we can observe that for each $v_i \in N_2$, $p_\pi(V(D_i)) \geq \mathit{opt}((D_i+u)_{u,u'}, d_{u,u'}) - d(u,v_i)$ holds.

Thus, we can see that $p_\pi(V) \geq f(D,d,u,N_1,N_2)$. Finally, we show that some sequence π^* of A satisfies $p_{\pi^*}(V) = f(D,d,u,N_1,N_2)$; π^* is optimal and proves this lemma. Let π_0' be a sequence of $A((D_0)_{N_1,N_1'})$ obtained by applying Algorithm MinStar$((D_0)_{N_1,N_1'}, d_{N_1})$, and π_0 be the sequence of $A(D_0)$ obtained from π_0' by letting $\pi_0(u,v) = \pi_0'(u,v')$ for all $v\in N_1$ and $\pi_0(a) = \pi_0'(a)$ for all other arcs a incident on u. For a tree D_i with $v_i \in N_1$, let π_i be a sequence of $A(D_i+u)$ with $p_{\pi_i}(V(D_i+u)) = \mathit{opt}(D_i+u,d)$ with $\pi_i(v_i,u) < \pi_i(u,v_i)$. For a tree D_i with $v_i\in N_2$, let π_i be a sequence of $A((D_i+u)_{u,u'})$ with $p_{\pi_i}(V((D_i+u)_{u,u'})) = \mathit{opt}((D_i+u)_{u,u'}, d_{u,u'})$ with $\pi_i(v_i,u') > \pi_i(u,v_i)$. Note that such a π_i exists for each $v_i \in N_1\cup N_2$. We can construct a sequence π^* of A with $p_{\pi^*}(V) = f(D,d,u,N_1,N_2)$ by combining π_0 and π_i, $i\in N_1\cup N_2$. ◀

Let $D=(V,A)$ be a tree. Based on Lemma 14, we will give a dynamic programming algorithm for finding an optimal sequence of A in $O(2^\Delta n)$ time, which proves Theorem 11.

Here, for a vertex $r\in V$ chosen arbitrarily, we regard D as a rooted tree with root r. For a vertex u in D, let $\mathit{pa}(u)$ be the parent of u if it exists, $Ch(u)$ be the children of u, and $D(u)$ be the subtree of D rooted at u. For a partition $\{N_1,N_2\}$ of $Ch(u)$, we define $\mathit{opt}_1(u,N_1,N_2)$ (resp., $\mathit{opt}_2(u,N_1,N_2)$) as the minimum amount of funds clearing $A(D(u)+\mathit{pa}(u))$ under the assumption that (v,u) is cleared before (u,v) for each $v\in N_1$ (resp., $v\in N_1\cup\{\mathit{pa}(u)\}$) and (u,v) is cleared before (v,u) for each $v\in N_2\cup\{\mathit{pa}(u)\}$ (resp., $v\in N_2$). Note that $\mathit{opt}_1(r,N_1,N_2) = \mathit{opt}_2(r,N_1,N_2)$. Let $\mathit{opt}_i^*(u) = \min\{\mathit{opt}_i(u,N_1,N_2) \mid N_1 \subseteq Ch(u)\}$ for $i=1,2$. We here remark that $\mathit{opt}_1^*(u)$ (resp., $\mathit{opt}_2^*(u)$) is the minimum amount of funds for clearing $A(D(u)+\mathit{pa}(u))$ under the assumption that $(u,\mathit{pa}(u))$ (resp., $(\mathit{pa}(u),u)$) is cleared before $(\mathit{pa}(u),u)$ (resp., $(u,\mathit{pa}(u))$).

Our dynamic programming algorithm proceeds in a bottom-up manner in D, while computing these two values $\mathit{opt}_1^*(u)$ and $\mathit{opt}_2^*(u)$ for each vertex u in D. note that $\mathit{opt}_1^*(r) = \mathit{opt}_2^*(r) = \mathit{opt}(D,d)$. Lemma 14 indicates that $\mathit{opt}_1(u,N_1,N_2)$ and $\mathit{opt}_2(u,N_1,N_2)$ can be computed by using $\mathit{opt}_1^*(v)$ and $\mathit{opt}_2^*(v)$ for $v\in Ch(u)$. Namely, we have

$$\begin{aligned}\mathit{opt}_1(u,N_1,N_2) \quad = \quad & \mathit{opt}((D_0)_{N_1,N_1'}, d_{N_1}) - \textstyle\sum_{v\in N_1} d(v,u) \\ & - \textstyle\sum_{v\in N_2}\max\{0, d(v,u)-d(u,v)\} \\ & + \textstyle\sum_{v\in N_1}(\mathit{opt}_1^*(v) - \max\{0, d(u,v)-d(v,u)\}) \\ & + \textstyle\sum_{v\in N_2}(\mathit{opt}_2^*(v) - d(u,v)),\end{aligned}$$

and

$$\begin{aligned}\mathit{opt}_2(u,N_1,N_2) \quad = \quad & \mathit{opt}((D_0)_{N_1\cup\{\mathit{pa}(u)\},N_1'\cup\{\mathit{pa}'\}}, d_{N_1\cup\{\mathit{pa}(u)\}}) - \textstyle\sum_{v\in N_1} d(v,u) \\ & - \textstyle\sum_{v\in N_2}\max\{0, d(v,u)-d(u,v)\} \\ & + \textstyle\sum_{v\in N_1}(\mathit{opt}_1^*(v) - \max\{0, d(u,v)-d(v,u)\}) \\ & + \textstyle\sum_{v\in N_2}(\mathit{opt}_2^*(v) - d(u,v)),\end{aligned}$$

where $D_0 = D[\{u,\mathit{pa}(u)\}\cup Ch(u)]$, pa' denotes the vertex generated by splitting $\mathit{pa}(u)$ in $(D_0)_{N_1,N_1'}$, and $d_{N_1\cup\{\mathit{pa}(u)\}}$ denotes the debt function on $A((D_0)_{N_1\cup\{\mathit{pa}(u)\},N_1'\cup\{\mathit{pa}'\}})$.

Here we note that in these two equations, $opt_1^*(v) = opt(D(v) + u, d)$ and $opt_2^*(v) = opt((D(v) + u)_{u,u'}, d_{u,u'})$, by the assumption on N_1 and N_2. For stars $(D_0)_{N_1,N_1'}$ and $(D_0)_{N_1 \cup \{pa(u)\}, N_1' \cup \{pa'\}}$, we can compute $opt((D_0)_{N_1,N_1'}, d_{N_1})$ and $opt((D_0)_{N_1 \cup \{pa(u)\}, N_1' \cup \{pa'\}}, d_{N_1 \cup \{pa(u)\}})$ in $O(|N_D(u)| \log |N_D(u)|)$ time by Theorem 10. Hence, if we know $opt_1^*(v)$ and $opt_2^*(v)$ for all $v \in Ch(u)$, then we can compute $opt_1^*(u)$ and $opt_2^*(u)$ in $O(2^{|N_D(u)|}|N_D(u)| \log |N_D(u)|)$ time by computing $opt_1(u, N_1, N_2)$ and $opt_2(u, N_1, N_2)$ for all possible N_1 and N_2. Thus, we can compute $opt(D, d) = opt_1^*(r) = opt_2^*(r)$ in $O(2^{\Delta} n \log n)$ time.

NP-hardness

Next, we give a proof of Theorem 3; we show the NP-hardness of MIN-SFC with a tree. We will reduce from PARTITION, which is known to be NP-hard [8].

PARTITION
Instance: $\{x_1, x_2, \ldots, x_n\}$: A set of n positive integers $x_1, x_2, \ldots, x_n$.
Question: Is there a partition $\{X_1, X_2\}$ of $[n]$ such that $\sum_{i \in X_1} x_i = \sum_{i \in X_2} x_i$?

Take an instance $I_{\text{PART}} = \{x_1, x_2, \ldots, x_n\}$ of PARTITION. From the I_{PART}, we construct an instance $I_{\text{SFC}} = (D = (V, A), d)$ of MIN-SFC as follows. Let $V = \{r, u\} \cup \bigcup_{i=1}^{n} \{v_i, w_i\}$ and $E(G_D) = \{\{r, u\}\} \cup \bigcup_{i=1}^{n} \{\{u, v_i\}, \{v_i, w_i\}\}$. Let $x^* = \sum_{i \in [n]} x_i$, $d(r, u) = d(u, r) = x^*/2$, $d(u, v_i) = d(v_i, u) = d(v_i, w_i) = x_i$, and $d(w_i, v_i) = x_i/2$ for $i \in [n]$.

We here claim that there exists a partition $\{X_1, X_2\}$ of $[n]$ such that $\sum_{i \in X_1} x_i = \sum_{i \in X_2} x_i$ if and only if there exists a sequence π of A with $p_\pi(V) \le 3x^*/4$. Notice that since I_{SFC} can be constructed from I_{PART} in polynomial time, this claim proves Theorem 3.

▶ **Claim 15.** *There exists a partition $\{X_1, X_2\}$ of $[n]$ such that $\sum_{i \in X_1} x_i = \sum_{i \in X_2} x_i$ if and only if there exists a sequence π of A with $p_\pi(V) \le 3x^*/4$.*

5 Max-SFC

5.1 Tractable Case

Assume that D is an acyclic digraph and let $\tau : V \to [n]$ be a topological ordering of V. It is not difficult to see that a sequence π of A such that $\pi(a_1) > \pi(a_2)$ if and only if $\tau(t(a_1)) \le \tau(t(a_2))$ for each $a_1, a_2 \in A$ satisfies $p_\pi(V) = \sum_{a \in A} d(a)$. Thus, MAX-SFC is linearly solvable, since the summation of all debts is an upper bound on the optimal value.

▶ **Theorem 16.** *For an acyclic digraph D,* MAX-SFC *can be solved in linear time.*

5.2 NP-hardness

Below, we show that MAX-SFC is NP-hard, even in the case where each debt is unit or a given graph is bipartite.

▶ **Theorem 17.** *For a digraph D,* MAX-SFC *is NP-hard even if each debt of an arc in $A(D)$ is unit or D is bipartite.*

We prove this theorem by reducing from VERTEX COVER, which is known to be NP-hard [8]. For an undirected graph $G = (V, E)$, a set $V' \subseteq V$ of vertices is called a *vertex cover* if every edge $e = \{u, v\} \in E$ satisfies $\{u, v\} \cap V' \neq \emptyset$.

Vertex Cover
Instance: An undirected graph $G = (V, E)$ and an integer k, that is, $(G = (V, E), k)$.
Question: Is there a vertex cover X with $|X| \leq k$ in G?

Take an instance $I_{\mathrm{VC}} = (G = (V, E), k)$ of Vertex Cover. From the I_{VC}, we construct an instance $I_{\mathrm{SFC}} = (D = (V', A), d)$ of Max-SFC as follows. For each vertex $v_i \in V$, we introduce two copies v_i^1 and v_i^2 of v_i and an arc (v_i^1, v_i^2), and let $V' = \bigcup_{v_i \in V} \{v_i^1, v_i^2\}$ and $A_1 = \bigcup_{v_i \in V} (v_i^1, v_i^2)$. For each edge $\{v_i, v_j\} \in E$, we introduce two arcs (v_i^2, v_j^1) and (v_j^2, v_i^1), and let $A_2 = \bigcup_{\{v_i, v_j\} \in E} \{(v_i^2, v_j^1), (v_j^2, v_i^1)\}$. Let $A = A_1 \cup A_2$ and $d(u, v) = 1$ for all $(u, v) \in A$. Note that D is bipartite. The following lemma completes the proof of Theorem 17.

▶ **Lemma 18.** *G has a vertex cover with cardinality at most k if and only if there exists a sequence π of A such that $p_\pi(V') \geq |A| - k$ in D.*

6 Future work

One of the most important future work is to deal with more appropriate graphs classes that reflects well the debts relationship among banks in our real economic activities. As we mentioned in Section 4.2, it is known that the interbank network system in Japan was a kind of star structures before 1997 [7]. On the contrary, Imakubo and Soejima [7] also showed that in the year of 2005 it had changed and turned to be a core-periphery structure, which is a certain kind of classic hub-authority biclique model [9] and thus one of the so-called complex networks. In the model, banks are classified into either one of the two categories, *core banks* or *periphery* banks, such that payments among the core banks are more densely connected among them compared to those among the periphery banks. Recent research observed similar facts in some other countries, e.g., in the US in 2004 [13], in the Netherlands in 2006 [11], and so on. In view of these recent observations, it would extremely be important to consider our problem on this realistic model and develop efficient algorithms for it.

References

1 Franklin Allen and Douglas Gale. Financial contagion. *Journal of Political Economy*, 108(1):1–33, february 2000.

2 Larry Eisenberg and Thomas H. Noe. Systemic risk in financial systems. *Management Science*, 47(2):236–249, February 2001.

3 John Gallant, David Maier, and James Astorer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50 – 58, 1980.

4 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

5 Hitoshi Hayakawa. Theoretical analyses on settlement system: Chapter 1. complexity of payment network (doctoral dissertation). *Retrieved from UTokyo Repository (No. A30011)*, January 2014.

6 Hitoshi Hayakawa. Characterization of lower bound and upper bound of required settlement fund under real-time gross settlement. *Available at SSRN: http://ssrn.com/abstract=2659975 [Accessed June 1, 2017]*, February 2016.

7 Kei Imakubo and Yutka Soejima. The transaction network in japan's interbank money markets. *Monetary and Economic Studies*, 28:107–150, November 2010.

8 Richard M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

9 Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

10 François Quesnay. *Tableau économique.* Versailles, 1758.

11 Kirsten Bonde Rordam and Morten L. Bech. The topology of danish interbank money flows. *Finance Research Unit No. 2009/01*, 2009.

12 Julio J. Rotemberg. Minimal settlement assets in economies with interconnected financial obligations. *Journal of Money, Credit, and Banking*, 43(1):81–108, February 2011.

13 Kimmo Soramäki, Morten L. Bech, Jeffrey Arnold, Robert J. Glass, and Walter E. Beyeler. The topology of interbank payment flows. *Physica A: Statistical Mechanics and its Applications*, 379(1,i1):317–333, June 2007.

14 World Bank. *Global Financial Development Report 2013 : Rethinking the Role of the State in Finance.* World Bank, Washington, D.C., 2013.

An Efficient Sum Query Algorithm for Distance-based Locally Dominating Functions*

Ziyun Huang[1] **and Jinhui Xu**[2]

1 **Department of Computer Science and Engineering, State University of New York at Buffalo, USA**
`ziyunhua@buffalo.edu`

2 **Department of Computer Science and Engineering, State University of New York at Buffalo, USA**
`jinhui@buffalo.edu`

Abstract

In this paper, we consider the following sum query problem: Given a point set P in $\mathbb{R}^d$, and a distance-based function $f(p,q)$ (*i.e.,* a function of the distance between p and q) satisfying some general properties, the goal is to develop a data structure and a query algorithm for efficiently computing a $(1+\epsilon)$-approximate solution to the sum $\sum_{p\in P} f(p,q)$ for any query point $q \in \mathbb{R}^d$ and any small constant $\epsilon > 0$. Existing techniques for this problem are mainly based on some core-set techniques which often have difficulties to deal with functions with local domination property. Based on several new insights to this problem, we develop in this paper a novel technique to overcome these encountered difficulties. Our algorithm is capable of answering queries with high success probability in time no more than $\tilde{O}_{\epsilon,d}(n^{0.5+c})$, and the underlying data structure can be constructed in $\tilde{O}_{\epsilon,d}(n^{1+c})$ time for any $c > 0$, where the hidden constant has only polynomial dependence on $1/\epsilon$ and d. Our technique is simple and can be easily implemented for practical purpose.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Sum Query, Distance-based Function, Local Domination, High Dimensions, Data Structure

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.47

1 Introduction

In this paper, we consider the following *sum query* problem: Given a set P of points in $\mathbb{R}^d$ (where the dimensionality d could be very high) and a function $f(,)$, the sum query problem is to build a data structure for P so that the sum of $\sum_{p\in P} f(p,q)$ can be efficiently computed or approximated for any query point q in $\mathbb{R}^d$, where $f(p,q)$ is a non-negative *distance-based* function. We say that $f(p,q)$ is *distance-based* if the value of $f(p,q)$ depends only on the distance between p and q. In other words, $f(p,q)$ can be written as $F(\|p,q\|)$ for some non-negative real function $F(\cdot)$.

The distance-based sum query problem are frequently encountered in many applications. A good example is the well known 1-median problem: given a point set P in $\mathbb{R}^d$, find a point q such that the objective value $C(q) = \sum_{p\in P}\|q-p\|$ is minimized. $C(q)$ is clearly an example of the distance-based sum query problem (with respect to the to-be-determined median

* This research was supported in part by NSF through Grants IIS-1422591, CCF-1422324, CNS-1547167, and CCF-1716400.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 47; pp. 47:1–47:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

point q), where each term of the summation is trivially the Euclidean distance $\|q-p\|$ of p and q. The sum query problem also appears in many real world applications. For example, the problem of computing the illumination intensity of a given point can be viewed as a sum query problem. In such an application, the intensity of the query point may jointly be determined by the total amount of light received from multiple light sources. The light contributed by each source is inversely proportional to its squared distance to the given point (*i.e.* obeying the inverse squared distance law in physics). Note that in this case the distance-based functions may be different for each light source, depending on its base intensity. However, if we view a light source with base intensity w as a collection of w light sources with "unit" intensity located at the same place, we may still treat the intensity as a purely distance-based function.

Several previous results are closely related to some versions of the problem considered in this paper. They are mainly based on some *core-set* techniques [2, 7, 10]. In the 1-median problem, for example, a core-set of a point set P in $\mathbb{R}^d$ is a small-size (weighted) subset of P such that for any $q \in \mathbb{R}^d$, the sum $\sum_{p\in P}\|p-q\|$ can be approximated by just inspecting the distances between q and points in the core-set. In general, a core set of P with respect to a function $f(p,q)$ is a small subset of P such that for any q, $\sum_{p\in P} f(p,q)$ can be estimated by using only the information of the points in the core-set. For functions $f(p,q)$ satisfying certain properties, it is possible to construct a core-set for any point set P efficiently [8].

In this paper, we aim to develop an efficient algorithm for supporting distance-based functions that have *local domination property*[6], which means that $f(p,q)$ can be very large when $\|p-q\|$ is small. For example, a distance-based function obeying the inverse squared distance law (*i.e.* $f(p,q) = w/\|p-q\|^2$ for some constant w), is a function having such a property. While the aforementioned core-set method is useful for a large family of functions $f(p,q)$, it does not directly apply to functions which have local domination property. This is because the $\sum_{p\in P} f(p,q)$ could become infinitely large when q approaches any one of the points in P, which means that any "traditional" core-set of P will fail if the core-set is a proper subset of P.

The local domination property imposes additional challenges to the sum query problem. Particularly, it requires the query algorithm to be able to detect points that are close to the query points. This means that the algorithm should have certain ability for proximity search. However, in high dimensional space, highly accurate nearest neighbor search cannot be done very efficiently. Well-known techniques for high dimensional nearest neighbor search, such as the Locality Sensitive Hashing (LSH) [9], require almost linear time to achieve a c-approximate nearest neighbor when c is close to 1 [3]. Thus, for the sum query problem, we are required to develop an estimation algorithm with high accuracy, but not allowed to use the high accuracy proximity search techniques.

To deal with the additional challenge caused by the local domination property, we first assume that the distance function F satisfies the following local domination implied properties.

1. $F(\cdot)$ is positive and $F(0)$ could be infinite.
2. $F(\cdot)$ is monotonously decreasing. [1]
3. For any constant $\lambda \geq 1$, there exists a constant $\Delta(\lambda) \geq 1$, such that $F(x) \leq \Delta(\lambda)F(x\lambda)$ for any $x \geq 0$.

[1] Indeed this restriction can be greatly soften. Our scheme applies as long as $F(\cdot)$ is "not increasing rapidly", *i.e.*, $F(x_1) \leq CF(x_2)$ for some constant C when $x_1 > x_2$. The listed restriction is mainly for ease of presentation.

It is worth noting that although our technique is designed for functions with local domination property, it actually works for any distance-based non-negative functions. Particularly, our approach is capable of solving the “inverse" version of the problem, where $F(\cdot)$ is a monotonically increasing function satisfying some accordingly changed conditions. Since other types of distance-based functions have already been studied in [8], we focus our investigation on locally dominating functions in this paper.

Our Result: Our main result for the sum query problem is a novel scheme based on some sampling and searching techniques, and is capable of reporting a $(1+\epsilon)$-approximation for each sum query ($\sum_{p\in P} f(p,q) = F(\|p-q\|)$) in $\tilde{O}_{\epsilon,d}(n^{0.5+c})$ time with success probability at least $1-1/n$ for any $c>0$. The query algorithm makes use of a soft boundary range reporting data structure to determine a number of points that are among the closest to the query point q. The soft boundary range reporting data structure can be computed within $\tilde{O}_{\epsilon,d}(n^{1+c})$ time for any $c>0$. The hidden constants in the time complexities depend only polynomially on d and $1/\epsilon$. The error factor ϵ can be very small and is assumed to be within the range of $[8/\sqrt{n}, 1)$. One major advantage of our scheme is that the query algorithm runs much faster than the best existing $(1+\epsilon)$-approximate nearest neighbor search technique (which takes almost linear time) in high dimensional space for small enough ϵ .

Our Technique: Our query algorithm consists of 2 main steps. In the first step, we identify a number of points P_Ω that are among the closest to the query point q, and compute directly their contributions to the sum $\sum_{p\in P_\Omega} f(p,q)$. In the second step, we sample, from the rest of the points in P, a small subset of points to estimate their contributions to the sum. Intuitively speaking, since we have already identified a number of points that have the largest contribution to the sum before sampling, the error incurred by sampling the rest of points is relatively small and thus controllable. We combine the results from the 2 steps to obtain an approximate final solution. We use a soft boundary range reporting data structure to identify points that are among the closest to q. With properly chosen parameters, we are able to show that it suffices to use a relatively low quality approximate range search procedure to obtain an accurate solution.

Related Work: As mentioned earlier, the sum query problems can be solved by using core-sets for distance functions satisfying some “nice" properties. Our work can be viewed as a complement to those core-set results as it addresses a rather general case that is hard to use core-sets.

Our scheme makes use of some ideas from range search and top-k indexing. There are a number of previous results on both problems [13, 12, 4, 1]. Many of them are not the best fit, especially in high dimensional space, as they cannot be directly applied to our problem. The special property of our problem enable us to develop a range search scheme with better performance.

2 Query Algorithm by Searching and Sampling

In this section, we present our algorithm for the sum query problem. We start our discussion with a high level description of our ideas.

2.1 Starting Point: Estimation by Sampling

Answering a distance-based sum query for a given query point q is essentially estimating the sum (or equivalently, the mean) of a set of numbers: $\{f(p,q) \mid p \in P\}$. A common practice for efficiently estimating the mean of a set of numbers is using sampling. It is well-known that even for a large set of numbers, it suffices to take only a small sample from the set and calculate the mean of the sampled set. The calculated sum is very likely to be a high quality estimation of the mean value of the whole set. The following lemma is one of the known results on concentration of sample mean.

▶ **Lemma 1** (Hoeffding's Inequality [11]). *Let $X = \{x_1, \ldots, x_n\}$ be a multi-set of n real numbers, and $x'_1, \ldots, x'_m$ be a random sample drawn without replacement from X. Let $a = \min_{1\le i\le n} x_i$ and $b = \max_{1\le i\le n} x_i$. Then, for any $\epsilon > 0$,*

$$\mathbb{P}\Big(|\frac{1}{m}\sum_{i=1}^{n} x'_i - \mu| \ge \epsilon\Big) \le \exp\Big(-\frac{2m\epsilon^2}{(b-a)^2}\Big),$$

where $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the mean of X.

Note that the error bound in the above estimation depends on the spread (*i.e.* the difference between the largest and smallest elements) of the original number set. This implies that a straightforward application of the sampling technique may not be sufficient to achieve highly accurate solution (*i.e.* $(1+\epsilon)$-approximation) of the sum query problem. The error bound ensured by Lemma 1 could be small in terms of the *spread* (with high probability, by setting ϵ to be $\Theta((b-a))$ and $m = \Theta(\log n)$, for example), but still might be large compared to the *mean*. In the distance-based sum query problem where we are essentially estimating the mean of all the addictive terms, the error is evaluated with respect to the mean value $\sum_{p\in P} f(p,q)/n$. If the spread is very large compared to the mean (which could happen if, for example, the query point q is very close to one of the data point), the error (in terms of the mean value) will also be large.

Intuitively, since all additive terms ($f(p,q)$ for all $p \in P$) are nonnegative in the distance-based sum query problem, the largest terms in the sum tends to contribute more to the error incurred by sampling. This leads us to the idea of identifying a few of the largest terms in the sum and considering them separately. To implement this idea for distance-based sum query, we partition the input point set P into 2 subsets, P_O and P_Ω, based on $f(p,q)$ and q, where P_Ω contains the k points in P corresponding to the k largest terms of $\{f(p,q) \mid p \in P\}$ and $k \ll n$ is a factor to be determined later. We then estimate the contributions S_Ω and S_O of P_Ω and P_O, respectively. S_Ω can be computed directly from P_Ω, and S_O is determined from P_O by using standard sampling technique. Thus we can obtain the solution from $S = S_O + S_\Omega$. The estimation process is efficient if $k \ll n$ is sufficiently small. By intuition, this method could achieve better accuracy, since excluding P_Ω from the sampling process avoids the situation that a few very large additive terms exist in the sum, making the sampling technique not applicable.

2.2 Identifying Close Points: Soft Boundary Range Search

Clearly, the aforementioned approach requires that given any query point q, the set P_Ω has to be determined efficiently. Recall the assumption that $f(p,q)$ is a monotonously decreasing function with respect to $\|p-q\|$. This means that the set P_Ω is indeed the subset of P which consists of the k closest points in P to q. To perform this task efficiently, we need to build an k-nearest neighbor data structure for P, which is capable of reporting the k nearest neighbors of q in P for any query point q.

The k-nearest neighbor (kNN) problem in $\mathbb{R}^d$ is known to be hard when d is large due to the curse of dimensionality. If approximation is allowed, there are several techniques, for example the well known Locality Sensitive Hashing (LSH), that are applicable to kNN in arbitrary dimensions. Nonetheless these techniques do not directly provide a solution to our searching problem with the desired performance. When the approximation ratio is small, the nearest neighbor query using LSH takes near linear time in high dimensional space. It seems that it would also be the case for the distance-based sum query problem that the query would be inefficient for small ϵ.

To overcome this obstacle, we make use of a bi-criteria approximation scheme to report P_Ω. For a predefined parameter k and a controlling constant factor $\lambda > 1$, instead of reporting the k approximate nearest neighbors of the query point q, we try to report all points that lie in $B(q, r_O)$, where $B(q, x)$ denotes the closed ball centered at q and with radius x, and $r_O > 0$ satisfies the condition that $|B(q, \lambda r_O) \cap P| = O(k)$. In other words, we report the near neighbors of q in P that lie in a soft boundary that is based on the $O(k)$ nearest neighbor of q. When λ is not very close to 1, the reporting can be performed efficiently using known proximity search techniques (the technical details of the kNN soft boundary range search algorithm will be presented in later sections).

Note that in the above soft boundary range reporting scheme, the controlling factor λ does not depend on ϵ. This avoids the potential issue that it may take near linear time to answer a query when ϵ is small. Later we will show that λ does not need to be close to 1 (*i.e.* the accuracy of the soft boundary search does not need to be high) when ϵ is small. The reason is the follows. If k is small (*e.g.*, $k = O(\sqrt{n})$), the k-nearest neighbors of q in P is only a very small fraction of points in P. Therefore, we are able to afford large error from estimating these points, while still keeping the error of the final solution within the $(1 + \epsilon)$-approximation range.

The remaining problem of this scheme is how to determine the value of r_O efficiently when answering a query. This can be achieved by sampling. Suppose that we sample m points from P where m is a sufficiently large integer. Let P'_s be the sampled point set, and let p_α be the $\lceil mk/n \rceil$-th closest point to q in P'_s. Intuitively, by performing a "scaling" argument, p_α should be approximately the k-th (by $(mk/n) * (n/m) = k$) closest point to q in P. Later we will show that this intuition is correct. We then set $r_O = \|p_\alpha - q\|/\lambda$.

2.3 Algorithm for Sum Query

We summarize the above discussion with the following explanation of the query procedure. Suppose that the controlling factor λ is given, and k is set to be $\lceil \sqrt{n} \rceil$. Note that k is just for analysis purpose and the algorithm does not really depend on it. Let m be the size of the sample and assume that its value has already been provided. To answer a distance-based sum query for a query point q, we first sample a subset P'_s from P with size m. Let p_α be the $\lceil m/\sqrt{n} \rceil$-th closest point to q in P'_s. Then p_α is approximately the $\sqrt{n}$-th closest point to q in P. We choose r_O to be $\|p_\alpha - q\|/\lambda$, and use range search technique to determine the point set $P'_\Omega = B(q, r_O) \cap P$. P'_Ω contains points that are the closest to q. We use sampling to estimate the mean value of $f(p, q)$ for all point $p \in P \setminus P'_\Omega$ without incurring large error. This mean value gives us an estimation of the value $S_O = \sum_{p \in P \setminus P'_\Omega} f(p, q)$. The value $S_\Omega = \sum_{p \in P'_\Omega} f(p, q)$ can be directly computed. The sum of S_Ω and S_O is then an accurate estimation of the distance-based sum $\sum_{p \in P} f(p, q)$.

Below are the main steps of query algorithm, where the approximation factor ϵ satisfies the condition of $4/\sqrt{n} \leq \epsilon < 1/2$ and $n \geq 100$. We assume the existence of a soft boundary range reporting data structure (details of the data structure will be discussed in later section)

Algorithm 1 ComputeSum(q, λ, ϵ)

```
Input: Query point $q$, controlling factor $\lambda > 1$, approximation ratio $\epsilon$
Output: A value $\bar{S}$ which is an approximate value of $S = \sum_{p \in P} f(p, q)$.
Set $\gamma = 262\Delta^2\epsilon^{-2}$. Randomly sample $m = \lceil \gamma\sqrt{n} \ln 4n \rceil$ points from $P$ without replacement. Let $P'_s$ denote the sampled point set.
Let $p_\alpha$ be the $\lceil m/\sqrt{n} \rceil$-th closest point to $q$ in $P'_s$. Let $r_\alpha$ denotes $\|p_\alpha - q\|$. Let $r_O = r_\alpha/\lambda$.
Report points lying inside $B(q, r_O)$ by using the $\lambda$-approximate soft boundary range search data structure. Let $P'_\Omega$ denote the set of reported points.
Compute $S'_\Omega = \sum_{p \in P'_\Omega} f(p)$.
Let $P'_O = P'_s \setminus B(q, r_O)$. Compute $S'_O = \sum_{p \in P'_O} f(p)$
Output $\bar{S} = S_\Omega + nS'_O/|P'_O|$ as the result
```

which can answer the range reporting query made by the algorithm. $\lambda > 1$ is a factor for controlling the accuracy of the soft boundary range reporting data structure. We let Δ denote the constant such that $F(x) \leq \Delta F(x\lambda)$ for any $x \geq 0$, where $F(\cdot)$ is the distance-based function for $f(p, q)$ (*i.e*, $f(p, q) = F(\|p - q\|)$). Since the query point q is given, we write $f(p, q)$ as $f(p)$ for convenience.

2.4 Algorithm Analysis

In this section we prove the correctness of the algorithm and analyze its performance.

For ease of our presentation, we assume that there is no more than one point with exactly the same distance to q. This assumption is actually not needed for our algorithm. Our argument still holds using any tie-break mechanism if multiple points have the same distance to q. For example, we may assign a unique integer label to every point in P and use it as a tie break.

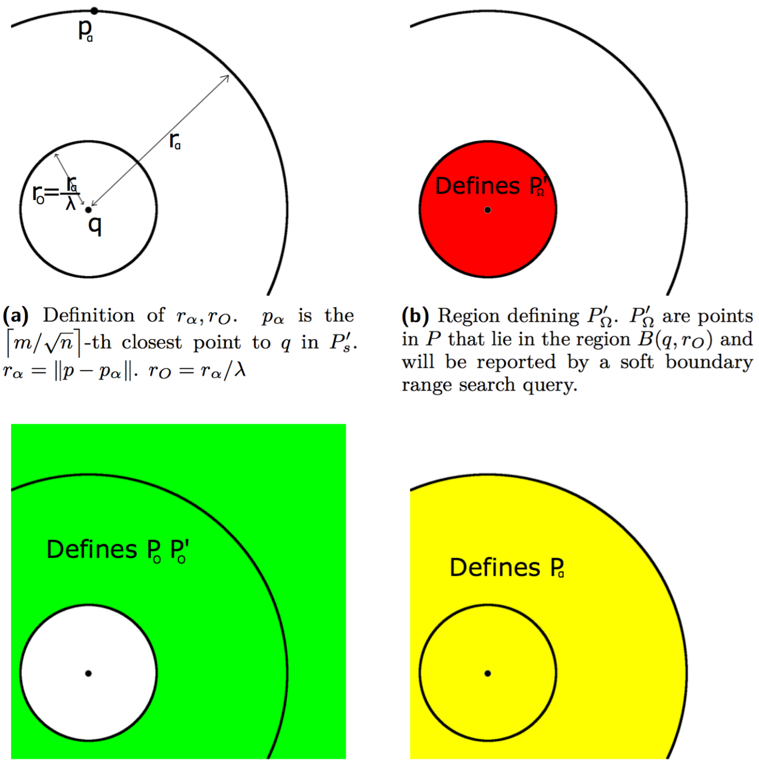
We first present some lemmas that will be used for later analysis. The following is a useful bound for random sample without replacement.

▶ **Lemma 2** (Bernstein's Inequality [5]). *Let $X = \{x_1, \ldots, x_n\}$ be a multi-set of n real numbers, and $x'_1, \ldots, x'_m$ be a random sample drawn without replacement from X. Let $a = \min_{1 \leq i \leq n} x_i$ and $b = \max_{1 \leq i \leq n} x_i$. Let $\sigma = \frac{1}{n}\sum_{x \in X}(x - \mu)^2$ be the variance of X. For any $\epsilon > 0$,*

$$\mathbb{P}\Big(|\frac{1}{m}\sum_{i=1}^{n} x'_i - \mu| \geq \epsilon\Big) \leq \exp\Big(-\frac{m\epsilon^2}{2\sigma^2 + (2/3)(b - a)\epsilon}\Big).$$

▶ **Lemma 3.** *Let X be a set of $n \geq 1$ real numbers, $K \geq 1$, such that for each $x \in X$, $0 \leq x \leq K\sqrt{n}$. Let $\mu = \sum_{x \in X} x/n$ be the mean of X, and $\sigma^2 = \sum_{x \in X}(x - \mu)^2/n$ be the variance. Suppose $\mu \geq 1$. Then $\sigma^2/\mu^2 \leq K\sqrt{n}$.*

Proof. Fixing the value of $\mu \geq 1$, we consider how to construct X so that σ^2 is maximized, subject to the constraint that for each $x \in X$, $0 \leq x \leq K\sqrt{n}$. It is clear that σ^2 is maximized when X is in its most "uneven" state, *i.e.*, with the exception of at most 1 element in X, all other elements are either 0 or $K\sqrt{n}$. In fact, σ^2 can be written as $\sum_{x \in X} x^2/n - \mu^2$. If we can find 2 elements x_1 and x_2 in X, such that $0 < x_1 \leq x_2 < K\sqrt{n}$, increasing x_2 and decreasing x_1 by a same small number will increase the value of $\sum_{x \in X} x^2$ (since $f(x) = x^2$ is convex), while the mean μ of X is unchanged, which means that σ^2 is increased. This proves that when σ^2 is maximized, all elements in X are either 0 or $K\sqrt{n}$ with only 1 exception.



(a) Definition of r_α, r_O. p_α is the $\lceil m/\sqrt{n} \rceil$-th closest point to q in P'_s. $r_\alpha = \|p - p_\alpha\|$. $r_O = r_\alpha/\lambda$

(b) Region defining P'_Ω. P'_Ω are points in P that lie in the region $B(q, r_O)$ and will be reported by a soft boundary range search query.

(c) Region defining $P_O(P'_O)$. $P_O(P'_O)$ are points in $P(P'_s)$ that lie in the region $\mathbb{R}^d \setminus B(q, r_O)$.

(d) Region defining P_α. P_α are points in P that lie in the region $B(q, r_\alpha)$

Figure 1 Illustrations of $r_\alpha, r_O, P'_\Omega, P_O, P'_O, P_\alpha$.

We can easily list elements in such a set X. There exist integer $a \geq 0$ and real number $K\sqrt{n} > b \geq 0$, such that $n\mu = aK\sqrt{n} + b$. Therefore, X contains a elements of value $K\sqrt{n}$, $n - 1 - a$ elements of 0, and the rest of the elements take value b. The term $\sum_{x \in X} x^2$ can be easily computed as $aK^2n + b^2$. Then, we have

$$\sigma^2 = \sum_{x \in X} x^2/n - \mu^2 = (aK^2n + b^2)/n - \mu^2. \tag{1}$$

Note that $aK^2n + b^2 \leq aK^2n + bK\sqrt{n} = \sqrt{n}K(aK\sqrt{n} + b) = \sqrt{n}Kn\mu$. Combine this with the above inequality, we have

$$\sigma^2 = (aK^2n + b^2)/n - \mu^2 \leq \sqrt{n}K\mu - \mu^2. \tag{2}$$

Therefore $\sigma^2/\mu^2 \leq (\sqrt{n}K/\mu) - 1$. Since $\mu \geq 1$, $\sigma^2/\mu^2 \leq (\sqrt{n}K/\mu) - 1 \leq \sqrt{n}K/\mu \leq K\sqrt{n}$. ◀

In the following, we let $P_O = P \setminus B(q, r_O)$. Define $P_\alpha = P \cap B(q, r_\alpha)$. (Figure 1 gives a simple illustration of $r_\alpha, r_O, P'_\Omega, P_O, P'_O, P_\alpha$ for easy understanding of later analysis.)

▶ **Definition 4.** We say that the *Good Sample Condition* is satisfied in a query procedure of Algorithm 1, if all of the following conditions hold.

1. $\sqrt{n}/2 \leq |P_\alpha| \leq 2\sqrt{n}$.
2. $||P'_O|n/m - |P_O|| \leq \epsilon |P_O|$

▶ **Lemma 5.** *With probability at least* $1 - 1/2n$*, the good sample condition satisfies.*

Proof. We first show that $|P_\alpha| \leq 2\sqrt{n}$ happens with probability at least $1 - 1/2n$.

Let P_β denote the set of $\lceil 2\sqrt{n} \rceil$ points in P that are the closest to q. For every $p \in P$, we define $x(p)$ as follows. $x(p) = 1$ if $p \in P_\beta$, and $x(p) = 0$ otherwise. The mean value of $x(p)$ for all $p \in P$ can be easily computed as $\mu = \lceil 2\sqrt{n} \rceil / n$. Let μ' be the mean value of $x(p)$ in the sampled set P'_s. Clearly $\mu' = |P'_s \cap P_\beta|/m$.

Recall that, from the definition of P_α, we know that P_α contains the $\lceil m/\sqrt{n} \rceil$ closest points in P'_s to q but does not include any point in P'_s farther than the $\lceil m/\sqrt{n} \rceil$ closest points. Consider the event that $|P_\alpha| > 2\sqrt{n}$. If it happens that $|P_\alpha| > 2\sqrt{n}$, it implies that the closest $\lceil 2\sqrt{n} \rceil$ points in P to q have no more than $\lceil m/\sqrt{n} \rceil$ points in P'_s, which means that $|P'_s \cap P_\beta| \leq \lceil m/\sqrt{n} \rceil \leq m/\sqrt{n} + 1 \leq 1.01m/\sqrt{n}$ (where the last inequality comes from simple calculation $m/\sqrt{n} = \lceil \gamma\sqrt{n}\ln 4n \rceil / \sqrt{n} \geq \gamma \ln 4n - 1 \geq 262 - 1 = 261$). Therefore we have $\mu' = |P'_s \cap P_\beta|/m \leq 1.01/\sqrt{n}$. From $\mu = \lceil 2\sqrt{n} \rceil / n \geq 2\sqrt{n}/n = 2/\sqrt{n}$, we get $|\mu - \mu'| \geq 0.99/\sqrt{n}$.

Now we bound the probability of the event $|\mu - \mu'| \geq 0.99/\sqrt{n}$ using Lemma 2. Applying Lemma 2 to sample P'_s of P about value $x(p)$, we have

$$\mathbb{P}\Big(|\mu' - \mu| \geq 0.99/\sqrt{n}\Big) \leq \exp\Big(-\frac{m(0.99/\sqrt{n})^2}{2\sigma^2 + (2/3)(0.99/\sqrt{n})}\Big),$$

where $\sigma^2 = \sum_{p \in P}(x(p) - \mu)^2/n$. It is straightforward to calculate $\sigma^2 = 2(\lceil 2\sqrt{n} \rceil/n)(1 - \lceil 2\sqrt{n} \rceil/n)$. Thus, we have $\sigma^2 \leq 2(\lceil 2\sqrt{n} \rceil/n) \leq 5\sqrt{n}/n$ (estimation from the assumption that $n \geq 100$). Therefore, we know that

$$\exp\Big(-\frac{m(0.99/\sqrt{n})^2}{2\sigma^2 + (2/3)(0.99/\sqrt{n})}\Big) \leq \exp\Big(-\frac{m(0.99/\sqrt{n})^2}{10/\sqrt{n} + (2/3)(0.99/\sqrt{n})}\Big).$$

The right hand side becomes $\exp(-(m/\sqrt{n})(0.99)^2/10.66)$. Note that $m = \lceil 262\Delta^2\epsilon^{-2}\sqrt{n}\ln 4n \rceil \geq 262\sqrt{n}\ln 4n$. By simple calculation, we have $(m/\sqrt{n})(0.99)^2/10.66 \geq \ln 4n$. As a result, we know that

$$\mathbb{P}\Big(|\mu' - \mu| \geq 0.99/\sqrt{n}\Big) \leq e^{-\ln 4n} = 1/4n.$$

Since we have already shown that $|P_\alpha| > 2\sqrt{n}$ implies $|\mu - \mu'| \geq 0.99/\sqrt{n}$, we know that $|P_\alpha| > 2\sqrt{n}$ may also happen with probability at most $1/4n$.

Using the same argument we can also prove that the event $|P_\alpha| < \sqrt{n}/2$ happens with probability at most $1/4n$. We omit the proof for this case due to similarity with the above case. To summarize, we have proved that Condition 1 of the lemma, $\sqrt{n}/2 \leq |P_\alpha| \leq 2\sqrt{n}$, holds with probability at least $1 - 1/2n$.

For the second condition, *i.e.* $||P'_O|n/m - |P_O|| \leq \epsilon|P_O|$, we will show that it follows from Condition 1.

From definition, we know that $P'_O \supseteq P'_s \setminus B(q, r_\alpha)$. Thus, $|P'_O| \geq |P'_s \setminus B(q, r_\alpha)| = m - \lceil m/\sqrt{n} \rceil$. Clearly we also have $|P'_O| \leq m$. Therefore, we get $n - n\lceil m/\sqrt{n} \rceil/m \leq |P'_O|n/m \leq n$. It is easy to have an estimation $\lceil m/\sqrt{n} \rceil \leq m/\sqrt{n} + 1 \leq 1.01m/\sqrt{n}$. Thus we obtain $n - 1.01\sqrt{n} \leq |P'_O|n/m \leq n$

By Condition 1 and $P_O \supseteq P \setminus P_\alpha$, we have $|P_O| \geq n - |P_\alpha| \geq n - 2\sqrt{n}$. Note that we also clearly have $|P_O| \leq n$. As a result, it follows that $||P'_O|n/m - |P_O|| \leq (1.01 + 2)\sqrt{n} = 3.01\sqrt{n}$.

Now we need to prove that $3.01\sqrt{n} \leq \epsilon|P_O|$. From Condition 1, we know that $|P_O| \geq n - 2\sqrt{n}$. It suffices to show that $3.01\sqrt{n} \leq \epsilon(n - 2\sqrt{n})$. Indeed, this trivially follows from the assumption that $\epsilon \geq 4/\sqrt{n}$ and $n \geq 100$. ◀

Below is an important lemma which shows that our sampling scheme gives a good approximation of the mean value of $f(p)$ for all $p \in P_O$.

▶ **Lemma 6.** *Let μ'_O be the mean of $f(p)$ for all $p \in P'_O$. Let μ_O be the mean of $f(p)$ for all $p \in P_O$. Assume that the good sample condition holds. With probability at least $1 - 1/4n$, $|\mu'_O - \mu_O| \leq \epsilon S/n$*

Proof. Denote $f_*(p) = (\sqrt{n} f(p) + F(r_O))/F(r_O)$ for all $p \in P_O$. Let μ'_* be the mean of $f_*(p)$ for all $p \in P'_O$, and μ_* be the mean of $f_*(p)$ for all $p \in P_O$. Below we first show that

$$\mathbb{P}\Big(|\mu'_* - \mu_*| \geq \epsilon\mu_*/4\Delta\Big) \leq 1/4n. \tag{3}$$

We apply Lemma 2 to bound the probability of the event $|\mu'_*$ - $\mu_*| \geq \epsilon\mu_*/4\Delta$ as follows. The set P'_O can be viewed as a random sample without replacement of size $|P'_O|$ from set P_O, since for a fixed r_O, every $|P'_O|$-subset of P_O has equal probability to be the first $|P'_O|$ points in P'_s, sorted by decreasing order of distances to q. (Note that this fact is true regardless whether the sample satisfies the good sample condition.) Note that for any $p \in P_O$, it is easy to see that $f_*(p) \geq 1$ and $f_*(p) \leq \sqrt{n} + 1$ (since, by $\|q - p\| \leq r_O$, we have $F(r_O) \geq F(\|q - p\|)$). Let $\sigma^2 = (\sum_{p \in P_O}(f_*(p) - \mu_*)^2)/|P_O|$. From Lemma 2, we have

$$\mathbb{P}\Big(|\mu'_* - \mu_*| \geq \epsilon\mu_*/4\Delta\Big) \leq \exp\Big(-\frac{(1/16)|P'_O|\epsilon^2\Delta^{-2}\mu_*^2}{2\sigma^2 + (2/3)\sqrt{n}(\epsilon\mu_*/4\Delta)}\Big). \tag{4}$$

Let

$$\xi = \frac{(1/16)\mu_*^2}{2\sigma^2 + (2/3)\sqrt{n}(\epsilon\mu_*/4\Delta)}.$$

The right hand side of the above inequality (4) becomes $e^{-|P'_O|\epsilon^2\Delta^{-2}\xi}$.

To estimate ξ, we first bound σ^2/μ_*^2. From the good sample condition, we know that $||P'_O|n/m - |P_O|| \leq \epsilon|P_O|$. Thus, we have $|P_O| \geq |P'_O|n/(1+\epsilon)m$. Also, by the definition of P'_O, we know that $P'_s \setminus P_\alpha \subseteq P'_O$. Thus, we get $|P'_O| \geq |P'_s \setminus P_\alpha| \geq m - m/\sqrt{n} - 1$. Therefore, we obtain $|P_O| \geq n\frac{m - m/\sqrt{n} - 1}{(1+\epsilon)m} = n\frac{1 - 1/\sqrt{n} - 1/m}{(1+\epsilon)}$. Since $\epsilon < 1/2$, $m \geq 100$, and $n \geq 100$, we have a rough estimation of $|P_O| \geq n/4$. Also, we know that for any $p \in P_O$, $f_*(p) \leq \sqrt{n} + 1 \leq 2\sqrt{n}$. Consequently, we have $f_*(p) \leq 4\sqrt{|P_O|}$ for every $p \in P_O$. Applying Lemma 3, we know that $\sigma^2/\mu_*^2 \leq 4\sqrt{|P_O|}$. Thus, we get $\sigma^2/\mu_*^2 \leq 4\sqrt{n}$.

Next we show a lower bound for $|P'_O|$. In fact, we know that $|P'_O| = \gamma\sqrt{n}\ln 4n - |P'_s \cap B(q, r_O)| \geq \gamma\sqrt{n}\ln 4n - |P'_s \cap B(q, r_\alpha)| \geq \gamma\sqrt{n}\ln 4n - \gamma\ln 4n - 1 \geq (\gamma\sqrt{n}\ln 4n)/2$ (the last inequality can be easily obtained from the assumption of $n \geq 100$).

Now we estimate $\xi = (32\sigma^2/\mu_*^2 + (8/3)\sqrt{n}\epsilon/(\mu_*\Delta))^{-1}$. By $\sigma^2/\mu_*^2 \leq 4\sqrt{n}$, $\Delta \geq 1$ and $\mu_* \geq 1$, we have $\xi \geq (128\sqrt{n} + (8/3)\sqrt{n})^{-1} \geq (131\sqrt{n})^{-1}$. Then we immediately have $e^{-|P'_O|\epsilon^2\Delta^{-2}\xi} \leq e^{-\gamma\sqrt{n}\epsilon^2\Delta^{-2}\xi/2} \leq e^{-262\epsilon^{-2}\Delta^2\sqrt{n}\epsilon^2\Delta^{-2}(131\sqrt{n})^{-1}/2} \leq e^{-\ln 4n} = 1/4n$. Inequality (3) then follows from this and inequality (4).

Below we show that, $|\mu'_O - \mu_O| > \epsilon S/n$ implies that $|\mu'_* - \mu_*| \geq \epsilon\mu_*/4\Delta$. If this is the case, by inequality (3), we will know that the latter event happens with probability no more than $1/4n$, which also implies that the former event happens with probability no more than $1/4n$, and thus the lemma follows. We will prove the claim by showing that $|\mu'_* - \mu_*| < \epsilon\mu_*/4\Delta$ implies that $|\mu'_O - \mu_O| \leq \epsilon S/n$.

Assume that $|\mu'_* - \mu_*| < \epsilon\mu_*/4\Delta$. Multiplying each side of this inequality by a factor of $F(r_O)/\sqrt{n}$, we have

$$|(\sum_{p\in P'_O}(f(p)+F(r_O)/\sqrt{n})/|P'_O|) - (\sum_{p\in P_O}(f(p)+F(r_O)/\sqrt{n})/|P_O|)| \leq \epsilon((\sum_{p\in P_O} f(p))/|P_O| + F(r_O)/\sqrt{n})/4\Delta. \tag{5}$$

Rearranging the terms gives us

$$|\mu'_O - \mu_O| \leq \epsilon(\mu_O + F(r_O)/\sqrt{n})/4\Delta. \tag{6}$$

In the following, we will obtain an upper bound on $(\mu_O + F(r_O)/\sqrt{n})/4\Delta$ in terms of S.

We first consider $\mu_O/4\Delta$. For each $p \in P \setminus P_O$, since $\|p-q\| \leq r_O$, we have $f(p) \geq f(p')$ for any $p' \in P_O$. Therefore we have $\sum_{p\in P\setminus P_O} f(p)/|P\setminus P_O| \geq \sum_{p'\in P_O} f(p')/|P_O| = \mu_O$. Note that S/n is the mean value of $f(p)$ for all $p \in P$. Thus we have $S/n \geq \min(\mu_O, \sum_{p\in P\setminus P_O} f(p)/|P\setminus P_O|)$. Hence we get $S/2n \geq \mu_O/2 \geq \mu_O/4\Delta$.

Now we bound $F(r_O)/4\sqrt{n}\Delta$ in terms of S. By the fact that $r_O = r_\alpha/\lambda$, we have $F(r_O) \leq F(r_\alpha)\Delta$. Thus we know that $F(r_O)/4\sqrt{n}\Delta \leq F(r_\alpha)/4\sqrt{n}$. From the good sample condition, we have $|P_\alpha| \geq \sqrt{n}/2$. For each $p \in P_\alpha = P \cap B(q, r_\alpha)$, since $\|p-q\| \leq r_\alpha$, it follows that $f(p) \geq F(r_\alpha)$. Therefore we get $S = \sum_{p\in P} f(p) \geq \sum_{p\in P_\alpha} f(p) \geq F(r_\alpha)\sqrt{n}/2$. It then follows that $F(r_O)/4\sqrt{n}\Delta \leq F(r_\alpha)/4\sqrt{n} \leq S/2n$.

Combining the above results and recall (6), we obtain $|\mu'_O - \mu_O| \leq \epsilon S/n$.

To summarize, we have showed that $|\mu'_* - \mu_*| < \epsilon\mu_*/4\Delta$ implies $|\mu'_O - \mu_O| \leq \epsilon S/n$, which means that $|\mu'_O - \mu_O| > \epsilon S/n$ implies $|\mu'_* - \mu_*| \geq \epsilon\mu_*/4\Delta$. This completes the proof. ◀

Below is a result for soft boundary range search. The details of the method will be discussed in next section.

▶ **Lemma 7.** *For any $\lambda > 1$ and $\tau > 0$, there exists a λ-approximate soft boundary range search data structure that can be built in time $O(dn^{1+1/2\lambda+\tau})$. Each query, provided that the good sample condition is satisfied,*

1. *reports all the points in $P_\Omega = P \cap B(q, r_\alpha/\lambda)$ in Algorithm 1 (*i.e. *$P'_\Omega = P_\Omega$) with probability at least $1 - 1/4n$;*
2. *takes time $O(dn^{1/2\lambda+1/2+\tau})$.*

Finally, we have the main theorem for our algorithm.

▶ **Theorem 8.** *With probability at least $1 - 1/n$, $\bar{S}$ produced by Algorithm 1 satisfies the inequality $|\bar{S} - S| \leq 2\epsilon S$.*

Proof. In the following argument, we assume that the good sample condition is satisfied.

P can be partitioned into 2 subsets according to their distances to q: $P = P_O \cup P_\Omega$, where $P_O = P \setminus B(q, r_O)$ (as defined before), and $P_\Omega = P \cap B(q, r_\alpha/\lambda) = P \setminus P_O$.

First, we show that the value $S'_O = \mu'_O|P'_O|n/m$ is a good approximation of $\sum_{p\in P_O} f(p)$. By Lemma 6, we know that with probability at least $1 - 1/4n$, $|\mu'_O - \mu_O| \leq \epsilon S/n$. Thus, we get $|\mu'_O|P'_O|n/m - \mu_O|P'_O|n/m| \leq \epsilon S|P'_O|/m \leq \epsilon S$. By the good sample condition, we have $||P'_O|n/m - |P_O|| \leq \epsilon|P_O|$. Since $P_O \subseteq P$, clearly we know that $|P_O|\mu_O = \sum_{p\in P_O} f(p) \leq S$. Thus, we have $|\mu_O|P'_O|n/m - \mu_O|P_O|| \leq \epsilon\mu_O|P_O| \leq \epsilon S$.

$$
\begin{aligned}
|S'_O - \sum_{p \in P_O} f(p)| &= |\mu'_O|P'_O|n/m - \mu_O|P_O|| & (7)\\
&= |\mu'_O|P'_O|n/m - \mu_O|P'_O|n/m + \mu_O|P'_O|n/m - \mu_O|P_O|| & (8)\\
&\le |\mu'_O|P'_O|n/m - \mu_O|P'_O|n/m| + |\mu_O|P'_O|n/m - \mu_O|P_O|| & (9)\\
&\le \epsilon S + \epsilon S = 2\epsilon S. & (10)
\end{aligned}
$$

For P_Ω, by Lemma 7, we know that this set is identical to P_Ω with probability at least $1 - 1/4n$. Thus we have $S'_\Omega = \sum_{p \in P_\Omega} f(p)$ with probability at least $1 - 1/4n$.

From the above results, we immediately know that when the good sample condition is satified, $S - \bar{S} \le 2\epsilon S$ with probability at least $1 - 1/2n$. Since the good sample condition holds with probability at least $1 - 1/2n$, the theorem then follows. ◀

3 Soft Boundary Range Reporting using Approximate Nearest Neighbor Search

In this section we present a method to report points in $P_\Omega = P \cap B(q, r_\alpha/\lambda)$. We assume that $\sqrt{n}/2 \le |P_\alpha| \le 2\sqrt{n}$, which is a part of the good sample condition.

We reduce the range search query to a number of nearest neighbor queries. Observe that, since $\sqrt{n}/2 \le |P_\alpha| \le 2\sqrt{n}$, if we take a sample Q of $\lceil \sqrt{n}/2 \rceil$ points from P uniformly and independently, with at least constant probability, Q and P_α share exactly 1 common point.

▶ **Lemma 9.** *For $n \ge 100$, the probability of the event that $|P_\alpha \cap Q| = 1$ happens with probability at least $\rho = 1/60$*

Proof. The probability that the event happens can be computed as follows.

$$\mathbb{P}(|P_\alpha \cap Q| = 1) = \lceil \sqrt{n}/2 \rceil \frac{|P_\alpha|}{n}(1 - \frac{P_\alpha}{n})^{\lceil \sqrt{n}/2 \rceil - 1}.$$

Since $|P_\alpha| \ge \sqrt{n}/2$, we have $\lceil \sqrt{n}/2 \rceil \frac{|P_\alpha|}{n} (\ge \sqrt{n}/3) \frac{|P_\alpha|}{n} \ge 1/6$.

Also, from $|P_\alpha| \le 2\sqrt{n}$, we get $(1 - \frac{P_\alpha}{n})^{\lceil \sqrt{n}/2 \rceil - 1} \ge (1 - \frac{P_\alpha}{n})^{\sqrt{n}} \ge (1 - 2/\sqrt{n})^{\sqrt{n}}$. It is well known that $f(x) = (1 - 1/x)^x$ is monotonously increasing for $x > 1$. Thus $(1 - 2/\sqrt{n})^{\sqrt{n}} \ge (1 - 1/5)^{10} > 1/10$. Combining this with the above inequality, the lemma follows. ◀

If Q and P_α share exactly 1 common point, clearly every point in P_α have the same probability to be the common point. There are at most $2\sqrt{n}$ points in P_α; therefore every point has a probability at least $\rho/2\sqrt{n}$ to be the only common point of Q and P_α. Similar observations are used in some other range search techniques[4].

Now, if we are allowed to perform a λ-approximate nearest neighbor search on Q, and if a point $p \in P_\Omega \subseteq P_\alpha$ happens to be the only common point of P_α and Q, then the approximate nearest neighbor search will output p. This is because any other point in Q must be in $P \setminus P_\alpha$, and thus their distance to q must be larger than $r_\alpha = \lambda r_O \ge \|p - q\|$, which means that p is the only λ-approximate nearest neighbor of q in Q.

Therefore, if we sample a point set Q as stated above, and build a nearest neighbor data structure that is able to output a λ-approximate nearest neighbor Q for any q, with at least constant success probability $\delta > 0$(*e.g.* using technique in [9]), then for any $p \in P_\Omega$, this data structure will be able to discover p with probability at least $(\delta\rho/2\sqrt{n})$.

This suggest us to build a range search data structure in the following way. For some number t, independently create t samples $Q_1, \dots, Q_t$, each one has size $\lceil \sqrt{n}/2 \rceil$, by sampling uniformly and independently from P. Then we build an λ-approximate nearest neighbor data structure with success query probability δ for each of $Q_1, \dots, Q_t$. For the reporting query, given q and r_α, we perform an λ-approximate nearest neighbor.

We set t so that $(1 - \delta\rho/2\sqrt{n})^t$ (*i.e.* the probability that a certain point $p \in P_\Omega$ is not reported) is less than $1/8\sqrt{n}n$. By simple calculation, we know that it is possible to find such a t that satisfies the condition $t = O(\sqrt{n}\log n)$. Since P_Ω contains no more than $2\sqrt{n}$ points, it means that if we perform a nearest neighbor search for all $Q_1, \dots, Q_t$, with probability at least $1 - 1/4n$, we are able to output all points in P_Ω.

Note that using this scheme, for each range reporting query, we are required to perform $t = O(\sqrt{n}\log n)$ times nearest neighbor search queries. For any $\tau > 0$, it is possible to build a nearest neighbor data structure that answer each nearest neighbor search query in $O(d(\sqrt{n})^{1/\lambda+\tau})$ time[9]. Therefore the time required for a reporting process is $O(td(\sqrt{n})^{1/\lambda+\tau})$. By $t = O(\sqrt{n}\log n)$, we know that for any $\tau' > 0$, it is possible to perform the range reporting operation in time $O(d(\sqrt{n})^{1+1/\lambda+\tau'})$. For the construction time, each nearest neighbor data structure is built on a $O(\sqrt{n})$ point set, which can be built within time $O(d(\sqrt{n})^{1+1/\lambda+\tau})$ for any $\tau > 0$, Therefore, the total construction complexity is $(dn^{1+1/2\lambda+\tau'})$ for any $\tau' > 0$. The bounds for Lemma 7 are proved.

References

1 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186. SIAM, 2009.

2 Pankaj K. Agarwal, Sariel Har-peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry, MSRI*, pages 1–30. University Press, 2005.

3 Alexandr Andoni, Piotr Indyk, Huy L Nguyễn, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1028. SIAM, 2014.

4 Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM Journal on Computing*, 38(3):899–921, 2008.

5 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

6 D. Z. Chen, Z. Huang, Y. Liu, and J. Xu. On clustering induced voronoi diagrams. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 390–399, 2013.

7 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

8 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578. ACM, 2011.

9 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

10 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300. ACM, 2004.

11 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

12 Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 277–288. ACM, 2016.

13 Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 121–130. ACM, 2012.

Complexity of the Multi-Service Center Problem*

Takehiro Ito[1], Naonori Kakimura[2], and Yusuke Kobayashi[3]

1 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
`takehiro@ecei.tohoku.ac.jp`

2 **Department of Mathematics, Keio University, Yokohama, Japan**
`kakimura@math.keio.ac.jp`

3 **Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Japan.**
`kobayashi@sk.tsukuba.ac.jp`

Abstract

The multi-service center problem is a variant of facility location problems. In the problem, we consider locating p facilities on a graph, each of which provides distinct service required by all vertices. Each vertex incurs the cost determined by the sum of the weighted distances to the p facilities. The aim of the problem is to minimize the maximum cost among all vertices. This problem is known to be NP-hard for general graphs, while it is solvable in polynomial time when p is a fixed constant. In this paper, we give sharp analyses for the complexity of the problem from the viewpoint of graph classes and weights on vertices. We first propose a polynomial-time algorithm for trees when p is a part of input. In contrast, we prove that the problem becomes strongly NP-hard even for cycles. We also show that when vertices are allowed to have negative weights, the problem becomes NP-hard for paths of only three vertices and strongly NP-hard for stars.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases facility location, graph algorithm, multi-service location

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.48

1 Introduction

Facility location is one of the most well-studied topics in combinatorial optimization. There are various kinds of settings depending on the situations. (See e.g., [4].) Generally, in facility location problems, we are given a set of clients and a set of facilities in a graph, and we aim to decide which facilities are open to satisfy the demand of the clients. For example, the well-known *k-center problem* is to place k facilities in a graph so that the maximum distance from each client to their closest facility is minimized [6, 7]. Note that this standard situation assumes that all k facilities can provide the same service so that each client meets their demand by only accessing one facility.

Yu and Li [10] recently proposed a new framework of facility location problems, called *multi-service location problems*, motivated by the situation where each facility provides different services and each client needs to access all facilities to meet their demand. As the first problem of this kind, they proposed the p-SERVICE CENTER problem defined as follows. (The formal definition will be given in Section 2.) In the problem, we assume that clients

* This work is partially supported by JST ERATO Grant Number JPMJER1201, JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP16H03118, JP16K00004, JP16K16010 and JP17K00028, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 48; pp. 48:1–48:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are all vertices in a graph G, and facilities can be located on any place in G even on an edge. When we locate p facilities, each of which provides distinct service, the cost of each client v is determined by the sum of the weighted distances to the p facilities, where the weighted distance from v to a facility x is the shortest-path distance from v to x multiplied by a positive weight (representing the *demand*) of v to the service provided by x. The aim of the problem is to find a location of p facilities that minimizes the maximum cost among the clients.

Yu and Li [10] studied the computational complexity of p-SERVICE CENTER for several cases. They designed a polynomial-time algorithm for general graphs when p is a fixed constant, and an $O(n \log n)$-time algorithm for trees having n vertices when restricted to $p = 2$. On the negative side, they showed that the problem is NP-hard for general graphs when p is a part of input. Anzai et al. [1] showed that this case remains NP-hard even for split graphs with identical edge-length.

In this paper, we consider a simple generalization of p-SERVICE CENTER, that is, each client can have zero or negative weights (demands) to a facility; recall that the weight must be positive in the original setting. This generalization is very simple, but enables us to express several natural situations: a zero demand means that the client does not need the service provided by the facility, while a negative demand means that the client refuses the service provided by the facility; furthermore, any vertex can be a non-client by setting all demands to be zero. In this paper, we sharply analyze the computational complexity of this generalized problem from the viewpoint of graph classes and weights of vertices. Our main contributions are summarized as follows:

(1) The problem with nonnegative weights is solvable in polynomial time for trees, even when the number p of facilities is a part of input.

(2) The problem with nonnegative weights is strongly NP-hard for cycles with identical edge-length. Thus, the problem cannot be solved in pseudo-polynomial time even for a cycle unless P = NP.

(3) When clients are allowed to have negative weights, the problem becomes NP-hard even for paths of only three vertices and strongly NP-hard for stars.

Thus, the problem is polynomially solvable only for trees with nonnegative weights, and is computationally intractable even for a bit larger graph class or negative weights. Let us remark that, while both of the algorithms by Yu and Li [10] require that the number p of facilities is a fixed constant, our algorithm in (1) allows to have p as a part of input.

The rest of the paper is organized as follows. In Section 2, we give a formal definition of the problem studied in this paper. In Section 3, we present a polynomial-time algorithm on a tree. Section 4 is devoted to showing the hardness results.

2 Problem Definition

In this section, we formally define the problem studied in this paper.

Let $G = (V, E)$ be an undirected connected graph. For a subgraph H of G, we sometimes denote by $V(H)$ and $E(H)$ the vertex set and edge set of H, respectively. Assume that each edge $e \in E$ has a length $\ell_e \in \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of all nonnegative real numbers. We may assume that all vertices in G are clients, and each facility can be located on any place in G, even on an edge. We will refer to interior locations on an edge $e \in E$ by their distances along e from its two endpoints. Throughout the paper, a *point* on G indicates either a vertex in V or an interior location on an edge in E. For notational convenience, we sometimes denote simply by G the set of all points on the graph. For two points $x, y \in G$, let $\mathsf{dist}(x, y)$ denote the shortest-path length between x and y.

Let I be the set of facilities. Then, a *location* of I on a graph $G = (V, E)$ is a tuple X of $|I|$ points on G (which are not necessarily distinct). We denote by G^I the family of all the locations of I on G. Suppose that each vertex $v \in V$ has a weight $w_{v,i} \in \mathbb{R}$ for a facility $i \in I$, where $\mathbb{R}$ is the set of all real numbers; the weight $w_{v,i}$ represents the demand of v to the service provided by $i \in I$. For each vertex $v \in V$ and a location $X \in G^I$, the *cost* $\mathsf{cost}(v, X)$ of v to receive the service from X is defined as follows:

$$\mathsf{cost}(v, X) := \sum_{i \in I} w_{v,i} \cdot \mathsf{dist}(v, x_i),$$

where x_i denotes the point on G at which the facility $i \in I$ is placed by X. In this paper, we study the following problem:

The MULTI-SERVICE CENTER problem
Instance. A graph $G = (V, E)$, an edge length $\ell_e \in \mathbb{R}_{\geq 0}$ for $e \in E$, a set I of facilities, and a weight $w_{v,i} \in \mathbb{R}$ for $v \in V$ and $i \in I$.
Question. Find a location X of I on G that minimizes $\max_{v \in V} \mathsf{cost}(v, X)$.

We call the problem p-SERVICE CENTER if the number p of facilities is a fixed constant. In addition, we sometimes write the name of the problem with its restriction: For example, the problem is called MULTI-SERVICE CENTER *with nonnegative weights* if all weights $w_{v,i}$ are nonnegative for $v \in V$ and $i \in I$.

3 Polynomial-Time Algorithm for Trees with Nonnegative Weights

Recall that Yu and Li [10] showed that p-SERVICE CENTER with positive weights is solvable in polynomial time for general graphs, and 2-SERVICE CENTER with positive weights is solvable in $O(n \log n)$ time for trees having n vertices. Both of the algorithms require that the number p of facilities is fixed. In this section, we prove that MULTI-SERVICE CENTER with *nonnegative* weights is solvable in polynomial time for trees even when the number p of facilities is taken as a part of input, as in the following theorem.

▶ **Theorem 1.** MULTI-SERVICE CENTER *with nonnegative weights can be solved in polynomial time for trees.*

In the remainder of this section, we prove Theorem 1. For notational convenience, we may assume that each edge of a given tree has a positive length; this assumption does not lose the generality because we simply regard each edge e with $\ell_e = 0$ as having a sufficiently small positive length.

3.1 Technical highlights

We first explain our main ideas and proof techniques briefly.

To describe a polynomial-time algorithm for trees, let us first consider the case when a graph is a path. In this case, it is not difficult to see that the problem can be reduced to a linear programming problem. In fact, we can identify a point on the path with a 1-dimensional coordinate x by taking one of the end of the path as the origin. Then, the distance from each client to x can be expressed by an absolute value function with respect to x. Therefore, MULTI-SERVICE CENTER for a path is equivalent to minimizing the maximum of the sum of absolute value functions with nonnegative coefficients, which can be formulated as a linear programming problem.

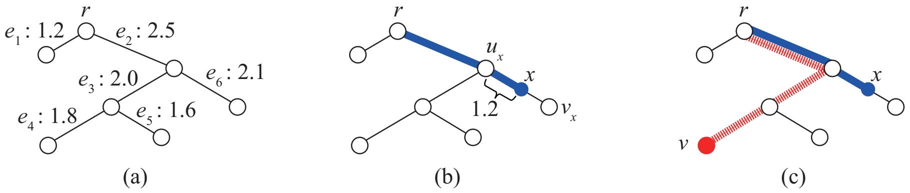


Figure 1 (a) A tree with edge-lengths, (b) the admissible vector $\tilde{x} = (\tilde{x}(e_1), \ldots, \tilde{x}(e_6))^\top = (0, 2.5, 0, 0, 0, 1.2)^\top$ representing a point x, and (c) the admissible vector $\tilde{v} = (0, 2.5, 2.0, 1.8, 0, 0)^\top$.

In order to extend the above observation to the tree case, we identify a point on a tree with a path from a specified vertex (a root). Then, we can represent a point on the tree by a vector in the m-dimensional space, where m is the number of edges in the tree. (See Figure 1(b) as an intuitive example; a formal definition will be given later.) This representation gives us a linear programming problem to find p vectors in the m-dimensional space, as formulated in Problem (3) later. However, since not all m-dimensional vectors correspond to a (feasible) point on the tree, the linear programming problem is a relaxation of MULTI-SERVICE CENTER. The key ingredient of our algorithm is to prove that the linear programming problem has in fact an optimal solution corresponding to an optimal facility location (Lemma 3). Since our proof is constructive, we can find an optimal facility location in polynomial time by solving the linear programming problem.

3.2 Algorithm

Let $T = (V, E)$ be a tree. We choose an arbitrary vertex r in V as the root of T, and regard T as a rooted tree. For notational convenience, when we denote an edge e by $e = uv$, we may assume that u is the parent of v. For each vertex v on T, we denote by P_v the path in T from r to v. For each interior point x of an edge $e_x = u_x v_x$, we denote by P_x the path in T from r to v_x, that is, $P_x = P_{v_x}$. For each edge $e = uv$, let $T - e$ be the subgraph of T obtained by deleting e from T. Then, $T - e$ consists of exactly two trees that have u and v, respectively; we denote the two trees by T_u and T_v where $u \in V(T_u)$ and $v \in V(T_v)$, respectively.

Let x be any point on T, and assume that x is located on an edge $e_x = u_x v_x$; note that $x = u_x$ or $x = v_x$ may hold. Then, we can express the point x using a vector $\tilde{x}$ in $\mathbb{R}^E_{\geq 0}$, defined as follows (see Figure 1(b)):

$$\tilde{x}(e) = \begin{cases} \ell_e & \text{if } e \in E(P_{u_x}) = E(P_x) \setminus \{e_x\}, \\ \mathsf{dist}(u_x, x) & \text{if } e = e_x, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Conversely, we say that a vector $\tilde{x} \in \mathbb{R}^E_{\geq 0}$ is *admissible* if there exist an edge $e_x = u_x v_x$ and d_x in $[0, \ell_{e_x}]$ such that $\tilde{x}$ has the form of (1) in which $\mathsf{dist}(u_x, x)$ is replaced with d_x. Then, there exists a one-to-one correspondence between a point $x \in T$ and an admissible vector $\tilde{x}$, and hence any point on T can be represented as an admissible vector. When a vertex $v \in V$ and a point x on T are expressed by $\tilde{v} \in \mathbb{R}^E_{\geq 0}$ and $\tilde{x} \in \mathbb{R}^E_{\geq 0}$, respectively, it holds that

$$\mathsf{dist}(v, x) = \sum_{e \in E} |\tilde{v}(e) - \tilde{x}(e)| \tag{2}$$

(see also Figure 1(c)), because we have

$$|\tilde{v}(e) - \tilde{x}(e)| = \begin{cases} \ell_e & \text{if } e \in (E(P_x) \triangle E(P_v)) \setminus \{e_x\}, \\ \mathsf{dist}(u_x, x) & \text{if } e = e_x \notin E(P_v), \\ \mathsf{dist}(v_x, x) & \text{if } e = e_x \in E(P_v), \\ 0 & \text{otherwise.} \end{cases}$$

For a vertex $v \in V$ and any vector $\tilde{x} \in \mathbb{R}^E_{\geq 0}$ (which is not necessarily admissible), we define

$$d_e(v, \tilde{x}) = |\tilde{v}(e) - \tilde{x}(e)|,$$

where $\tilde{v}$ is a vector expressing v by (1). Consider the problem of finding $|I|$ vectors $\tilde{x}_i \in \mathbb{R}^E_{\geq 0}$ $(i \in I)$ that minimizes

$$\max_{v \in V} \sum_{i \in I} \left(w_{v,i} \sum_{e \in E} d_e(v, \tilde{x}_i) \right) = \max_{v \in V} \sum_{i \in I} \left(w_{v,i} \sum_{e \in E} |\tilde{v}(e) - \tilde{x}_i(e)| \right) \tag{3}$$

subject to $\tilde{x}_i(e) \in [0, \ell_e]$ for $i \in I$ and $e \in E$. Note that, by (2), we have $\sum_{e \in E} d_e(v, \tilde{x}) = \mathsf{dist}(v, x)$ for any point x on T and its corresponding admissible vector $\tilde{x}$. Hence, if we have an additional constraint that each $\tilde{x}_i$ is admissible on the problem (3), then it is equivalent to MULTI-SERVICE CENTER. Thus the problem (3) can be seen as a relaxation of MULTI-SERVICE CENTER.

▶ **Lemma 2.** *The optimal value of the problem* (3) *is smaller than or equal to that of* MULTI-SERVICE CENTER *with nonnegative weights.*

Proof. Consider any optimal solution to MULTI-SERVICE CENTER with nonnegative weights which places each facility $i \in I$ at a point x_i on T. Then, the corresponding vectors $\tilde{x}_i$ form a feasible solution of the problem (3), and its objective value is equal to the optimal value of MULTI-SERVICE CENTER with nonnegative weights because of (2). Thus, the statement holds. ◀

We say that a feasible solution of the problem (3) is *admissible* if each vector $\tilde{x}_i$ $(i \in I)$ of the solution is admissible. Then, an admissible solution of the problem (3) gives a location of I on T. Lemma 2 and the following lemma ensure that solving the problem (3) is equivalent to solving MULTI-SERVICE CENTER with nonnegative weights.

▶ **Lemma 3.** *The problem* (3) *has an admissible optimal solution. Furthermore, given an optimal solution $\tilde{x}_i$ $(i \in I)$ to the problem* (3), *we can construct an admissible optimal solution in polynomial time.*

Proof. Let $\tilde{x}_i \in \mathbb{R}^E_{\geq 0}$ $(i \in I)$ be an optimal solution to the problem (3). For each edge $e \in E$, let P_e be the unique path in T from the root r to e which does *not* include e itself. Thus, $P_e = P_u$ for an edge $e = uv$. Let $F_i = \{e \in E \mid \tilde{x}_i(e) > 0\}$. By definition, $\tilde{x}_i$ is admissible if and only if it satisfies the following conditions:

(A) any two distinct edges $e_1, e_2 \in F_i$ satisfy either $e_1 \in E(P_{e_2})$ or $e_2 \in E(P_{e_1})$; and

(B) for each edge $e \in F_i$, there is no edge $e' \in E(P_e)$ such that $\tilde{x}_i(e') < \ell_{e'}$.

Suppose that $\tilde{x}_i$ is not admissible for some $i \in I$. We will show that we can modify the vector $\tilde{x}_i$ in polynomial time so that the resulting vector is admissible (i.e., satisfies both (A) and (B) above), without increasing the objective value of (3).

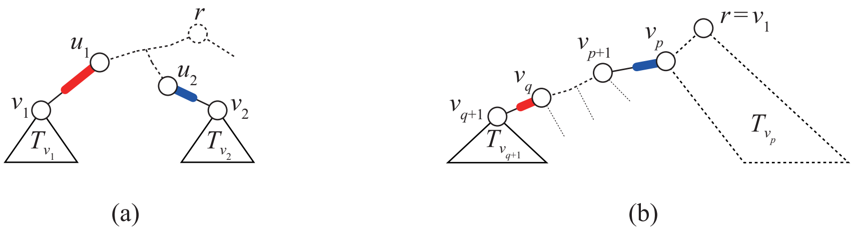


Figure 2 Illustration for the proof of Lemma 3.

We first modify $\tilde{x}_i$ so that it satisfies (A). Suppose that there exist two distinct edges $e_1 = u_1v_1$ and $e_2 = u_2v_2$ in F_i such that both $e_1 \notin E(P_{e_2})$ and $e_2 \notin E(P_{e_1})$ hold. (See Figure 2(a).) Define a new vector $\tilde{x}'_i$ by

$$\tilde{x}'_i(e) = \begin{cases} \tilde{x}_i(e) - \varepsilon & \text{if } e \in \{e_1, e_2\}; \\ \tilde{x}_i(e) & \text{otherwise,} \end{cases}$$

where $\varepsilon = \min\{\tilde{x}_i(e_1), \tilde{x}_i(e_2)\}$. Then, $\tilde{x}'_i(e) \in [0, \ell_e]$ for each $e \in E$, and hence $\tilde{x}'_i$ is feasible to (3). We now claim that this modification does not increase the objective value as follows. For each vertex $z \in V$ expressed by $\tilde{z} \in \mathbb{R}^E_{\geq 0}$ and an index $q \in \{1, 2\}$, we have

$$\tilde{z}(e_q) = \begin{cases} 0 & \text{if } z \in V(T_{u_q}); \\ \ell_{e_q} & \text{if } z \in V(T_{v_q}), \end{cases}$$

where we recall that T_{u_q} and T_{v_q} are trees in $T - e_q$ such that $u_q \in V(T_{u_q})$ and $v_q \in V(T_{v_q})$. We thus have

$$d_{e_q}(z, \tilde{x}'_i) - d_{e_q}(z, \tilde{x}_i) = \begin{cases} -\varepsilon & \text{if } z \in V(T_{u_q}); \\ \varepsilon & \text{if } z \in V(T_{v_q}). \end{cases}$$

Since $V(T_{v_1}) \cap V(T_{v_2}) = \emptyset$, the vertex z is contained in $V(T_{u_1})$ or $V(T_{u_2})$. Therefore, it holds that

$$\sum_{e \in E} d_e(z, \tilde{x}'_i) - \sum_{e \in E} d_e(z, \tilde{x}_i) = d_{e_1}(z, \tilde{x}'_i) - d_{e_1}(z, \tilde{x}_i) + d_{e_2}(z, \tilde{x}'_i) - d_{e_2}(z, \tilde{x}_i) \leq 0. \tag{4}$$

In this way, while F_i violates (A), we can repeatedly replace $\tilde{x}_i$ with $\tilde{x}'_i$ as above. Since this procedure decreases $|F_i|$ monotonically, the number of repetition is at most $|F_i| \leq |E|$. Thus we can obtain $\tilde{x}_i$ satisfying (A) in polynomial time.

We next modify $\tilde{x}_i$ so that it also satisfies (B). If $F_i = \emptyset$, then $\tilde{x}_i(e) = 0$ for any $e \in E$, and hence $\tilde{x}_i$ is admissible. Otherwise, since $\tilde{x}_i$ satisfies (A), all the edges in F_i are on some path P from r. Let $V(P) = \{v_1, v_2, \ldots, v_{k+1}\}$ and $E(P) = \{e_1, e_2, \ldots, e_k\}$ be the vertex set and the edge set of P, respectively, such that $v_1 = r$ and $e_j = v_jv_{j+1}$ for $j = 1, \ldots, k$. (See Figure 2(b).) Define $p := \min\{j \in \{1, \ldots, k\} \mid \tilde{x}_i(e_j) < \ell_{e_j}\}$; let $p = +\infty$ if such j does not exist. Define $q := \max\{j \in \{1, \ldots, k\} \mid \tilde{x}_i(e_j) > 0\}$; such j always exists because $F_i \subseteq E(P)$. Note that $\tilde{x}_i$ satisfies (B) if and only if $p \geq q$. Suppose that $\tilde{x}_i$ does not satisfy (B), that is, $p < q$. Then, $p \neq +\infty$ holds, and hence we have $\tilde{x}_i(e_p) < \ell_{e_p}$. Define a new vector $\tilde{x}'_i$ by

$$\tilde{x}'_i(e) = \begin{cases} \tilde{x}_i(e) + \varepsilon & \text{if } e = e_p; \\ \tilde{x}_i(e) - \varepsilon & \text{if } e = e_q; \\ \tilde{x}_i(e) & \text{otherwise,} \end{cases}$$

where $\varepsilon = \min\{\ell_{e_p} - \tilde{x}_i(e_p),\ \tilde{x}_i(e_q)\}$. Then, $\tilde{x}'_i(e) \in [0, \ell_e]$ for each $e \in E$, and hence $\tilde{x}'_i$ is feasible to (3). We now claim that this modification does not increase the objective value as follows. For any vertex $z \in V$, we have

$$d_{e_p}(z, \tilde{x}'_i) - d_{e_p}(z, \tilde{x}_i) = \begin{cases} \varepsilon & \text{if } z \in V(T_{v_p}); \\ -\varepsilon & \text{if } z \in V(T_{v_{p+1}}), \end{cases}$$

and

$$d_{e_q}(z, \tilde{x}'_i) - d_{e_q}(z, \tilde{x}_i) = \begin{cases} -\varepsilon & \text{if } z \in V(T_{v_q}); \\ \varepsilon & \text{if } z \in V(T_{v_{q+1}}), \end{cases}$$

where we recall that T_{v_p} and $T_{v_{p+1}}$ are trees in $T - e_p$ such that $v_p \in V(T_{v_p})$ and $v_{p+1} \in V(T_{v_{p+1}})$, and recall that T_{v_q} and $T_{v_{q+1}}$ are trees in $T - e_q$ such that $v_q \in V(T_{v_q})$ and $v_{q+1} \in V(T_{v_{q+1}})$. Since $V(T_{v_p}) \cap V(T_{v_{q+1}}) = \emptyset$, we can see that the objective value does not increase similarly to (4).

Therefore, we can repeat replacing $\tilde{x}_i$ with $\tilde{x}'_i$ as above while $\tilde{x}_i$ violates (B). Since this procedure either increases p or decreases q monotonically, we can finally obtain $\tilde{x}_i$ satisfying $p \geq q$, that satisfies (B), in polynomial time.

In this way, we can obtain an optimal solution x that is admissible in polynomial time. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. It follows from Lemmas 2 and 3 that it suffices to solve the problem (3). Note that $\sum_{i \in I} \left(w_{v,i} \sum_{e \in E} |\tilde{v}(e) - \tilde{x}_i(e)| \right)$ is a separable-convex function. Since the maximum of convex functions is also convex, so is the objective function of (3). Therefore, the problem (3) is a convex programming problem, which can be solved in polynomial time (see e.g., [3]).

In fact, we can reduce the problem (3) to the following linear programming problem:

$$\begin{aligned} &\text{minimize} && c \\ &\text{subject to} && \sum_{i \in I} \left(w_{v,i} \sum_{e \in E} |\tilde{v}(e) - \tilde{x}_i(e)| \right) \leq c \quad (v \in V), \\ &&& \tilde{x}_i(e) \leq \ell_e \quad (i \in I,\ e \in E), \\ &&& \tilde{x}_i \in \mathbb{R}^E_{\geq 0} \quad (i \in I), \\ &&& c \in \mathbb{R}_{\geq 0}, \end{aligned}$$

where $\tilde{x}_i(e)$ $(i \in I,\ e \in E)$ and c are variables. Note that the first constraint can be described by linear inequalities, since the left-hand side is

$$\sum_{i \in I} \left(w_{v,i} \sum_{e \in E} |\tilde{v}(e) - \tilde{x}_i(e)| \right) = \sum_{i \in I} w_{v,i} \left(\sum_{e \in E(P_v)} (\ell_e - \tilde{x}_i(e)) + \sum_{e \in E \setminus E(P_v)} \tilde{x}_i(e) \right).$$

Therefore, it is a linear programming problem with polynomial size, which can be solved in polynomial time (see e.g., [9]). ◀

4 Hardness Results

In this section, we show that MULTI-SERVICE CENTER is computationally intractable even for very restricted instances. We emphasize again that our analyses are sharp in contrast to Theorem 1.

4.1 Technical Highlights

Recall that, in MULTI-SERVICE CENTER, we are allowed to place each facility at any point on a graph (even on an edge), which makes a solution flexible. We design reductions so that reduced instances force all facilities to be placed at only vertices in any optimal solution. To ensure this condition, we need to analyze the structure of optimal solutions carefully. Interestingly, we will verify this condition for cycles (Theorem 7) by using the nonsingularity of a "distance matrix" [2], which has been studied in the area of algebraic graph theory.

4.2 NP-hardness for paths and stars with negative weights

In this subsection, we show that MULTI-SERVICE CENTER is intractable even for paths and stars if weights of vertices take negative integers. More specifically, the problem is NP-hard for paths of only three vertices, and is strongly NP-hard for stars. Indeed, a path of three vertices is a star, and hence we will construct a common reduction from the following problem:

The EQUALLY PARTITION problem

Instance. A set A of elements, a bound $b \in \mathbb{Z}_{\geq 0}$, and a size $s_i \in \mathbb{Z}_{\geq 0}$ for each $i \in A$ such that $\sum_{i \in A} s_i = mb$ for some positive integer m.

Question. Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that $\sum_{i \in A_j} s_i = b$ for all $j \in \{1, 2, \ldots, m\}$?

Here, $\mathbb{Z}_{\geq 0}$ is the set of all nonnegative integers. We summarize our reduction from EQUALLY PARTITION to MULTI-SERVICE CENTER as in the following theorem.

▶ **Theorem 4.** *There is a polynomial-time reduction from* EQUALLY PARTITION *to* MULTI-SERVICE CENTER *for instances such that*

(a) $G = (V, E)$ *is a star* $K_{1,m}$ *with the center vertex* r *having* m *leaves*;

(b) $\ell_e := 1$ *for every* $e \in E$;

(c) $I := A$; *and*

(d) *for* $v \in V$ *and* $i \in I$ $(= A)$,

$$w_{v,i} := \begin{cases} -s_i & \text{if } v \in V \setminus \{r\}; \\ -s_i \cdot \frac{2(m-1)}{m} & \text{if } v = r. \end{cases}$$

Notice that EQUALLY PARTITION corresponds to an NP-hard problem PARTITION [5, SP12] if $m = 2$. In addition, for general m, EQUALLY PARTITION contains all instances of a strongly NP-hard problem 3-PARTITION [5, SP15]. Thus, the following corollary can be obtained from Theorem 4.

▶ **Corollary 5.** *The following* (i) *and* (ii) *hold.*

(i) MULTI-SERVICE CENTER *is NP-hard even when* $G = (V, E)$ *is a path of three vertices,* $\ell_e = 1$ *for every* $e \in E$*, and* $w_{v,i} = w_{v',i}$ *for any* $v, v' \in V$ *and* $i \in I$.

(ii) MULTI-SERVICE CENTER *is NP-hard in the strong sense even when* $G = (V, E)$ *is a star, and* $\ell_e = 1$ *for every* $e \in E$.

As described in Theorem 4, our reduction from EQUALLY PARTITION to MULTI-SERVICE CENTER is as follows. Suppose that we are given an instance of EQUALLY PARTITION, that is, a set A of elements, a bound $b \in \mathbb{Z}_{\geq 0}$, and a size $s_i \in \mathbb{Z}_{\geq 0}$ for each $i \in A$ such that $\sum_{i \in A} s_i = mb$. Then, we construct a corresponding instance of MULTI-SERVICE CENTER as follows. Let $G = (V, E)$ be a star $K_{1,m}$ with the center vertex r and having m leaves $v_1, v_2, \ldots, v_m$. Set $\ell_e := 1$ for every $e \in E$, and define $I := A$. For $v \in V$ and $i \in I$ $(= A)$, set

$$w_{v,i} := \begin{cases} -s_i & \text{if } v \in V \setminus \{r\}; \\ -s_i \cdot \frac{2(m-1)}{m} & \text{if } v = r. \end{cases}$$

This reduction can be done in polynomial time.

To show the correctness of our reduction above, it suffices to prove the following lemma.

▶ **Lemma 6.** *The original instance of* EQUALLY PARTITION *has a desired partition if and only if there is a location X of I for the corresponding instance of* MULTI-SERVICE CENTER *such that* $\max_{v \in V} \mathsf{cost}(v, X) \le -2(m-1)b$.

Proof. Necessity ("only if" part). Suppose that the original instance of EQUALLY PARTITION has a partition $(A_1, A_2, \ldots, A_m)$ of A such that $\sum_{i \in A_j} s_i = b$ for all $j \in \{1, 2, \ldots, m\}$. In this case, we place the facilities in A_j at the vertex $v_j \in V$, that is, for each $i \in A_j$, we define $x_i := v_j$ in the corresponding instance of MULTI-SERVICE CENTER. Since $(A_1, A_2, \ldots, A_m)$ is a partition of $A = I$, this properly defines a location X of I. Then, for each leaf $v_j \in V \setminus \{r\}$, we can estimate the cost of v to receive the service from X as follows:

$$\mathsf{cost}(v_j, X) = \sum_{i \in I} w_{v_j,i} \cdot \mathsf{dist}(v_j, x_i) = 2 \cdot \sum_{i \in I \setminus A_j} (-s_i) = -2(m-1)b.$$

Similarly, for the center vertex r of the star, its cost can be estimated as follows:

$$\mathsf{cost}(r, X) = \sum_{i \in I} w_{r,i} \cdot \mathsf{dist}(r, x_i) = \sum_{i \in I} \left(-s_i \cdot \frac{2(m-1)}{m} \right) = -2(m-1)b.$$

Therefore, X is a location of I for the corresponding instance of MULTI-SERVICE CENTER such that $\max_{v \in V} \mathsf{cost}(v, X) \le -2(m-1)b$, as required.

Sufficiency ("if" part). Suppose that there is a location $X \in G^I$ of I for the corresponding instance of MULTI-SERVICE CENTER such that $\max_{v \in V} \mathsf{cost}(v, X) \le -2(m-1)b$. For each facility $i \in I$, let x_i denote the point on G at which i is placed by X. Since r is the center vertex of the star, $\mathsf{dist}(r, x_i) \le 1$ for any $i \in I$. In addition, since $w_{r,i}$ is negative for any $i \in I$, we have

$$\mathsf{cost}(r, X) = \sum_{i \in I} w_{r,i} \cdot \mathsf{dist}(r, x_i) \ge \sum_{i \in I} \left(-s_i \cdot \frac{2(m-1)}{m} \right) = -2(m-1)b.$$

Since we have assumed that $\max_{v \in V} \mathsf{cost}(v, X) \le -2(m-1)b$, the inequality above is tight. We thus have $\mathsf{dist}(r, x_i) = 1$ for any $i \in I$. Observe that $\mathsf{dist}(r, x_i) = 1$ means that x_i is equal to one of the points $v_1, v_2, \ldots, v_m$. With this observation, we obtain a partition $(A_1, A_2, \ldots, A_m)$ of A by defining $A_j := \{i \in I \mid x_i = v_j\}$ for each $j \in \{1, 2, \ldots, m\}$.

We now claim that $\sum_{i \in A_j} s_i = b$ for all $j \in \{1, 2, \ldots, m\}$, and hence $(A_1, A_2, \ldots, A_m)$ is a desired partition for EQUALLY PARTITION. To see this, we evaluate $\mathsf{cost}(v_j, X)$ as follows:

$$\begin{aligned} \max_{j \in \{1,2,\ldots,m\}} \mathsf{cost}(v_j, X) &= \max_{j \in \{1,2,\ldots,m\}} \left(\sum_{i \in I} w_{v_j,i} \cdot \mathsf{dist}(v_j, x_i) \right) \\ &\ge \frac{1}{m} \sum_{j=1}^{m} \sum_{i \in I} w_{v_j,i} \cdot \mathsf{dist}(v_j, x_i) = \frac{1}{m} \sum_{j=1}^{m} \sum_{i \in I \setminus A_j} (-s_i) \cdot 2 \\ &= -\frac{2}{m} \sum_{i \in I} (m-1) s_i = -2(m-1)b. \end{aligned}$$

Since we have assumed that $\max_{v\in V} \mathsf{cost}(v, X) \leq -2(m-1)b$, the inequality above is tight. Then, the tightness of the inequality shows that $\sum_{i\in I} w_{v_j,i}\mathsf{dist}(v_j, x_i) = -2(m-1)b$ for every $v_j \in V \setminus \{r\}$. Therefore, we have

$$-2(m-1)b = \sum_{i\in I} w_{v_j,i} \cdot \mathsf{dist}(v_j, x_i) = \sum_{i\in I\setminus A_j} (-s_i) \cdot 2 = -2\left(mb - \sum_{i\in A_j} s_i\right).$$

We thus have $\sum_{i\in A_j} s_i = b$ for all $j \in \{1, 2, \ldots, m\}$, as claimed. ◀

This completes the proof of Theorem 4, and hence Corollary 5 follows.

4.3 Strong NP-hardness for cycles with nonnegative weights

We show that the problem is strongly NP-hard even when restricted to cycles with identical edge-length and nonnegative integer weights.

▶ **Theorem 7.** Multi-service center *with nonnegative weights is NP-hard in the strong sense even when $G = (V, E)$ is a cycle, $\ell_e = 1$ for every $e \in E$, and $w_{v,i} = w_{v',i} \in \mathbb{Z}_{\geq 0}$ for any $v, v' \in V$ and $i \in I$.*

Thus, multi-service center cannot be solved in pseudo-polynomial time even for such restricted instances unless P = NP.

In the remainder of this subsection, we prove the theorem by giving a polynomial-time reduction from a strongly NP-hard problem 3-partition to multi-service center for such restricted instances. The 3-partition problem is defined as follows (see, e.g., [5, SP15]):

The 3-partition problem

Instance. A set A of $3m$ elements, a bound $b \in \mathbb{Z}_{\geq 0}$, and a size $s_i \in \mathbb{Z}_{\geq 0}$ with $\frac{b}{4} < s_i < \frac{b}{2}$ for each $i \in A$ such that $\sum_{i\in A} s_i = mb$.

Question. Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that $\sum_{i\in A_j} s_i = b$ for all $j \in \{1, 2, \ldots, m\}$?

Note that since $\frac{b}{4} < s_i < \frac{b}{2}$ for each $i \in A$, we have $|A_j| = 3$ for all $j \in \{1, 2, \ldots, m\}$. It is known that 3-partition remains NP-hard in the strong sense even if m is restricted to be odd [8].

Suppose that we are given a set A of $3m$ elements, a bound $b \in \mathbb{Z}_{\geq 0}$, and a size $s_i \in \mathbb{Z}_{\geq 0}$ for each $i \in A$ as an instance of 3-partition, where m is an odd number. We construct a corresponding instance of multi-service center as follows. Let $G = (V, E)$ be a cycle with m vertices such that $V = \{v_1, v_2, \ldots, v_m\}$, $E = \{v_1v_2,\ v_2v_3,\ \ldots,\ v_{m-1}v_m,\ v_mv_1\}$, and $\ell_e := 1$ for every $e \in E$. Define $I := A$, and set $w_{v,i} := s_i$ for $v \in V$ and $i \in I$ $(= A)$. This reduction can be done in polynomial time.

To show the correctness of our reduction above, it suffices to prove the following lemma.

▶ **Lemma 8.** *The original instance of* 3-partition *has a desired partition if and only if there is a location X of I for the corresponding instance of* multi-service center *such that*

$$\max_{v\in V} \mathsf{cost}(v, X) \leq \frac{(m^2-1)b}{4}.$$

Proof. Necessity ("only if" part). Suppose that the original instance of 3-partition has a partition $(A_1, A_2, \ldots, A_m)$ of A such that $\sum_{i\in A_j} s_i = b$ for all $j \in \{1, 2, \ldots, m\}$. In this case, we place the (three) facilities in A_j at the vertex $v_j \in V$, that is, for each $i \in A_j$, we define

$x_i := v_j$ in the corresponding instance of MULTI-SERVICE CENTER. Since $(A_1, A_2, \ldots, A_m)$ is a partition of $A = I$, this properly defines a location X of I. Then, for each vertex $v \in V$, we can estimate the cost of v to receive the service from X as follows:

$$\begin{aligned}\mathsf{cost}(v, X) &= \sum_{i\in I} w_{v,i} \cdot \mathsf{dist}(v, x_i) = \sum_{j=1}^{m} \left(\sum_{i\in A_j} w_{v,i} \cdot \mathsf{dist}(v, v_j) \right) \\ &= \sum_{j=1}^{m} \left(\mathsf{dist}(v, v_j) \sum_{i\in A_j} s_i \right) = b \sum_{j=1}^{m} \mathsf{dist}(v, v_j) = 2b \sum_{k=1}^{(m-1)/2} k = \frac{(m^2-1)b}{4}.\end{aligned}$$

Therefore, X is a location of I for the corresponding instance of MULTI-SERVICE CENTER such that $\max_{v\in V} \mathsf{cost}(v, X) \leq \frac{(m^2-1)b}{4}$, as required.

Sufficiency ("if" part). Suppose that there is a location $X \in G^I$ of I for the corresponding instance of MULTI-SERVICE CENTER such that $\max_{v\in V} \mathsf{cost}(v, X) \leq \frac{(m^2-1)b}{4}$. For each facility $i \in I$, let x_i denote the point on G at which i is placed by X. We will prove the following (a) and (b):

(a) X places all facilities in I at vertices of G; and

(b) $\sum_{i:x_i=v_j} s_i = b$ for every $v_j \in V$.

Then, by defining $A_j := \{i \in I \mid x_i = v_j\}$ for each $j \in \{1, 2, \ldots, m\}$, we obtain a desired partition $(A_1, A_2, \ldots, A_m)$ of A for the original instance of 3-PARTITION.

We first prove (a). To see properties of the location X, we begin with the following claim.

▶ **Claim 9.** *For any point x on G, it holds that $\sum_{j=1}^{m} \mathsf{dist}(v_j, x) \geq \frac{m^2-1}{4}$. Furthermore, $\sum_{j=1}^{m} \mathsf{dist}(v_j, x) = \frac{m^2-1}{4}$ holds if and only if x is a vertex of G.*

Proof of the claim. Let $\varepsilon \geq 0$ be the distance from x to the nearest vertex in V. Then,

$$\sum_{j=1}^{m} \mathsf{dist}(v_j, x) = \sum_{k=1}^{(m-1)/2} (k - \varepsilon) + \sum_{k=0}^{(m-1)/2} (k + \varepsilon) = \frac{m^2-1}{4} + \varepsilon.$$

This shows the claim, because $\varepsilon = 0$ if and only if x is a vertex of G. ◀

By Claim 9, we have

$$\begin{aligned}\max_{v\in V} \mathsf{cost}(v, X) &= \max_{v\in V} \left(\sum_{i\in I} w_{v,i} \mathsf{dist}(v, x_i) \right) \\ &\geq \frac{1}{m} \sum_{v\in V} \sum_{i\in I} s_i \mathsf{dist}(v, x_i) = \frac{1}{m} \sum_{i\in I} \left(s_i \sum_{v\in V} \mathsf{dist}(v, x_i) \right) \\ &\geq \frac{1}{m} \sum_{i\in I} \left(s_i \cdot \frac{m^2-1}{4} \right) = \frac{(m^2-1)b}{4}. \qquad (5)\end{aligned}$$

Since we have assumed that $\max_{v\in V} \mathsf{cost}(v, X) \leq \frac{(m^2-1)b}{4}$, all the inequalities above are tight. The tightness of the inequality in (5) shows that the point x_i is a vertex of G for each $i \in I$ by Claim 9.

We then prove (b). Define $y_u := \sum_{i:x_i=u} s_i$ for each $u \in V$, and define $y \in \mathbb{R}^V$ as the vector consisting of y_u's. The tightness of the above inequalities shows that $\sum_{i\in I} s_i \mathsf{dist}(v, x_i) =$

$\frac{(m^2-1)b}{4}$ for every $v \in V$, which is equivalent to

$$\sum_{u \in V} y_u \mathsf{dist}(v,u) = \frac{(m^2-1)b}{4} \text{ for every } v \in V. \tag{6}$$

Let $D \in \mathbb{R}^{V \times V}$ be the distance matrix of G defined by $D_{uv} = \mathsf{dist}(u,v)$ for $u, v \in V$. Then, (6) is represented as $Dy = \frac{(m^2-1)b}{4} \cdot \mathbf{1}$, where $\mathbf{1}$ is the all-one vector in $\mathbb{R}^V$. Since $D\mathbf{1} = \frac{m^2-1}{4} \cdot \mathbf{1}$ by a simple calculation, we have

$$D(y - b\mathbf{1}) = \mathbf{0}. \tag{7}$$

It is shown in [2, Theorem 3.4] that the determinant of D is equal to $\frac{m^2-1}{4}$, which implies that D is nonsingular. Thus, (7) shows that $y = b\mathbf{1}$, that is, $y_u = b$ for every $u \in V$. ◀

This completes the proof of Theorem 7.

We finally note that our reductions indeed show that MULTI-SERVICE CENTER remains computationally hard even with an additional constraint that all facilities must be placed at only vertices.

References

1 Toshimitsu Anzai, Takehiro Ito, Akira Suzuki, and Xiao Zhou. The multi-service center decision problem is NP-complete for split graphs. In *the 6th World Congress on Engineering and Technology* (*CET 2016*), 2016.

2 Ravindra B. Bapat, Stephen J. Kirkland, and Michael Neumann. On distance matrices and Laplacians. *Linear Algebra and Its Applications*, 401:193–209, 2005.

3 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

4 Zvi Drezner and Horst W. Hamacher, editors. *Facility Location*: *Applications and Theory.* Springer-Verlag, Berlin Heidelberg, 2002.

5 Michael R. Garey and David S. Johnson. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co., New York, NY, USA, 1990.

6 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

7 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10:180–184, 1985.

8 Sadish Sadasivam and Huaming Zhang. NP-completeness of st-orientations for plane graphs. *Theoretical Computer Science*, 411:995–1003, 2010.

9 Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

10 Hung-I Yu and Cheng-Chung Li. The multi-service center problem. In *the 23rd International Symposium on Algorithms and Computation* (*ISAAC 2012*), volume 7676 of *Lecture Notes in Computer Science*, pages 578–587, 2012.

Improved Algorithms for Scheduling Unsplittable Flows on Paths*†

Hamidreza Jahanjou[1], Erez Kantor[2], and Rajmohan Rajaraman[3]

1 **Northeastern University, Boston, USA**
`hamid@ccs.neu.edu`

2 **University of Massachusetts, Amherst, USA**
`erez.kantor@gmail.com`

3 **Northeastern University, Boston, USA**
`rraj@ccs.neu.edu`

Abstract

In this paper, we investigate offline and online algorithms for Round-UFPP, the problem of minimizing the number of rounds required to schedule a set of unsplittable flows of non-uniform sizes on a given path with non-uniform edge capacities. Round-UFPP is NP-hard and constant-factor approximation algorithms are known under the no bottleneck assumption (NBA), which stipulates that maximum size of a flow is at most the minimum edge capacity. We study Round-UFPP *without the NBA*, and present improved online and offline algorithms. We first study offline Round-UFPP for a restricted class of instances called α-small, where the size of each flow is at most α times the capacity of its bottleneck edge, and present an $O(\log(1/(1-\alpha)))$-approximation algorithm. Our main result is an online $O(\log\log c_{\max})$-competitive algorithm for Round-UFPP for general instances, where $c_{\max}$ is the largest edge capacities, improving upon the previous best bound of $O(\log c_{\max})$ due to [16]. Our result leads to an offline $O(\min(\log n, \log m, \log\log c_{\max}))$-approximation algorithm and an online $O(\min(\log m, \log\log c_{\max}))$-competitive algorithm for Round-UFPP, where n is the number of flows and m is the number of edges.

1998 ACM Subject Classification F.2.2 Nonnumerical algorithms and Problems

Keywords and phrases Approximation algorithms, Online algorithms, Unsplittable flows, Interval coloring, Flow scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.49

1 Introduction

The *unsplittable flow problem on paths* (UFPP) considers selecting a maximum-weight subset of flows to be routed simultaneously over a path while satisfying capacity constraints on the edges of the path. In this work, we investigate a variant of UFPP known in the literature as Round-UFPP or *capacitated interval coloring.* The objective in Round-UFPP is to schedule *all* the flows in the smallest number of rounds, subject to the constraint that the flows scheduled in a given round together respect edge capacities. Formally, in Round-UFPP we are given a path $P = (V, E)$, consisting of m links, with capacities $\{c_j\}_{j\in[m]}$, and a set of n flows $\mathcal{F} = \{f_i = (s_i, t_i, \sigma_i) : i \in [n]\}$ each consisting of a source vertex, a sink vertex, and a size. A set R of flows is feasible if all of its members can be scheduled simultaneously while satisfying

[*] This work was partially supported by NSF grant CCF-1422715, a Google Research Award, and an ONR grant on network algorithms.

[†] A full version of the paper is available at `https://arxiv.org/abs/1708.00143`.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 49; pp. 49:1–49:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

capacity constraints. The objective is to partition $\mathcal{F}$ into the smallest number of feasible sets (rounds) $R_1, ..., R_t$.

One practical motivation for Round-UFP is routing in optical networks. Specifically, a flow f_i of size σ_i can be regarded as a connection request asking for a bandwidth of size σ_i. Connections using the same communication link can be routed at the same time as long as the total bandwidth requested is at most the link capacity. Most modern networks have heterogeneous link capacities; for example, some links might be older than others. In this setting, each round (or color) corresponds to a transmission frequency, and minimizing the number of frequencies is a natural objective in optical networks.

A common simplifying assumption, known as the no-bottleneck assumption (NBA), stipulates that the maximum demand size is at most the (global) minimum link capacity; i.e. $\max_{i\in[n]} \sigma_i \leq \min_{j\in[m]} c_j$; most results on UFPP and its variants are under the NBA (see §1.1). A major breakthrough was the design of $O(1)$-approximation algorithms for the unsplittable flow problem on paths (UFPP) without the NBA [10, 2]. In this paper, we make progress towards an optimal algorithm for Round-UFPP *without* imposing NBA.

We consider both offline and online versions of Round-UFPP. In the offline case, all flows are known in advance. In the online case, however, the flows are not known *à priori* and they appear one at a time. Moreover, every flow must be scheduled (i.e. assigned to a partition) immediately on arrival; no further changes to the schedule are allowed.

Even the simpler problem Round-UFPP-NBA, that is, Round-UFPP with the NBA, in the offline case, is **NP**-hard since it contains Bin Packing as a special case (consider an instance with a single edge). On the other hand, if all capacities and flow sizes are equal, then the problem reduces to interval coloring which is solvable by a simple greedy algorithm.

1.1 Previous work

The unsplittable flow problem on paths (UFPP) concerns selecting a maximum-weight subset of flows without violating edge capacities. UFPP is a special case of UFP, the unsplittable flow problem on general graphs. The term, *unsplittable* refers to the requirement that each flow must be routed on a single path from source to sink. [1] UFPP, especially under the NBA, UFPP-NBA, and its variants have been extensively studied [9, 3, 7, 6, 8, 11, 14, 22, 13]. Recently, $O(1)$-approximation algorithms were discovered for UFPP (without NBA) [10, 2]. Note that, on general graphs, UFP-NBA is **APX**-hard even on depth-3 trees where all demands are 1 and all edge capacities are either 1 or 2 [18].

Round-UFPP has been mostly studied in the online setting where it generalizes the interval coloring problem (ICP) which corresponds to the case where all demands and capacities are equal. In their seminal work, Kierstead and Trotter gave an optimal online algorithm for ICP with a competitive ratio of $3\omega - 2$, where ω denotes the maximum clique size [20]. Note that, since interval graphs are prefect, the optimal solution is simply ω. Many works consider the performance of the first-fit algorithm on interval graphs. Adamy and Erlebach were the first to generalize ICP [1]. In their problem, interval coloring with bandwidth, all capacities are 1 and each flow f_i has a size $\sigma_i \in (0, 1]$. The best competitive ratio known for this problem is 10 [5, 17] and a lower bound of slightly greater than 3 is known [19]. The online Round-UFPP is considered in Epstein et. al. [16]. They give a 78-competitive algorithm for Round-UFPP-NBA, an $O(\log \frac{\sigma_{\max}}{c_{\min}})$-competitive algorithm for the general Round-UFPP, and

[1] Clearly, in the case of paths and trees, the term is redundant. We use the terminology UFPP to be consistent with the considerable prior work in this area.

lower bounds of $\Omega(\log\log n)$ and $\Omega(\log\log\log \frac{c_{\max}}{c_{\min}})$ on the competitive ratio achievable for Round-UFPP. In the offline setting, a 24-approximation algorithm for Round-UFPP-NBA is presented in [15].

1.2 Our results

We design improved algorithms for offline and online Round-UFPP. Let m denote the number of edges in the path, n the number of flows, and $c_{\max}$ the maximum edge capacity.

- In §3, we design an $O(log(1/(1-\alpha)))$-approximation algorithm for offline Round-UFPP for α-small instances in which the size of each flow is at most an α fraction of the capacity of the smallest edge used by the flow, where $0 < \alpha < 1$. This implies an $O(1)$-approximation for any α-small instance, with constant α. Previously, constant-factor approximations were only known for $\alpha \leq 1/4$.
- In §4, we present our main result, an online $O(\log\log c_{\max})$)-competitive algorithm for general instances. This result leads to an offline $O(\min(\log n, \log m, \log\log c_{\max}))$-approximation algorithm and an online $O(\min(\log m, \log\log c_{\max}))$-competitive algorithm.

Our algorithm for general instances, which improves on the $O(\log c_{\max})$-bound achieved in [16], is based on a reduction to the classic rectangle coloring problem (e.g., see [4, 21, 12]). We introduce a class of "line-sparse" instances of rectangle coloring that may be of independent interest, and show how competitive algorithms for such instances lead to competitive algorithms for Round-UFPP.

Due to space limitations, we are unable to include all of the proofs in the main body; we refer the reader to the full version of this paper[2] for any missing proof and pseudocode as well as extra figures.

2 Preliminaries

In Round-UFPP we are given a path $P = (V, E)$ consisting of $m + 1$ vertices and m links, enumerated left-to-right as $v_0, e_1, v_1, ..., v_{m-1}, e_m, v_m$, with edge capacities $\{c_j\}_{j\in[m]}$, and a set of n flows $\mathcal{F} = \{f_i = (s_i, t_i, \sigma_i) : i \in [n]\}$, where s_i and t_i represent the two endpoints of flow f_i, and σ_i denotes the size of the flow. Without loss of generality, we assume that $s_i < t_i$. We say that a flow f_i uses a link e_j if $s_i < j \leq t_i$. For a set of flows F, we denote by $F(e)$ and $F(j)$ the subset of flows in F using edge e and e_j respectively.

▶ **Definition 1.** The *bottleneck capacity* of a flow f_i, denoted by b_i, is the smallest capacity among all links used by f_i – such an edge is called the bottleneck edge for flow f_i.

A set of flows R is called *feasible* if all of its members can be routed simultaneously without causing capacity violation. The objective is to partition $\mathcal{F}$ into the smallest number of feasible sets $R_1, ..., R_t$. A feasible set is also referred to as a *round*.

Alternatively, partitioning can be seen as coloring where rounds correspond to colors.

▶ **Definition 2.** For a set of flows F, we define its *chromatic number*, $\chi(F)$, to be smallest number of rounds (colors) into which F can be partitioned.

▶ **Definition 3.** The *congestion* of an edge e_j with respect to a set of flows F is

$$r_j(F) = \frac{\sum_{i\in F(j)} \sigma_i}{c_j}, \tag{1}$$

[2] https://arxiv.org/abs/1708.00143

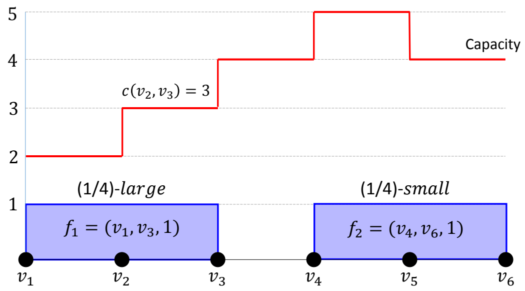


Figure 1 An example of a path with 5 links and two flows. The first flow f_1 is from v_1 to v_3 of size 1; the second flow f_2 is from v_4 to v_6 also of size 1. Even though both flows have the same size, f_1 is $\frac{1}{4}$-large whereas f_2 is $\frac{1}{4}$-small. The reason is different bottleneck capacities, $b_1 = 2$ and $b_2 = 4$.

that is, the ratio of the total size of flows in F using e_j to its capacity. Likewise $r_e(F)$ denotes the congestion of an edge e with respect to F. Also, let $r_{\max}(F) = \max_j r_j(F)$ be the maximum edge congestion with respect to F. When the set of flows is clear from the context, we simply write $r_{\max}$.

An obvious lower bound on $\chi(\mathcal{F})$ is maximum edge congestion; that is,

▶ **Observation 4.** $\chi(\mathcal{F}) \geq \lceil r_{\max}(\mathcal{F}) \rceil$.

Proof. Suppose e_j is any edge of the path. In each round, the amount of flow passing through the edge is at most its capacity c_j. Therefore, the number of rounds required for the flows in F using e_j to be scheduled is at least $\lceil r_j(F) \rceil$. ◀

Without loss of generality, we assume that the minimum capacity, $c_{\min}$, is 1. Furthermore, let $c_{\max} = \max_{e \in E} c_e$ denote the maximum edge capacity. As is standard in the literature, we classify flows according to the ratio of size to bottleneck capacity.

▶ **Definition 5.** Let α be a real number satisfying $0 \leq \alpha \leq 1$. A flow f_i is said to be α-*small* if $\sigma_i \leq \alpha \cdot b_i$ and α-*large* if $\sigma_i > \alpha \cdot b_i$ (refer to Figure 1 for an example). Accordingly, the set of flows $\mathcal{F}$ is divided into small and large classes

$$F^S_\alpha = \{f \in F \mid f \text{ is } \alpha\text{-small}\}; \quad F^L_\alpha = \{f \in F \mid f \text{ is } \alpha\text{-large}\}.$$

As is often the case for unsplittable flow algorithms, we treat small and large instances independently. In §3 and §4 we study small and large instances respectively.

3 An approximation algorithm for Round-UFPP with α-small flows

In this section, we design an offline $O(1)$-approximation algorithm for α-small flows for *any* $\alpha \in (0, 1)$. We note that offline and online algorithms for α-small instances are known when α is sufficiently small. More precisely, if $\alpha = 1/4$, 16-approximation and 32-competitive algorithms for offline and online cases have been presented in [15] and [16] respectively.

▶ **Lemma 6** ([15, 16]). *There exist $O(1)$-approximation algorithms for* Round-UFPP *where all flows are $\frac{1}{4}$-small.*

Algorithm 1: `ProcMids`

```
input  : A set of $[\frac{1}{4}, \alpha]$-mid flows $F$
output : A partition of $F$ into rounds (colors)
for $i \leftarrow 1$ to $\lceil \log c_{\max} \rceil$ do
    $F_i \leftarrow \{f_k \in F \mid 2^{i-1} \leq b_k < 2^i\}$;
    $(C_1^i, C_2^i) \leftarrow$ FlowDec($F_i$);
$R \leftarrow$ ColOptimize($\{(C_1^k, C_2^k) : k = 1, ..., \lceil \log c_{\max} \rceil\}$);
return $R$;
```

However, these results do not extend to the case where α is an arbitrary constant in $(0, 1)$. In contrast, we present an algorithm that works for any choice of $\alpha \in (0, 1)$. In our algorithm, flows are partitioned according to the ratio of their size to their bottleneck capacity. If $\alpha \leq 1/4$, we simply use Lemma 6. Suppose that $\alpha > 1/4$. The overall idea is to further partition the set of flows into two subsets and solve each independently. This motivates the following definition.

▶ **Definition 7.** Given two real numbers $0 \leq \beta < \alpha < 1$, a flow f_i is said to be $[\beta, \alpha]$-*mid* if $\sigma_i \in [\beta \cdot b_i, \alpha \cdot b_i]$. Accordingly, we define the corresponding set of flows as

$$F^M(\beta, \alpha) = \{f \in F \mid f \text{ is } [\beta, \alpha]\text{-mid}\}.$$

Observe that, $F^M(\beta, \alpha) = F^S_\alpha \cap F^L_\beta$.

In the remainder of this section, we present an $O(1)$-approximation algorithm, called `ProcMids`, for $F^M(1/4, \alpha)$. `ProcMids` (see Algorithm 1) starts by partitioning $F^M(1/4, \alpha)$ into $\lceil \log c_{\max} \rceil$ classes according to their bottleneck capacity.

Next, it computes a coloring for each class by running a separate procedure called `FlowDec`, explained in §3.1. This will result in a coloring of $F^M(1/4, \alpha)$ using $O(r_{\max} \log c_{\max})$ colors. Finally, `ProcMids` runs `ColOptimize`, described in §3.2, to optimize color usage in different subsets; this results in the removal logarithmic factor and, thereby, a more efficient coloring using only $O(r_{\max})$ colors.

3.1 A logarithmic approximation

Procedure `FlowDec` partitions F^M_ℓ into $O(r_{\max}(F^M_\ell))$ rounds. In each iteration, it calls procedure `rCover` (Algorithm 2) which takes as input a subset $F'_\ell \subseteq F^M_\ell$ and returns two disjoint feasible subsets C_1, C_2 of F'_ℓ. In other words, flows in each subset can be scheduled simultaneously without causing any capacity violation. On the other hand, these two subsets cover all the links used by the flows in F'_ℓ. More formally, C_1 and C_2 are guaranteed to have the following two properties:

(P1) $\forall e \in E : |C_1(e)| \leq 1$ and $|C_2(e)| \leq 1$,
(P2) $|F'_\ell(e)| > 1 \Rightarrow C_1(e) \cup C_2(e) \neq \emptyset$.

`rCover` maintains a set of flows F'' which is initially empty. It starts by finding the longest flow f_{i_1} among those having the first (leftmost) source node. Next, it processes the flows in a loop. In each iteration, the procedure looks for a flow overlapping with the currently selected flow f_{i_k}. If one is found, it is added to the collection and becomes the current flow. Otherwise, the next flow is chosen among those remaining flows that start after the current flow's sink t_{i_k}. Finally, `rCover` splits F'' into two feasible subsets and returns them.

Algorithm 2: `rCover`

```
input  : A set of flows $F$
output : Two disjoint feasible subsets of $F$ satisfying Properties (P1) and (P2)
$F'' \leftarrow \emptyset$;
$s_{\min} \leftarrow \min_{f_k \in F} s_k$;
$t_{i_1} \leftarrow \max\{t_k \mid f_k \in F \text{ and } s_k = s_{\min}\}$;
$F'' \leftarrow \{f_{i_1}\}$;
$F \leftarrow F \setminus \{f_{i_1}\}$;
$k \leftarrow 1$;
while $t_{i_k} < \max_{f_i \in F}\{t_i\}$ do
    if $\exists f_i \in F : s_i \leq t_{i_k}$ and $t_i > t_{i_k}$ then
        $t_{i_{k+1}} \leftarrow \max\{t_i \mid f_i \in F \text{ and } s_i \leq t_{i_k}\}$;
    else
        $s_{\min} \leftarrow \min\{s_i \mid f_i \in F,\ s_i > t_{i_k}\}$;
        $t_{i_{k+1}} \leftarrow \max\{t_i \mid f_i \in F,\ s_i = s_{\min}\}$;
    $F'' \leftarrow F'' \cup \{f_{i_{k+1}}\}$;
    $k \leftarrow k + 1$;
$C_1 \leftarrow \{f_{i_j} \in F'' \mid j \text{ is odd}\}$;
$C_2 \leftarrow \{f_{i_j} \in F'' \mid j \text{ is even}\}$;
return $(C_1, C_2)$;
```

Algorithm 3: `ColOptimize`

```
input  : A set of pairs $\{(C_1^i(j), C_2^i(j))\}$, parameter $\tau$
output : A new set of pairs $\{(D_1^i(j), D_2^i(j))\}$
for $i \leftarrow 1$ to $4r_{\max}$ do
    for $k \leftarrow 1$ to $\tau$ do
        $D_1^i(k) \leftarrow \bigcup_{z=0}^{\lceil(\log c_{\max})/\tau\rceil - 1} C_1^i(z\tau + k)$;
        $D_2^i(k) \leftarrow \bigcup_{z=0}^{\lceil(\log c_{\max})/\tau\rceil - 1} C_2^i(z\tau + k)$;
return $\{(D_1^i(k), D_2^i(k)) : k = 1, ..., \tau \text{ and } i \in \{1, ..., 4r_{\max}\}\}$;
```

▶ **Lemma 8.** *Procedure* `rCover` *finds two feasible subsets C_1 and C_2 satisfying properties (P1) and (P2).*

▶ **Lemma 9.** *Procedure* `FlowDec` *partitions F_ℓ^M into at most $8r_{\max}(F_\ell^M)$ feasible subsets.*

3.2 Removing the log factor

In this subsection, we illustrate Procedure `ColOptimize` (see Algorithm 3), which removes the logarithmic factor by optimizing color usage. The result is a coloring with $O(r_{\max})$ colors.

Let τ be a constant to be determined later. Intuitively, the idea is to combine subsets of different levels in an alternating manner with τ serving as the granularity parameter. More precisely, let $C_a^i(j)$, where $a \in \{1, 2\}$, $j \in \{1, ..., \lceil \log c_{\max} \rceil\}$, and $i \in \{1, ..., 4r_{\max}\}$, denote the set of colors resulting from the execution of `FlowDec`. `ColOptimize` combines colors from different classes to reduce the number of colors by a factor of $\tau / \lceil \log c_{\max} \rceil$ resulting in $4\tau \cdot r_{\max}$ colors being used. Next, we show that setting $\tau = \log(1/(1 - \alpha)) + 2$ results in a valid coloring.

▶ **Lemma 10.** *For $\tau = \log(1/(1-\alpha)) + 2$, the sets $D^i_a(k)$, where $a \in \{1, 2\}$, $k \in \{1, ..., \tau\}$, and $i \in \{1, ..., 4r_{\max}\}$, constitute a valid coloring.*

The main result of this section now directly follows from Lemma 10.

▶ **Theorem 11.** *For any $\alpha \in (0, 1)$, there exists an offline $O(\log(1/1-\alpha))$-approximation algorithm for* Round-UFPP *with α-small flows. In particular, we have a constant-factor approximation for any constant $\alpha < 1$.*

4 Algorithms for general Round-UFPP instances

In what follows, we present offline and online algorithms for general instances of Round-UFPP. Our treatment of large flows involves a reduction from Round-UFPP to the rectangle coloring problem (RCOL) which is discussed in §4.1. Next, in §4.2, we design an online algorithm for the RCOL instances arising from the reduction. Later, in §4.3, we cover our online algorithm for Round-UFPP with $\frac{1}{4}$-large flows. Finally, in §4.4, we present our final algorithm for the general Round-UFPP instances.

4.1 The reduction from Round-UFPP with large flows to RCOL

▶ **Definition 12** (Rectangle Coloring Problem (RCOL).)**.** Given a collection $\mathcal{R}$ of n axis-parallel rectangles, the objective is to color the rectangles with the minimum number of colors such that rectangles of the same color are disjoint.

Each rectangle $R \in \mathcal{R}$ is given by a quadruple $(x^l(R), x^r(R), y^t(R), y^b(R))$ of real numbers, corresponding to the x-coordinates of its left and right boundaries and the y-coordinates of its top and bottom boundaries, respectively. More precisely, $R = \{(x, y) \mid x^l(R) \leq x \leq x^r(R)$ and $y^b(R) \leq y \leq y^t(R)\}$. When the context is clear, we may omit R and write x^l, x^r, y^t, y^b. Two rectangles R and R' are called compatible if they do not intersect each other; else, they are called incompatible.

The reduction from Round-UFPP with large flows to RCOL is based on the work in [10]. It starts by associating with each flow $f_i = (s_i, t_i, \sigma_i)$, a rectangle $R_i = (s_i, t_i, b_i, b_i - \sigma_i)$. If we draw the capacity profile over the path P, then R_i is a rectangle of thickness σ_i sitting under the curve touching the "ceiling." Let $\mathcal{R}(F)$ denote the set of rectangles thus associated with flows in F. We assume, without loss of generality, that rectangles do not intersect on their border; that is, all intersections are with respect to internal points. We begin with an observation stating that a disjoint set of rectangles constitutes a feasible set of flows.

▶ **Observation 13** ([10])**.** *Let $\mathcal{R}(F)$ be a set of disjoint rectangles corresponding to a set of flows F. Then, F is a feasible set of flows.*

The main result here is that if all flows in F are k-large then an optimal coloring of $\mathcal{R}(F)$ is at most a factor of $2k$ worse than the optimal solution to Round-UFPP instance arising from F. The following key lemma is crucial to the result.

▶ **Lemma 14** ([10])**.** *Let F be a feasible set of flows, and let $k \geq 2$ be an integer, such that every flow in F is $\frac{1}{k}$-large. Then there exists a $2k$ coloring of $\mathcal{R}(F)$.*

As an immediate corollary, we get the following.

▶ **Corollary 15.** *Let F be a feasible set of flows, and let $k \geq 2$ be an integer, such that every flow in F is $\frac{1}{k}$-large. Then, $\chi(\mathcal{R}(F)) \leq 2k\chi(F)$.*

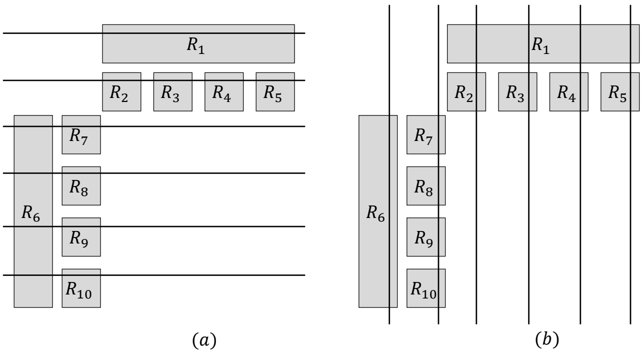


Figure 2 A collection $\mathcal{R}$ of 4-line-sparse rectangles. The lines can be either (a) horizontal or (b) vertical.

Proof. Consider an optimal coloring C of F with $\chi(F)$ colors. Apply Lemma 14 to each color class C_i, for $1 \leq i \leq \chi(F)$, to get a $2k$-coloring of $\mathcal{R}(C_i)$. The final result is a coloring of $\mathcal{R}(F)$ using at most $2k\chi(F)$ colors. ◀

We are ready to state the main result of this subsection.

▶ **Lemma 16.** *Suppose there exists an offline α-approximation (online α-competitive) algorithm $\mathfrak{A}$ for* RCOL*. Then, for every integer $k \geq 2$ there exists an offline $2k\alpha$-approximation (online $2k\alpha$-competitive) algorithm for* Round-UFPP *consisting of $\frac{1}{k}$-large flows.*

Proof. Given a set F of $\frac{1}{k}$-large flows for some integer $k \geq 2$, construct the set of associated rectangles $\mathcal{R}(F)$ and apply the algorithm $\mathfrak{A}$ to it. The solution is a valid Round-UFPP solution (Observation 13). Furthermore, by Corollary 15,

$$\mathfrak{A}(\mathcal{R}(F)) \leq \alpha\chi(\mathcal{R}(F)) \leq 2k\alpha\chi(F).$$

Finally, the reduction does not depend on future flows; hence, it is online in nature. ◀

4.2 Algorithms for RCOL

In this section, we consider algorithms for the rectangle coloring problem (RCOL). We begin by introducing a key notion measuring the sparsity of rectangles with respect to a set of lines. This is similar to the concept of point sparsity investigated by Chalermsook [12].

▶ **Definition 17** (s-line-sparsity)**.** A collection of rectangles $\mathcal{R}$ is *s-line-sparse* if there exists a set of axis-parallel lines $L_{\mathcal{R}}$ (called an *s-line-representative set* of $\mathcal{R}$), such that every rectangle $R \in \mathcal{R}$ is intersected by $k_R \in [1, s]$ lines in $L_{\mathcal{R}}$ (see Figure 2 for an example).

For simplicity, we assume that representative lines are all horizontal. The objective is to design an online $O(\log s)$-competitive algorithm for RCOL consisting of s-line-sparse rectangles. In the online setting, rectangles appear one by one; however, we assume that an s-line-representative set $L_{\mathcal{R}}$ is known in advance. As we will later see, this will not cause any issues since the RCOL instances considered here arise from Round-UFPP instances with large flows from which it is straightforward to compute s-line-representative sets. In the offline case, on the other hand, we get a $\log(n)$ approximation by (trivially) computing an n-line-representative set–associate to each rectangle an arbitrary line intersecting it. The remainder of this subsection is organized as follows. First, in §4.2.1, we consider the 2-line-sparse case. Later, in §4.2.2, we study the general s-line-sparse case.

4.2.1 The 2-line-sparse case

Consider a collection of rectangles $\mathcal{R}$ and a 2-line-representative set $L_{\mathcal{R}} = \{\ell_0, \ell_1, ..., \ell_k\}$ (that is, each rectangle R is intersected by either one or two lines in $L_{\mathcal{R}}$) where the rectangles in $\mathcal{R}$ appears in an online fashion. Recall, however, that the line set $L_{\mathcal{R}}$ is known in advance. Without loss of generality, assume that $y(\ell_0) < y(\ell_1) < ... < y(\ell_k)$.

For each $R \in \mathcal{R}$, let $T(R)$ denote the index of the topmost line in $L_{\mathcal{R}}$ that intersects R; $T(R) = \max\{i \mid \ell_i \text{ intersects } R\}$. Next, partition $\mathcal{R}$ into three subsets

$$\mathcal{R}_l = \{R \in \mathcal{R} \mid T(R) \equiv l \mod 3\}, \text{ for } l = 0, 1, 2. \tag{2}$$

The following lemma shows that each of the above subsets can be viewed as a collection of interval coloring problem (ICP) instances.

▶ **Lemma 18.** *Suppose two rectangles $R, R' \in \mathcal{R}$ belong to the same subset; that is, $R, R' \in \mathcal{R}_l$ for some $l \in \{0, 1, 2\}$. Then, the following are true.*

(1) *If $T(R) = T(R')$ and the projection of R and R' on the x-axis have a non-empty intersection, then $R \cap R' \neq \emptyset$.*

(2) *If $T(R) \neq T(R')$, then $R \cap R' = \emptyset$.*

We will use the optimal 3-competitive online algorithm due to Kierstead and Trotter for ICP [20]. The algorithm colors an instance of ICP of clique size ω with at most $3\omega - 2$ colors which matches the lower bound shown in the same paper. Henceforth, we refer to this algorithm as the KT algorithm.

Now we can present an $O(1)$-competitive online algorithm, named COL2SP, with a known 2-line-representative set. COL2SP computes a partition of $\mathcal{R}$ into $\mathcal{R}_0, \mathcal{R}_1$, and $\mathcal{R}_2$ as explained above. Then, it applies the KT algorithm to each subset. Note that COL2SP can be seen as executing multiple instances of the KT algorithm in parallel.

▶ **Lemma 19.** *Algorithm* COL2SP *is an online $O(1)$-competitive algorithm for* RCOL *on 2-line-sparse instances given prior knowledge of a 2-line-representative set for the incoming rectangles. Moreover,* COL2SP *uses at most $3 \cdot \omega(\mathcal{R})$ colors.*

4.2.2 The s-line-sparse case

Consider a set of s-line-sparse rectangles $\mathcal{R}$ and an s-line-representative set $L_{\mathcal{R}}$. Our goal in this subsection is to demonstrate a partitioning of $\mathcal{R}$ into $O(\log s)$ 2-line-sparse subsets, where each subset is accompanied by its own 2-line-representative set.

Given a set of lines L, we define the degree of a rectangle $R \in \mathcal{R}$, with respect to L, to be the number of lines in L that intersect R,

$$\text{Deg}_L(R) = |\{\ell \in L \mid \ell \cap R \neq \emptyset\}| .$$

We say that a rectangle $R \in \mathcal{R}$ is of level $l \geq 0$ with respect to $L_{\mathcal{R}}$, if $2^l \leq \text{Deg}_{L_{\mathcal{R}}}(R) < 2^{l+1}$. The partitioning is based on the level of rectangles. More precisely, $\mathcal{R}$ is partitioned into $\lceil \log s \rceil + 1$ “levels"

$$\text{Lev}(i) = \{R \in \mathcal{R} \mid R \text{ is of level } i\}, \text{ for } i = 0, 1, ..., \lceil \log s \rceil.$$

Next we show that each level is a 2-line-sparse set. To this end, we present a 2-line-representative set for each level. Let $L_{\mathcal{R}} = \{\ell_1, \ell_2, ..., \ell_k\}$ and define

$$S(i) = \{\ell_j \in L_{\mathcal{R}} \mid j \equiv 1 \mod 2^i\}, \text{ for } i \in \{0, ..., \lceil \log s \rceil.$$

```
Algorithm 4: RectCol
input  : A rectangle $R \in \mathcal{R}$
input  : The last state of RectCol; an $s$-representative-line set $L_{\mathcal{R}}$ for $\mathcal{R}$
output : A color for $R$
$i \leftarrow \text{argmin}_j (2^j \leq \text{Deg}_{L_{\mathcal{R}}}(R) < 2^{j+1})$;
$\text{Lev}(i) \leftarrow \text{Lev}(i) \cup \{R\}$;
return COL2SP$(R, L_{\mathcal{R}})$;
```

▶ **Lemma 20.** *For every* $i \in \{0, ..., \lceil \log s \rceil\}$*,* $\text{Lev}(i)$ *is a 2-line-sparse set and* $S(i)$ *is a 2-line-representative set for* $\text{Lev}(i)$*.*

We are ready to present an $O(\log s)$-competitive online algorithm, named RectCol, for RCOL with a known line-representative set. Algorithm RectCol works as follows (see Algorithm 4).

▶ **Lemma 21.** RectCol *is an online* $O(\log s)$*-competitive algorithm for* RCOL *with s-line-sparse rectangles, given a representative-line set. Moreover,* RectCol *uses* $O(\omega(\mathcal{R}) \cdot \log s)$ *colors.*

4.3 An algorithm for Round-UFPP with large flows

We are ready to present ProcLarges, an algorithm for Round-UFPP with large flows. For concreteness, we present the algorithm for $\frac{1}{4}$-large flows; this result can be easily generalized to α-large flows for any $\alpha \leq 1/2$.

The online algorithm we have designed for RCOL need to have access to an s-line-representative set $L_{\mathcal{R}}$ for the set of rectangles $\mathcal{R}$. In our case, these rectangles are constructed from flows (§4.1) which themselves arrive in an online fashion. However, all we need to be able to compute an s-line-representative set is the knowledge of the path over which the flows will be running–that is $P = (V, E)$ with capacities $\{c_e\}_{e \in E}$ (recall that we assume that $c_{\min} = 1$, which can always be achieved via scaling if needed). It is possible to construct (at least) three different s-line-representative sets for $\mathcal{R}$:

$\boldsymbol{L_1}$ A set of $s = \lceil \log_{4/3} c_{\max} \rceil + 1$ horizontal lines $L = \{l_0, l_1, ..., l_s\}$ where the y-coordinate of the ith line is $y(l_i) = (3/4)^i \cdot c_{\max}$. Note that ℓ_0 is the topmost line.

$\boldsymbol{L_2}$ A set of m vertical lines, one per edge in the path.

$\boldsymbol{L_3}$ A set of n axis-parallel lines, one per rectangle.

Note that L_3 is only useful in the offline setting. It is obvious that L_2 and L_3 are valid line-representative sets for $\mathcal{R}$. Below, we show that L_1 is valid as well.

▶ **Lemma 22.** L_1 *is an s-line-representative set for* $\mathcal{R}(\mathcal{F})$*.*

▶ **Theorem 23.** ProcLarges *is an* $O(\log \log c_{\max})$*-competitive algorithm for* Round-UFPP *with* $\frac{1}{4}$*-large flows. Furthermore, the bound can be improved to* $O(\min(\log m, \log \log c_{\max}))$*.*

Proof. ProcLarges executes algorithm RectCol on $\mathcal{R}(F)$ with a representative-line set $L = L_1$ of size $O(\log c_{\max})$. The colors returned by RectCol are used for the flows without modification. Now, setting $s = O(\log c_{\max})$, Lemma 21 states that Algorithm RectCol uses $O(\omega(\mathcal{R}(F)) \log \log c_{\max})$ colors. Lemma 16 completes the argument. Finally, note that running algorithm RectCol with $L = L_2$ as the representative-line set, we get a sparsity of $s = m$ and a coloring using $O(\omega(\mathcal{R}(F)) \log m)$ colors. To get the improved bound, we run the algorithm with $L = L_1$, if $\log c_{\max} \leq m$; else, we run it with $L = L_2$. ◀

4.4 Putting it together – The final algorithm

At this point, we have all the ingredients needed to present our final algorithm, `SolveRUFPP`, for Round-UFPP. `SolveRUFPP` simply uses procedure `ProcLarges` (§4.3) for $\frac{1}{4}$-large flows and procedure `ProcSmalls` for $\frac{1}{4}$-small flows. For `ProcSmalls`, we can use our algorithm in §3 or the 16-competitive algorithm in [15] in the offline case; and the 32-competitive algorithm in [16] in the online case.

▶ **Theorem 24.** *There exists an online* $O(\min(\log m, \log\log c_{\max}))$*-competitive algorithm and an offline* $O(\min(\log n, \log m, \log\log c_{\max}))$*-approximation algorithm for* Round-UFPP.

Proof. In the online case, `ProcSmalls` is a 32-competitive [16]. On the other hand, by Proposition 23, `ProcLarges` is an $O(\min(\log m, \log\log c_{\max}))$-competitive. Thus overall, algorithm `SolveRUFPP` is $O(\min(\log m, \log\log c_{\max}))$-competitive. In the offline case, since the set of flows $\mathcal{F}$ is known in advance, we can get a slightly better bound by using L_3 in §4.3 as the third line-representative set (of sparsity $s = n$). Thus we get the $O(\min(\log n, \log m, \log\log c_{\max}))$ bound by running the algorithm three times with L_1, L_2, and L_3 and using the best one. ◀

5 Concluding remarks

In this paper, we present improved offline approximation and online competitive algorithms for Round-UFPP. Our work leaves several open problems. First, is there an $O(1)$-approximation algorithm for offline Round-UFPP? Second, can we improve the competitive ratio achievable in the online setting to match the lower bound of $\Omega(\log\log\log c_{\max})$ shown in [16], or improve the lower bound? From a practical standpoint, it is important to analyze the performance of simple online algorithms such as First-Fit and its variants for Round-UFPP and RCOL. Another natural direction for future research is the study of Round-UFP and variants on more general graphs.

References

1 Udo Adamy and Thomas Erlebach. Online coloring of intervals with bandwidth. In Klaus Jansen and Roberto Solis-Oba, editors, *Approximation and Online Algorithms, First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003, Revised Papers*, volume 2909 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. doi:10.1007/978-3-540-24592-6_1.

2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing 2+eps approximation for unsplittable flow on a path. *CoRR*, abs/1211.2670, 2012.

3 Esther M. Arkin and Ellen B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1 – 8, 1987.

4 E. Asplund and B. Grünbaum. On a coloring problem. *Mathematica Scandinavica*, 8(0):181–188, 1960.

5 Yossi Azar, Amos Fiat, Meital Levy, and N. S. Narayanaswamy. An improved algorithm for online coloring of intervals with bandwidth. *Theor. Comput. Sci.*, 363(1):18–27, 2006. doi:10.1016/j.tcs.2006.06.014.

6 Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-ptas for unsplittable flow on line graphs. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 721–729. ACM, 2006. doi:10.1145/1132516.1132617.

7 Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 702–709. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496847.

8 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, September 2001.

9 Mark Bartlett, Alan M. Frisch, Youssef Hamadi, Ian Miguel, Armagan Tarim, and Chris Unsworth. The temporal knapsack problem and its solution. In Roman Barták and Michela Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*, volume 3524 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2005. doi:10.1007/11493853_5.

10 P. Bonsma, J. Schulz, and A. Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *FOCS'11*, pages 47–56, 2011.

11 Gruia Calinescu, Amit Chakrabarti, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Trans. Algorithms*, 7(4):48:1–48:7, September 2011.

12 Parinya Chalermsook. Coloring and maximum independent set of rectangles. *APPROX'11*, pages 123–134, 2011. URL: http://dx.doi.org/10.1007/978-3-642-22935-0_11.

13 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP'03*, pages 410–425, 2003.

14 Andreas Darmann, Ulrich Pferschy, and Joachim Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411(49):4217 – 4234, 2010.

15 Khaled M. Elbassioni, Naveen Garg, Divya Gupta, Amit Kumar, Vishal Narula, and Arindam Pal. Approximation algorithms for the unsplittable flow problem on paths and trees. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 267–275. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.267.

16 Leah Epstein, Thomas Erlebach, and Asaf Levin. Online capacitated interval coloring. *SIAM Journal on Discrete Mathematics*, 23(2):822–841, 2009.

17 Leah Epstein and Meital Levy. Online interval coloring and variants. In *ICALP'05*, pages 602–613, 2005.

18 N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

19 H. A. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM Journal on Discrete Mathematics*, 1(4):526–530, 1988.

20 H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

21 Alexandr Kostochka. Coloring intersection graphs of geometric figures with a given clique number. In *Contemporary Mathematics 342, AMS*, 2004.

22 Cynthia A. Phillips, R. N. Uma, and Joel Wein. Off-line admission control for general scheduling problems. In *Journal of Scheduling*, pages 879–888, 2000.

Structural Parameters, Tight Bounds, and Approximation for (k, r)-Center[*]

Ioannis Katsikarelis[1], Michael Lampis[2], and Vangelis Th. Paschos[3]

1 **Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France**
ioannis.katsikarelis@lamsade.dauphine.fr

2 **Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France**
michail.lampis@lamsade.dauphine.fr

3 **Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France**
paschos@lamsade.dauphine.fr

Abstract

In (k, r)-Center we are given a (possibly edge-weighted) graph and are asked to select at most k vertices (centers), so that all other vertices are at distance at most r from a center. In this paper we provide a number of tight fine-grained bounds on the complexity of this problem with respect to various standard graph parameters. Specifically:

- For any $r \geq 1$, we show an algorithm that solves the problem in $O^*((3r+1)^{\mathrm{cw}})$ time, where cw is the clique-width of the input graph, as well as a tight SETH lower bound matching this algorithm's performance. As a corollary, for $r = 1$, this closes the gap that previously existed on the complexity of Dominating Set parameterized by cw.
- We strengthen previously known FPT lower bounds, by showing that (k, r)-Center is W[1]-hard parameterized by the input graph's vertex cover (if edge weights are allowed), or feedback vertex set, even if k is an additional parameter. Our reductions imply tight ETH-based lower bounds. Finally, we devise an algorithm parameterized by vertex cover for unweighted graphs.
- We show that the complexity of the problem parameterized by tree-depth is $2^{\Theta(\mathrm{td}^2)}$ by showing an algorithm of this complexity and a tight ETH-based lower bound.

We complement these mostly negative results by providing *FPT approximation schemes* parameterized by clique-width or treewidth which work efficiently independently of the values of k, r. In particular, we give algorithms which, for any $\epsilon > 0$, run in time $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, $O^*((\mathrm{cw}/\epsilon)^{O(\mathrm{cw})})$ and return a $(k, (1+\epsilon)r)$-center, if a (k, r)-center exists, thus circumventing the problem's W-hardness.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases FPT algorithms, Approximation, Treewidth, Clique-width, Domination

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.50

1 Introduction

In this paper we study the (k, r)-Center problem: given a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{N}^+$ which satisfies the triangle inequality and defines the length of each

[*] A full version of the paper is available at [25], https://arxiv.org/abs/1704.08868.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 50; pp. 50:1–50:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

edge, we are asked if there exists a set K (the *center-set*) of at most k vertices of V, so that $\forall u \in V \setminus K$ we have $\min_{v \in K} d(v,u) \leq r$, where $d(v,u)$ denotes the shortest-path distance from v to u under weight function w. If w assigns weight 1 to all edges we say that we have an instance of un-weighted (k,r)-Center. (k,r)-Center is an extremely well-investigated optimization problem with numerous applications. It has a long history, especially from the point of view of approximation algorithms, where the objective is typically to minimize r for a given k [24, 46, 29, 19, 42, 31, 28, 1, 18]. The converse objective (minimizing k for a given r) has also been well-studied, with the problem being typically called r-Dominating Set in this case [11, 43, 36, 12].

Because (k,r)-Center generalizes Dominating Set (which corresponds to the case $r = 1$), the problem can already be seen to be hard, even to approximate (under standard complexity assumptions). In particular, the optimal r cannot be approximated in polynomial time by a factor better than 2 (even on planar graphs [19]), while k cannot be approximated by a factor better than $\ln n$ [39]. Because of this hardness, we are strongly motivated to investigate the problem's complexity when the input graph has some restricted structure.

In this paper our goal is to perform a complete analysis of the complexity of (k,r)-Center that takes into account this input structure by using the framework of parameterized complexity. In particular, we provide *fine-grained* upper and lower bound results on the complexity of (k,r)-Center with respect to the most widely studied parameters that measure a graph's structure: treewidth **tw**, clique-width **cw**, tree-depth **td**, vertex cover **vc**, and feedback vertex set **fvs**. In addition to the intrinsic value of determining the precise complexity of (k,r)-Center, this approach is further motivated by the fact that FPT algorithms for this problem have often been used as building blocks for more elaborate approximation algorithms [16, 18]. Indeed, (some of) these questions have already been considered, but we provide a number of new results that build on and improve the current state of the art. Along the way, we also close a gap on the complexity of the flagship Dominating Set problem parameterized by clique-width. Specifically, we prove the following:

- (k,r)-Center *can be solved (on unweighted graphs) in time $O^*((3r+1)^{cw})$ (if a clique-width expression is supplied with the input), but it cannot be solved in time $O^*((3r+1-\epsilon)^{cw})$ for any (fixed) $r \geq 1$, unless the Strong Exponential Time Hypothesis (SETH) [26, 27] fails.* The algorithmic result relies on standard techniques (dynamic programming on clique-width, fast subset convolution), as well as several problem-specific observations which are required to obtain the desired table size. The SETH lower bound follows from a direct reduction from SAT. A noteworthy consequence of our lower bound result is that, *for the case of* Dominating Set*, it closes the gap between the complexity of the best known algorithm ($O^*(4^{cw})$ [9]) and the best previously known lower bound ($O^*((3-\epsilon)^{cw})$ [35]).*
- (k,r)-Center *cannot be solved in time $n^{o(vc+k)}$ on edge-weighted graphs, or time $n^{o(fvs+k)}$ on unweighted graphs, unless the Exponential Time Hypothesis (ETH) is false.* It was already known that an FPT algorithm parameterized just by tw (for unbounded r) is unlikely to be possible [10]. These results show that the same holds for the two more restrictive parameters fvs and vc, even if k is also added as a parameter. They are (asymptotically) tight, since it is easy to obtain $O^*(n^{\text{fvs}})$, $O^*(n^{\text{vc}})$, and $O^*(n^k)$ algorithms. We remark that (k,r)-Center is a rare example of a problem that turns out to be hard parameterized by vc. We complement these lower bounds by an FPT algorithm for the unweighted case, running in time $O^*(5^{\text{vc}})$.
- (k,r)-Center *can be solved in time $O^*(2^{O(td^2)})$ for unweighted graphs, but if it can be solved in time $O^*(2^{o(td^2)})$, then the ETH is false.* Here the upper bound follows from known connections between a graph's tree-depth and its diameter, while the lower bound

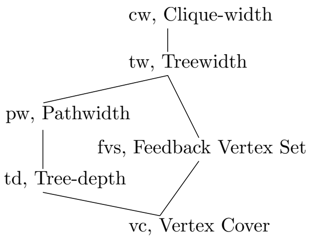


Figure 1 Relationships of parameters. Algorithmic results are inherited downwards, hardness results upwards.

Table 1 A summary of our results (theorem numbers) for all considered parameters. Initials u/w denote the unweighted/weighted variants of the problem.

	cw	tw	fvs	td	vc
FPT exact	3 (w/u)	10 (w/u)		7 (u)	6 (u)
FPT-AS	16 (w/u)	13 (w/u)			
SETH LB	1 (u)				
ETH LB			5 (w/u)	8 (u)	4 (w)
W[1]-hard			5 (w/u)		4 (w)

follows from a reduction from 3-SAT. We remark that this is a somewhat uncommon example of a parameterized problem whose parameter dependence turns out to be exponential in the *square* of the parameter.

These results, together with the recent work of [10] showing tight bounds of $O^*((2r+1)^{\mathrm{tw}})$ on the problem's complexity parameterized by tw, give a complete and often fine-grained, picture on (k,r)-Center for the most important graph parameters. One of the conclusions that can be drawn is that, as a consequence of the problem's hardness for vc (in the weighted case) and fvs, there are few cases where we can hope to obtain an FPT algorithm without bounding r: as r increases the complexity of exactly solving the problem quickly degenerates away from the case of Dominating Set, which is FPT for all considered parameters.

A further contribution of this paper is to complement this negative view by pointing out that it only applies if one insists on solving the problem *exactly*. If we allow algorithms that return a $(1+\epsilon)$-approximation to the optimal r, for arbitrarily small $\epsilon > 0$ and while respecting the given value of k, we obtain the following:

- *There exist algorithms which, for any $\epsilon > 0$, when given a graph that admits a (k,r)-center, return a $(k,(1+\epsilon)r)$-center in time $O^*((tw/\epsilon)^{O(tw)})$, or $O^*((cw/\epsilon)^{O(cw)})$, assuming a tree decomposition or clique-width expression is given in the input.*

The tw approximation algorithm is based on a technique introduced in [32], while the cw algorithm relies on a new extension of an idea from [23], which may be of independent interest. Thanks to these approximation algorithms, we arrive at an improved understanding of the complexity of (k,r)-Center by including the question of approximation, and obtain algorithms which continue to work efficiently even for large values of r. Figure 1 illustrates the relationships between parameters and Table 1 summarizes our results. We refer the reader to the full version [25] for all omitted definitions, constructions and proofs.

Related Work: Our work follows upon recent work by [10], which showed that (k,r)-Center can be solved in $O^*((2r+1)^{\mathrm{tw}})$, but not faster (under SETH), while its connected variant can be solved in $O^*((2r+2)^{\mathrm{tw}})$, but not faster. This paper in turn generalized previous results on Dominating Set for which a series of papers had culminated into an $O^*(3^{\text{tw}})$ algorithm [44, 2, 45], while on the other hand, [35] showed that an $O^*((3-\epsilon)^{\text{pw}})$ algorithm would violate the SETH, where pw denotes the input graph's pathwidth. The complexity of (k, r)-Center by the related parameter branchwidth had previously been considered in [16] where an $O^*((2r+1)^{\frac{3}{2}\text{bw}})$ algorithm is given. Moreover, [37] showed the problem parameterized by the number k of centers to be W[1]-hard in the L_∞ metric, in fact analysing Covering Points with Squares, a geometric variant. It remains W[2]-hard for 2-degenerate graphs [22]. On clique-width, a $O^*(4^{\text{cw}})$-time algorithm for Dominating Set was given in [9], while [41] notes that the lower bound of [35] for pathwidth/treewidth would also imply no $(3-\epsilon)^{\text{cw}} \cdot n^{O(1)}$-time algorithm exists for clique-width under SETH as well, since clique-width is at most 1 larger than pathwidth. For the edge-weighted variant, [20] shows that a $(2-\epsilon)$-approximation is W[2]-hard for parameter k and NP-hard for graphs of *highway dimension* $h = O(\log^2 n)$, while also offering a 3/2-approximation algorithm of running time $2^{O(kh \log(h))} \cdot n^{O(1)}$, exploiting the similarity of this problem with that of solving Dominating Set on graphs of bounded vc. Finally, for unweighted graphs, [34] provides efficient (linear/polynomial) algorithms computing $(r + O(\mu))$-dominating sets and $+O(\mu)$-approximations for (k, r)-Center, where μ is the *tree-breadth* or *cluster diameter* in a layering partition of the input graph, while [18] gives a polynomial-time bicriteria approximation scheme for graphs of bounded genus.

2 Definitions and Preliminaries

We use standard graph-theoretic notation. For a graph $G = (V, E)$, $n = |V|$ denotes the number of vertices, $m = |E|$ the number of edges and for a subset $X \subseteq V$, $G[X]$ denotes the graph induced by X. Further, we assume the reader has some familiarity with standard definitions from parameterized complexity theory, such as the classes FPT, W[1] (see [15, 21, 17]). For a parameterized problem with parameter k, an FPT-AS is an algorithm which for any $\epsilon > 0$ runs in time $O^*(f(k, \frac{1}{\epsilon}))$ (i.e. FPT time when parameterized by $k + \frac{1}{\epsilon}$) and produces a solution at most a multiplicative factor $(1+\epsilon)$ from the optimal (see [38]). We use $O^*(\cdot)$ to imply omission of factors polynomial in n.

In this paper we present approximation schemes with running times of the form $(\log n/\epsilon)^{O(k)}$. These can be seen to imply an FPT running time by a well-known win-win argument (see Lemma 1 in [25]): *If a parameterized problem with parameter k admits, for some $\epsilon > 0$, an algorithm running in time $O^*((\log n/\epsilon)^{O(k)})$, then it also admits an algorithm running in time $O^*((k/\epsilon)^{O(k)})$.*

Treewidth and *pathwidth* are standard notions in parameterized complexity which measure how close a graph is to being a tree or path (see [8, 5, 30]). We will also use the standard graph parameter of *clique-width*, which was introduced as a generalization of treewidth to dense graphs (see [13, 14]). Additionally, we will use the parameters *vertex cover number* and *feedback vertex set number* of a graph G, which are the sizes of the minimum vertex set whose deletion leaves the graph edgeless, or acyclic, respectively. Finally, we will consider the related notion of *tree-depth* [40], which is defined as the minimum height of a rooted forest whose completion (the graph obtained by connecting each node to all its ancestors) contains the input graph as a subgraph. We will denote these parameters for a graph G as $\text{tw}(G), \text{pw}(G), \text{cw}(G), \text{vc}(G), \text{fvs}(G)$, and $\text{td}(G)$, and will omit G if it is clear from the context. We recall the following well-known relations between these parameters [6, 14] which justify the hierarchy given in Figure 1: *For any graph G we have $tw(G) \leq pw(G) \leq td(G) \leq vc(G)$, $tw(G) \leq fvs(G) \leq vc(G)$, $cw(G) \leq pw(G) + 1$, and $cw(G) \leq 2^{tw(G)+1} + 1$.*

We also recall here the two main complexity assumptions used in this paper [26, 27]. The Exponential Time Hypothesis (ETH) states that 3-SAT cannot be solved in time $2^{o(n+m)}$ on instances with n variables and m clauses. The Strong Exponential Time Hypothesis (SETH) states that for all $\epsilon > 0$, there exists an integer k such that k-SAT cannot be solved in time $(2-\epsilon)^n$ on instances of k-SAT with n variables.

3 Clique-width

3.1 Lower bound based on SETH

The result of this section is that for any fixed constant $r \geq 1$, the existence of any algorithm for (k,r)-Center of running time $O^*((3r+1-\epsilon)^{\mathrm{cw}})$, for some $\epsilon > 0$, would imply the existence of some algorithm for SAT of running time $O^*((2-\delta)^n)$, for some $\delta > 0$.

Before we proceed, let us recall the high-level idea behind the SETH lower bound for Dominating Set given in [35], as well its generalization to (k,r)-Center given in [10]. In both cases the key to the reduction is the construction of long paths, which are conceptually divided into blocks of $2r+1$ vertices. The intended solution consists of selecting, say, the i-th vertex of a block of a path, and repeating this selection in all blocks of this path. This allows us to encode $(2r+1)^t$ choices, where t is the number of paths we make, which ends up being roughly equal to the treewidth of the construction. The reason this construction works in the converse direction is that, even though the optimal (k,r)-Center solution may “cheat” by selecting the i-th vertex of a block, and then the j-th vertex of the next, one can see that we must have $j \leq i$. Hence, by making the paths that carry the solution’s encoding long enough we can ensure that the solution eventually settles into a pattern that encodes an assignment to the original formula (which can be “read” with appropriate gadgets).

In our lower bound construction for clique-width we need to be able to “pack” more information per unit of width: instead of encoding $(2r+1)$ choices for each unit of treewidth, we need to encode $(3r+1)$ choices for each label. Our high-level plan to achieve this is to use *a pair* of long paths for each label. Because we only want to invest one label for each pair of paths we are forced to periodically (every $2r+1$ vertices) add cross edges between them, so that the connection between blocks can be performed with a single join operation (see the paths A_1, B_1 in Figure 2 for an illustation). Our plan now is to encode a solution by selecting a pair of vertices that will be repeated in each block, for example every i-th vertex of A_1 and every j-th vertex of B_1. One may naively expect that this would allow us to encode $(2r+1)^2$ choices for each label (which would lead to a SETH lower bound that would contradict the algorithm of Section 3.2). However, because of the cross edges, the optimal (k,r)-Center solution is not as well-behaved on a pair of cross-connected paths as it was on a path, and this makes it much harder to execute the converse direction of the reduction: a solution that takes every i-th vertex of A_1 could alternate repeatedly between various choices for B_1, because the selected vertices of A_1 also cover parts of B_1. Our strategy is therefore to identify $(3r+1)$ ordered selection pairs and show that any valid solution must be well-behaved with respect to these pairs. An overview of our construction, omitting most technical details of the reduction’s inner mechanism follows.

Construction overview: We construct a graph G, given some $\epsilon < 1$ and an instance ϕ of SAT with n variables and m clauses. We first choose an integer p, depending on ϵ and r (for technical reasons that become apparent in the proof of Theorem 1). Note that for the results of this section, both r and p are considered constants. We then group the variables of ϕ into $t = \lceil \frac{n}{\gamma} \rceil$ groups $F_1, \dots, F_t$, for $\gamma = \lfloor \log_2 (3r+1)^p \rfloor$, being also the maximum size of any such group. Our graph G will consist of t rows of $m(3rpt+1)$ gadgets $\hat{G}$, each row corresponding to one such group of variables. Each gadget $\hat{G}$ will contain p pairs of paths A_i, B_i and any selection of one vertex from each path will be associated with a specific partial assignment to the variables of the group. Gadgets of the same row will be connected in a path-like manner: for each $i \in [1, p]$ both final vertices of each pair A_i, B_i within each gadget will be connected to both first vertices of the corresponding pair A_i, B_i of the following gadget, with a global vertex h adjacent to all the first/last vertices of all such long paths, with an additional path of length r attached to h to ensure its selection in any minimum-sized center-set (and allowing for any selection in these first/last gadgets to be valid).

Furthermore, we will show that the possible selections of only one vertex from each path can be divided into $3r+1$ equivalence classes: we define $3r+1$ *canonical pairs* of numbers (α_y, β_y), indexed (and ordered) by $y \in [1, 3r+1]$, that give the indices of vertices from a pair of paths A_i, B_i (i.e. the α_y-th vertex of A_i and the β_y-th vertex of B_i) that would form the characteristic selection for each class, and show that any other selection within each class would be interchangeable (in terms of domination/coverage) with the characteristic selection, while if some pair with index y is used for selection of vertices from paths A_i, B_i in some gadget $\hat{G}^j_\tau$, then any pair used for the paths of the following gadget $\hat{G}^{j+1}_\tau$ (on the same row) must be of index $y' \leq y$. Observe that, as the path selections from each column must be well behaved with respect to our canonical pairs, there is an upper bound of $3r$ on the number of times the selection pattern can change on some pair of paths, giving $3rp$ for each row of gadgets and $3rpt$ times overall. In each gadget $\hat{G}$, we also make $3r+1$ vertices u^y_i for each pair A_i, B_i that signify these canonical selections from each path and further, a group of $(3r+1)^p$ vertices x_S for each set S that only contains one such u^y_i for each $i \in [1, p]$.

In this way, a selection of one vertex from each path A_i, B_i will correspond to a selection u^y_i, while all p such selections will correspond to one selection x_S that will in turn be associated with a partial assignment to the group of variables assigned to this row of gadgets (there are 2^γ partial assignments for each group and $(3r+1)^p \geq 2^\gamma$ sets S). Further, each column of gadgets will correspond to one clause, with the first m columns assigned to one clause each and $3rpt+1$ repetitions of this pattern giving the complete association. Our graph G will have one vertex $\hat{c}$ for each such column of gadgets (representing the associated clause) at distance r from vertices x_S in the gadgets $\hat{G}$ of its column that represent the partial assignments to the variables of the group associated with the gadget's row (and group F_τ) that would satisfy the clause (Figure 2 provides an illustration).

Thus, a satisfying assignment for ϕ will give a (k, r)-center for G by selecting in each gadget $\hat{G}$ all vertices corresponding to the partial assignment for its associated group of variables from each pair of paths, as well as the matching u^y_i and x_S vertices (and h). For the converse direction, as the number of changes of selection pattern is $\leq 3rpt$ and the number of columns is $m(3rpt+1)$, by the pigeonhole principle, there will always exist m consecutive columns for which the pattern does not change and thus we will be able to extract a consistent assignment for all clauses.

▶ **Theorem 1.** *For any fixed $r \geq 1$, if (k, r)-*Center *can be solved in $O^*((3r+1-\epsilon)^{cw(G)})$ time for some $\epsilon > 0$, then* SAT *can be solved in $O^*((2-\delta)^n)$ time for some $\delta > 0$.*

▶ **Corollary 2.** *If* Dominating Set *can be solved in $O^*((4-\epsilon)^{cw(G)})$ time for some $\epsilon > 0$, then* SAT *can be solved in $O^*((2-\delta)^n)$ time for some $\delta > 0$.*

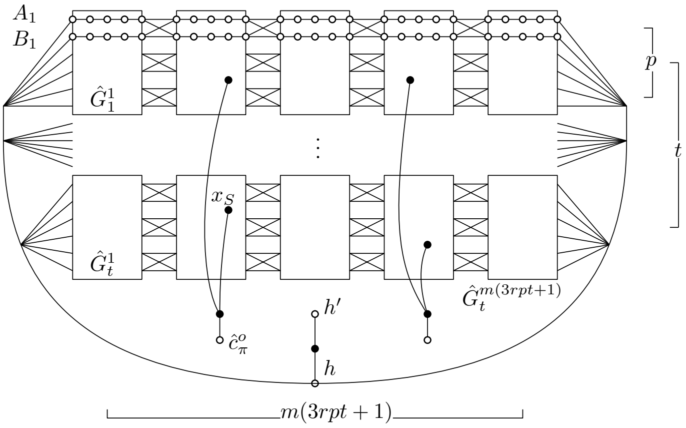


Figure 2 A simplified picture of the complete construction. Note boxes indicate block gadgets $\hat{G}$, while there is no vertex anywhere between h and the first/last vertices of the long paths.

3.2 Dynamic programming algorithm

We next present an $O^*((3r+1)^{\text{cw}})$-time dynamic programming (DP) algorithm for unweighted (k,r)-Center, using a given clique-width expression T_G for G with at most cw labels. Even though the algorithm relies on standard techniques, there are several non-trivial, problem-specific observations that we need to make to reach a DP table size of $(3r+1)^{\text{cw}}$.

Our first step is to re-cast the problem as a *distance-labeling* problem (not be confused with 'label'/'label-set' for a clique-width expression), that is, to formulate the problem as that of deciding for each vertex what is its precise distance to the optimal solution K. This is helpful because it allows us to make the constraints of the problem local, and hence easier to verify: roughly speaking, we say that a vertex is satisfied if it has a neighbor with a smaller distance to K. It is now not hard to design a clique-width based DP algorithm for this version of the problem: for each label l we need to remember two numbers, namely the smallest distance value given to some vertex with label l, and the smallest distance value given to a *currently unsatisfied* vertex with label l, if it exists.

The above scheme directly leads to an algorithm running in time (roughly) $((r+1)^2)^{\text{cw}}$. In order to decrease the size of this table, we now make the following observation: if a label-set contains a vertex at distance i from K, performing a join operation will satisfy all vertices that expect to be at distance $\geq i+2$ from K, since all vertices of the label-set will now be at distance at most 2. This implies that, in a label-set where the minimum assigned value is i, states where the minimum unsatisfied value is between $i+2$ and r are effectively equivalent. With this observation we can bring down the size of the table to $(4r)^{\text{cw}}$, because (intuitively) there are four cases for the smallest unsatisfied value: $i, i+1, \geq i+2$, and the case where all values are satisfied.

The last trick that we need to achieve the promised running time departs slightly from the standard DP approach. We will say that a label-set is *live* in a node of the clique-width expression if there are still edges to be added to the graph that will be incident to its vertices. During the execution of the dynamic program, we perform a "fore-tracking" step, by checking the part of the graph that comes higher in the expression to determine if a label-set is live. If it is, we merge the case where the smallest unsatisfied value is $i+2$, with the case where all values are satisfied (since a join operation will eventually be performed). Otherwise, a partial solution that contains unsatisfied vertices in a non-live label-set can safely be discarded. This brings down the size of the DP table to $(3r+1)^{\text{cw}}$, and then we need to use some further techniques to make the total running time quasi-linear in the size of the table. This involves counting the number of solutions instead of directly computing a solution of minimum size, as well as a non-trivial extension of fast subset convolution from [4] for a $3 \times (r+1)$-sized table (or *state-changes*, see [45, 9] and Chapter 11 of [15]).

▶ **Theorem 3.** *Given graph G, along with $k, r \in \mathbb{N}^+$ and clique-width expression T_G of clique-width cw for G, there exists an algorithm to solve the counting version of the (k, r)-*Center *problem in $O^*((3r+1)^{cw})$ time.*

4 Vertex Cover, Feedback Vertex Set and Tree-depth

In this section we first show that the edge-weighted variant of the (k, r)-Center problem parameterized by vc + k is W[1]-hard, and more precisely, that the problem does not admit a $n^{o(\text{vc}+k)}$ algorithm under the ETH. We give a reduction from k-Multicolored Independent Set.

This is a well-known W[1]-hard problem that cannot be solved in $n^{o(k)}$ under the ETH [15]. Using essentially the same reduction with that of Theorem 4, we obtain a similar hardness result for unweighted (k, r)-Center parameterized by fvs.

▶ **Theorem 4.** *The weighted (k, r)-*Center *problem is W[1]-hard parameterized by $vc + k$. Furthermore, if there is an algorithm for weighted (k, r)-*Center *running in time $n^{o(vc+k)}$ then the ETH is false.*

▶ **Corollary 5.** *The (k, r)-*Center *problem is W[1]-hard when parameterized by $fvs + k$. Furthermore, if there is an algorithm for weighted (k, r)-*Center *running in time $n^{o(fvs+k)}$, then the ETH is false.*

We next show that unweighted (k, r)-Center admits an algorithm running in time $O^*(5^{\text{vc}})$, in contrast to its weighted version (Theorem 4). We devise an algorithm that operates in two stages: first, it guesses the intersection of the optimal solution with the optimal vertex cover, and then it uses a reduction to Set Cover to complete the solution optimally.

▶ **Theorem 6.** *Given graph G, along with $k, r \in \mathbb{N}^+$ and a vertex cover of size vc of G, there exists an algorithm solving unweighted (k, r)-*Center *in $O^*(5^{vc})$ time.*

We next consider the un-weighted version of (k, r)-Center parameterized by td. Theorem 4 has estabilshed that weighted (k, r)-Center is W[1]-hard parameterized by vc (and hence also by td), but the complexity of unweighted (k, r)-Center parameterized by td does not follow from this theorem, since td is incomparable to fvs. Indeed, we show that (k, r)-Center is FPT parameterized by td and precisely determine its parameter dependence based on the ETH.

▶ **Theorem 7.** *Unweighted (k, r)-*Center *can be solved in time $O^*(2^{O(td^2)})$.*

▶ **Theorem 8.** *If (k, r)-*Center *can be solved in $2^{o(td^2)} \cdot n^{O(1)}$ time, then* 3-SAT *can be solved in $2^{o(n)}$ time.*

5 Treewidth: FPT approximation scheme

In this section we present an FPT approximation *scheme* (FPT-AS) for (k, r)-Center parameterized by tw. Given as input a weighted graph $G = (V, E)$, $k, r \in \mathbb{N}^+$ and an arbitrarily small error parameter $\epsilon > 0$, our algorithm is able to return a solution that uses a set of k centers K, such that all other vertices are at distance at most $(1+\epsilon)r$ from K, or to correctly conclude that no (k,r)-center exists. The running time of the algorithm is $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, which (for large r) significantly out-performs any *exact* algorithm for the problem (even for the unweighted case and more restricted parameters, as in Theorems 4 and 5), while only sacrificing a small ϵ error in the quality of the solution.

Our algorithm will rely heavily on a technique introduced in [32] (see also [3]) to approximate problems which are W-hard by treewidth. The idea is that, if the hardness of the problem is due to the fact that the DP table needs to store tw large numbers (in our case, the distances of the vertices in the bag from the closest center), we can significantly speed up the algorithm if we replace all entries by the closest integer power of $(1+\delta)$, for some appropriately chosen δ. This will reduce the table size from (roughly) r^{tw} to $(\log_{(1+\delta)} r)^{\mathrm{tw}}$.

The problem now is that a DP performing calculations on its entries will, in the course of its execution, create values which are not integer powers of $(1+\delta)$, and will therefore have to be "rounded" to retain the table size. This runs the risk of accumulating rounding errors, but we manage to show that the error on any entry of the rounded table can be bounded by a function of the height of its corresponding bag, then using a theorem of [7] stating that any tree decomposition can be balanced so that its width remains almost unchanged, yet its total height becomes $O(\log n)$. Beyond these ideas, which are for the most part present in [32], we will also need a number of problem-specific observations, such as the fact that we can pre-process the input by taking the metric closure of each bag, and in this way avoid some error-prone arithmetic operations.

To obtain the promised algorithm we thus do the following: first we re-cast the problem as a distance-labeling problem (as in the proof of Theorem 3) and formulate an exact treewidth-based DP algorithm running in time $O^*(r^{O(\mathrm{tw})})$. We remark that the algorithm essentially reproduces the ideas of [10], and can be made to run in $O^*((2r+1)^{\mathrm{tw}})$ if one uses fast subset convolution for the Join nodes (the naive implementation would need time $O^*((2r+1)^{2\mathrm{tw}}))$ but we give it here to ensure that we have a solid foundation upon which to build the approximation algorithm. We then apply the rounding procedure sketched above, and prove its approximation ratio by using the balancing theorem of [7] and indirectly comparing the value produced by the approximation algorithm with the value that would have been produced by the exact algorithm.

Distance-labeling: We give an equivalent formulation of (k,r)-Center that will be more convenient to work with in the remainder, similarly to Section 3.2. For an edge-weighted graph $G=(V,E)$, a distance-labeling function is a function $\mathrm{dl}: V \to \{0,\ldots,r\}$. We say that $u \in V$ is *satisfied* by dl, if $\mathrm{dl}(u)=0$, or if there exists $v \in N(u)$ such that $\mathrm{dl}(u) \geq dl(v) + w((v,u))$. We say that dl is *valid* if all vertices of V are satisfied by dl, and we define the *cost* of dl as $|\mathrm{dl}^{-1}(0)|$. The following lemma shows the equivalence between the two formulations:

▶ **Lemma 9.** *An edge-weighted graph $G=(V,E)$ admits a (k,r)-center if and only if it admits a valid distance-labeling function $dl: V \to \{0,\ldots,r\}$ with cost k.*

▶ **Theorem 10.** *There is an algorithm which, given an edge-weighted graph $G=(V,E)$ and $r \in \mathbb{N}^+$, computes the minimum cost of any valid distance labeling of G in time $O^*(r^{O(tw)})$.*

We now describe an approximation algorithm based on the exact DP algorithm of Theorem 10. We make use of a result of [7] stating that: *There is an algorithm which, given a tree decomposition of width w of a graph on n nodes, produces a decomposition of the same graph with width at most $3w+2$ and height $O(\log n)$ in polynomial time* and of the following lemma:

▶ **Lemma 11.** *Let $G = (V, E)$ be an edge-weighted graph, $\mathcal{T}$ a tree decomposition of G, and $u, v \in V$ two vertices that appear together in a bag of $\mathcal{T}$. Let G' be the graph obtained from G by adding (u, v) to E (if it does not already exist) and setting $w((u, v)) = d_G(u, v)$. Then $\mathcal{T}$ is a valid decomposition of G', and $\forall k, r$, G' admits a (k, r)-center if and only if G does.*

Let us also give an approximate version of the distance labeling problem we defined above, for a given error parameter $\epsilon > 0$. Let $\delta > 0$ be some appropriately chosen secondary parameter (we will eventually set $\delta \approx \frac{\epsilon}{\log n}$). We define a *$\delta$-labeling* function dl_δ as a function from V to $\{0\} \cup \{(1 + \delta)^i \mid i \in \mathbb{N}, (1 + \delta)^i \leq (1 + \epsilon)r\}$. In words, such a function assigns (as previously) a distance label to each vertex, with the difference that now all values assigned are integer powers of $(1 + \delta)$, and the maximum value is at most $(1 + \epsilon)r$. We now say that a vertex u is *ϵ-satisfied* if $\mathrm{dl}_\delta(u) = 0$ or, for some $v \in N(u)$ we have $\mathrm{dl}_\delta(u) \geq \mathrm{dl}_\delta(v) + \frac{w((v,u))}{1+\epsilon}$. As previously, we say that dl_δ is *valid* if all vertices are ϵ-satisfied, and define its cost as $|\mathrm{dl}_\delta^{-1}(0)|$. The following Lemma 12 shows the equivalence of a valid δ-labeling function of cost k and a $(k, (1 + \epsilon)^2 r)$-center for G and using it we conclude the proof of Theorem 13, stating the main result of this section.

▶ **Lemma 12.** *If for a weighted graph $G = (V, E)$ and any $k, r, \delta, \epsilon > 0$, there exists a valid δ-labeling function with cost k, then there exists a $(k, (1 + \epsilon)^2 r)$-center for G.*

▶ **Theorem 13.** *There is an algorithm which, given a weighted instance of (k, r)-Center, $[G, k, r]$, a tree decomposition of G of width tw and a parameter $\epsilon > 0$, runs in time $O^*((tw/\epsilon)^{O(tw)})$ and either returns a $(k, (1 + \epsilon))$-center of G, or correctly concludes that G has no (k, r)-center.*

6 Clique-width revisited: FPT approximation scheme

We give here an FPT-AS for (k, r)-Center parameterized by cw, both for un-weighted and for weighted instances (for a weighted definition of cw which we explain below). Our algorithm builds on the algorithm of Section 5, and despite the added generality of the parameterization by cw, we are able to obtain an algorithm with similar performance: for any $\epsilon > 0$, our algorithm runs in time $O^*((\mathrm{cw}/\epsilon)^{O(\mathrm{cw})})$ and produces a $(k, (1 + \epsilon)r)$-center if the input instance admits a (k, r)-center.

Our main strategy, which may be of independent interest, is to pre-process the input graph $G = (V, E)$ in such a way that the answer does not change, yet producing a graph whose tw is bounded by $O(\mathrm{cw}(G))$. The main insight that we rely on, which was first observed by [23], is that a graph of low cw can be transformed into a graph of low tw if one removes all large bi-cliques. Unlike previous applications of this idea (e.g. [33]), we do not use the main theorem of [23] as a "black box", but rather give an explicit construction of a tree decomposition, exploiting the fact that (k, r)-Center allows us to relatively easily eliminate complete bi-cliques. As a result, we obtain a tree decomposition of width not just bounded by some function of $\mathrm{cw}(G)$, but actually $O(\mathrm{cw}(G))$.

In the remainder we deal with the weighted version of (k, r)-Center. To allow clique-width expressions to handle weighted edges, we interpret the clique-width join operation η as taking three arguments. The interpretation is that $\eta(a, b, w)$ is an operation that adds (directed) edges from all vertices with label a to all vertices with label b and gives weight w to all these edges. It is not hard to see that if a graph has a (standard) clique-width expression with cw labels, it can also be constructed with cw labels in our context, if we replace every standard join operation $\eta(a, b)$ with $\eta(a, b, 1)$ followed by $\eta(b, a, 1)$. Hence, the algorithm we give also applies to un-weighted instances parameterized by (standard) clique-width. We will also deal with a generalization of (k, r)-Center, where we are also supplied, along with the input graph $G = (V, E)$, a subset $I \subseteq V$ of *irrelevant* vertices. In this version, a (k, r)-center is a set $K \subseteq V \setminus I$, with $|K| = k$, such that all vertices of $V \setminus I$ are at distance at most r from K. Clearly, the standard version of (k, r)-Center corresponds to $I = \emptyset$. As we explain in the proof of Theorem 16, this generalization does not make the problem significantly harder. In addition to the above, in this section we allow edge weights to be equal to 0. This does not significantly alter the problem, however, if we are interested in approximation and allow r to be unbounded, as the following lemma shows:

▶ **Lemma 14.** *There exists a polynomial algorithm which, for any $\epsilon > 0$, given an instance $I = [G, w, k, r]$ of (k, r)-Center, with weight function $w : V \to \mathbb{N}$, produces an instance $I' = [G, w', k, r']$ on the same graph with weight function $w' : V \to \mathbb{N}^+$, such that we have the following: for any $\rho \geq 1$, any $(k, \rho r')$-center of I' is a $(k, \rho r)$-center of I; any $(k, \rho r)$-center of I is a $(k, (1 + \epsilon)\rho r')$-center of I'.*

Our main tool is the following lemma, whose strategy is to replace every large label-set by two "representative" vertices, in a way that retains the same distances among all vertices of the graph. Applying this transformation repeatedly results in a graph with small treewidth. The main theorem of this section then follows from the above.

▶ **Lemma 15.** *Given a (k, r)-Center instance $G = (V, E)$ along with a clique-width expression T for G on cw labels, we can in polynomial time obtain a (k, r)-Center instance $G' = (V', E')$ with $V \subseteq V'$, and a tree decomposition of G' of width $tw = O(cw)$, with the following property: for all k, r, G has a (k, r)-center if and only if G' has a (k, r)-center.*

▶ **Theorem 16.** *Given $G = (V, E)$, $k, r \in \mathbb{N}^+$, clique-width expression T for G on cw labels and $\epsilon > 0$, there exists an algorithm that runs in time $O^*((cw/\epsilon)^{O(cw)})$ and either produces a $(k, (1 + \epsilon)r)$-center, or correctly concludes that no (k, r)-center exists.*

References

1 P.K. Agarwal and C.M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

2 J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *LATIN*, volume 2286 of *LNCS*, pages 613–628, 2002.

3 E. Angel, E. Bampis, B. Escoffier, and M. Lampis. Parameterized power vertex cover. In *WG*, volume 9941 of *LNCS*, pages 97–108, 2016.

4 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: fast subset convolution. In *STOC*, pages 67–74, 2007.

5 H.L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *WG*, volume 4271 of *LNCS*, pages 1–14. Springer, 2006.

6 H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

7 H.L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998.

8 H.L. Bodlaender and A.M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.

9 H.L. Bodlaender, E.J. van Leeuwen, J.M.M. van Rooij, and M. Vatshelle. Faster algorithms on branch and clique decompositions. In *MFCS*, volume 6281 of *LNCS*, pages 174–185, 2010.

10 G. Borradaile and H. Le. Optimal dynamic program for r-domination problems over tree decompositions. In *IPEC*, volume 63, pages 8:1–8:23, 2016.

11 A. Brandstädt and F.F. Dragan. A linear-time algorithm for connected r-domination and steiner tree on distance-hereditary graphs. *Networks*, 31(3):177–182, 1998.

12 R.S. Coelho, P.F. S. Moura, and Y. Wakabayashi. The k-hop connected dominating set problem: hardness and polyhedra. *Electronic Notes in Discrete Mathematics*, 50:59–64, 2015.

13 B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

14 B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

15 M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

16 E.D. Demaine, F.V. Fomin, M.T. 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.

17 R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

18 D. Eisenstat, P.N. Klein, and C. Mathieu. Approximating k-center in planar graphs. In *SODA*, pages 617–627, 2014.

19 T. Feder and D.H. Greene. Optimal algorithms for approximate clustering. In *STOC*, pages 434–444, 1988.

20 A.E. Feldmann. Fixed Parameter Approximations for k-Center Problems in Low Highway Dimension Graphs. In *ICALP*, volume 9135 of *LNCS*, 2015.

21 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

22 P.A. Golovach and Y. Villanger. Parameterized Complexity for Domination Problems on Degenerate Graphs. In *WG*, volume 5344 of *LNCS*, page 195–205, 2008.

23 F. Gurski and E. Wanke. The tree-width of clique-width bounded graphs without $K_{n,\ n}$. In *WG*, volume 1928 of *LNCS*, pages 196–205, 2000.

24 D.S. Hochbaum and D.B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986.

25 I.Katsikarelis, M. Lampis, and V.Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *CoRR*, abs/1704.08868, 2017.

26 R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

27 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

28 S. Khuller, R. Pless, and Y.J. Sussmann. Fault tolerant k-center problems. *Theor. Comput. Sci.*, 242(1-2):237–245, 2000.

29 S. Khuller and Y.J. Sussmann. The capacitated K-center problem. *SIAM J. Discrete Math.*, 13(3):403–418, 2000.

30 T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.

31 S.O. Krumke. On a generalization of the p-center problem. *Inf. Process. Lett.*, 56(2):67–71, 1995.

32 M. Lampis. Parameterized approximation schemes using graph widths. In *ICALP*, volume 8572 of *LNCS*, pages 775–786. Springer, 2014.

33 M. Lampis, K. Makino, V. Mitsou, and Y. Uno. Parameterized edge hamiltonicity. In *WG*, volume 8747 of *LNCS*, pages 348–359, 2014.

34 A. Leitert and F.F. Dragan. Parameterized approximation algorithms for some location problems in graphs. *CoRR*, abs/1706.07475v1, 2017.

35 D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA*, pages 777–789, 2011.

36 D. Lokshtanov, N. Misra, G. Philip, M.S. Ramanujan, and S. Saurabh. Hardness of r-dominating set on graphs of diameter (r + 1). In *IPEC*, volume 8246 of *LNCS*, pages 255–267, 2013.

37 D. Marx. Efficient approximation schemes for geometric problems? In *ESA*, volume 3669 of *LNCS*, pages 448–459, 2005.

38 D. Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

39 D. Moshkovitz. The projection games conjecture and the NP-hardness of ln n-approximating set-cover. *Theory of Computing*, 11:221–235, 2015.

40 J. Nesetril and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.

41 S. Oum, S.H. Sæther, and M. Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theor. Comput. Sci.*, 535:16–24, 2014.

42 R. Panigrahy and S. Vishwanathan. An o(log* n) approximation algorithm for the asymmetric *p*-center problem. *J. Algorithms*, 27(2):259–268, 1998.

43 P.J. Slater. R-domination in graphs. *J. ACM*, 23(3):446–450, 1976.

44 J.A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *WADS*, volume 709 of *LNCS*, pages 610–621, 1993.

45 J.M.M. van Rooij, H.L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, volume 5757 of *LNCS*, pages 566–577, 2009.

46 V.V. Vazirani. *Approximation algorithms.* Springer, 2001.

Optimal Matroid Partitioning Problems[*]

Yasushi Kawase[†1], Kei Kimura[‡2], Kazuhisa Makino[§3], and Hanna Sumita[¶4]

1 **Tokyo Institute of Technology, Tokyo, Japan**
`kawase.y.ab@m.titech.ac.jp`
2 **Toyohashi University of Technology, Toyohashi, Japan**
`kimura@cs.tut.ac.jp`
3 **Kyoto University, Kyoto, Japan**
`makino@kurims.kyoto-u.ac.jp`
4 **National Institute of Informatics, Tokyo, Japan**
`sumita@nii.ac.jp`

Abstract

This paper studies optimal matroid partitioning problems for various objective functions. In the problem, we are given a finite set E and k weighted matroids $(E, \mathcal{I}_i, w_i)$, $i = 1, \dots, k$, and our task is to find a minimum partition $(I_1, \dots, I_k)$ of E such that $I_i \in \mathcal{I}_i$ for all i. For each objective function, we give a polynomial-time algorithm or prove NP-hardness. In particular, for the case when the given weighted matroids are identical and the objective function is the sum of the maximum weight in each set (i.e., $\sum_{i=1}^{k} \max_{e \in I_i} w_i(e)$), we show that the problem is strongly NP-hard but admits a PTAS.

1998 ACM Subject Classification G.2.1 Combinatorial algorithms

Keywords and phrases Matroids, Partitioning problem, PTAS, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.51

1 Introduction

The *matroid partitioning problem* is one of the most fundamental problems in combinatorial optimization. In this problem, we are given a finite set E and k matroids $(E, \mathcal{I}_i)$, $i = 1, \dots, k$, and our task is to find a partition $(I_1, \dots, I_k)$ of E such that $I_i \in \mathcal{I}_i$ for all i. We say that such a partition $(I_1, \dots, I_k)$ of E is *feasible*. The matroid partitioning problem has been eagerly studied in a series of papers investigating structures of matroids. See, e.g., [7, 8, 9, 16, 23] for details. In this paper, we study weighted versions of the matroid partitioning problem. Namely, we assume that each matroid $(E, \mathcal{I}_i)$ has a weight function $w_i : E \to \mathbb{R}_+$. We consider several possible objective functions of the matroid partitioning problem.

Let $\mathrm{Op}^{(1)}$ and $\mathrm{Op}^{(2)}$ denote two mathematical operators taken from $\{\max, \min, \sum\}$. For any partition $P = (I_1, \dots, I_k)$ of E, we call $\mathrm{Op}^{(1)}{}_{i=1,\dots,k} \mathrm{Op}^{(2)}{}_{e \in I_i} w_i(e)$ the $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)})$-*value* of P. For example, $(\sum, \min)$-value of P denotes $\sum_{i=1,\dots,k} \min_{e \in I_i} w_i(e)$.

We define the *minimum* $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)})$-*value matroid partitioning problem* as the one for finding a feasible partition with minimum $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)})$-value. The maximum problems are

[*] A full version of the paper is available at [14], `https://arxiv.org/abs/1710.00950`.
[†] The first author is supported by JSPS KAKENHI Grant Number JP16K16005.
[‡] The second author is supported by JSPS KAKENHI Grant Number JP15H06286.0
[§] The third author is supported by JSPS KAKENHI Grant Number JP24106002, JP25280004, JP26280001, and JST CREST Grant Number JPMJCR1402, Japan.
[¶] The forth author is supported by JST ERATO Grant Number JPMJER1201, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 51; pp. 51:1–51:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

defined analogously. These matroid partitioning problems are natural to study, and have many applications in various areas such as scheduling and combinatorial optimization. We note that all the matroids and/or all the weights may be identical in case such as scheduling with identical machines.

The minimum $(\sum, \sum)$-value matroid partitioning problem is reducible to the *weighted matroid intersection problem*, and vice versa [8]. Here, the weighted matroid intersection problem is to find a maximum weight subset that is simultaneously independent in two given matroids. It is known that this problem is polynomially solvable, and many papers have worked on algorithmic aspects of this problem [16, 23]. Generalizations of the weighted matroid intersection problem have also been studied [17, 21, 18].

Special cases of the minimum $(\max, \sum)$-value matroid partitioning problem have been extensively addressed in the scheduling literature under the name of the minimum makespan scheduling. Since this problem is NP-hard, many papers have proposed polynomial-time approximation algorithms. We remark that most papers focused on subclasses of matroids as inputs: for example, free matroids [19, 24], partition matroids [25, 26, 20], uniform matroid [15, 1, 4], and general matroids [26]. Approximation algorithms for the maximum $(\min, \sum)$-value matroid partitioning problem are also well-studied, see, e.g., [3, 13, 25, 20].

The other matroid partitioning problems also have many applications, and yet they are not much studied especially for general matroids. We here describe some examples of applications.

Maximum total capacity spanning tree partition Assume that we are given an undirected weighted graph $G = (V, E; w)$, which can be partitioned into k edge-disjoint spanning trees. The maximum total capacity spanning tree partition problem is to compute a partition of the edges into k edge-disjoint spanning trees such that the total of the minimum weight in each spanning tree is maximized. Then, the problem can be written as the maximum $(\sum, \min)$-value matroid partitioning problem having k identical graphic matroids, where the $(\sum, \min)$-value is $\sum_{i=1}^{k} \min_{e \in I_i} w(e)$.

Minimum total memory of a scheduling In this problem we are also given n jobs E and k identical machines, and each job needs to be scheduled on exactly one machine. In addition, we are given size $s(e)$ of job $e \in E$. The set of feasible allocation for each machine i is represented by a family of independent sets $\mathcal{I}_i$ of a matroid. The goal of the problem is to minimize the total memory needed, i.e., $(\sum, \max)$-value $\sum_{i=1}^{k} \max_{e \in I_i} s(e)$.

Burkard and Yao [2] showed that the minimum $(\sum, \max)$-value matroid problem can be solved by a greedy algorithm for a subclass of matroids, which includes partition matroids. Dell'Olmo et al. [5] investigated optimal matroid partitioning problems where the input matroids are identical partition matroids.

The goal of our paper is to analyze the computational complexity of these matroid partitioning problems for general matroids.

Our results

We first show that the maximization problems can be reduced to the minimization problems. For example, the maximum $(\sum, \min)$-value matroid partitioning problem can be transformed to the minimum $(\sum, \max)$-value matroid partitioning problem. Hence, we focus only on the minimization problems.

Our main result is to analyze the computational complexity of the minimum $(\sum, \max)$-value matroid partitioning problem. This problem contains the maximum total capacity spanning tree partitioning problem and the minimum total memory scheduling problem. We first show that the problem is strongly NP-hard even when the matroids and weights

Table 1 The time complexity of the optimal matroid partitioning problems (the results of the paper are in bold). Identical case means $\mathcal{I}_1 = \cdots = \mathcal{I}_k$ and $w_1 = \cdots = w_k$.

objective	identical case	general case	reference
(Σ, Σ)	P	P	[8, 11]
$(\max, \Sigma)$	SNP-hard	SNP-hard	[12]
$(\Sigma, \max)$	**PTAS** **SNP-hard**	**εk-approx.** **NP-hard even for $o(\log k)$-approx.**	Section 3
$(\min, \min)$	**P**	**P**	Section 4
$(\max, \max)$	**P**	**P**	Section 4
$(\min, \max)$	**P**	**P**	Section 4
$(\min, \Sigma)$	**P**	**P**	Section 4
$(\max, \min)$	**P**	**NP-hard even to approximate**	Section 4
$(\Sigma, \min)$	**P**	**NP-hard even to approximate**	Section 4

are respectively identical. However, for such instances, we also propose a *polynomial-time approximation scheme* (PTAS), i.e., a polynomial-time algorithm that outputs a $(1+\varepsilon)$-approximate solution for each fixed $\varepsilon > 0$. Our PTAS computes an approximate solution by two steps: guess the maximum weight in each I_i^* for an optimal solution $(I_1^*, \ldots, I_k^*)$, and check the existence of such a feasible partition. We remark that the number of possible combinations of maximum weights is $|E|^k$ and it may be too large. To reduce the possibility, we use rounding techniques in the design of the PTAS. First, we guess the maximum weight in I_i^* for only s indices. Furthermore, we round the weight of each element and reduce the number of different weights to a small number r. Then, now we have r^s possibilities. To obtain the approximation ratio $(1+\varepsilon)$, we need to set r and s to be $\Omega(\log k)$ respectively, and hence the number of possibilities r^s is still large. Our idea to tackle this is to enumerate *sequences* of maximum weights in the nonincreasing order. This enables us to reduce the number of possibilities to $\binom{r+s-1}{r}$ $(\leq 2^{r+s-1})$. This implies that our algorithm is a PTAS.

Moreover, for the $(\sum, \max)$ case with general inputs, we provide an εk-approximation algorithm for any $\varepsilon > 0$. The construction is similar to the identical case. We also prove the NP-hardness even to approximate the problem within a factor of $o(\log k)$.

For the $(\min, \min)$, $(\max, \max)$, $(\min, \max)$, and $(\min, \sum)$ cases, we provide polynomial-time algorithms. The main idea of these algorithms is a reduction to the feasibility problem of the matroid partitioning problem. For the $(\max, \min)$ and $(\sum, \min)$ cases, we give polynomial-time algorithms when the matroids and weights are respectively identical, and prove strong NP-hardness even to approximate for the general case. These results are summarized in Table 1 with their references.

Due to the space limitation, we omit proofs of some results, which are found in [14].

2 Preliminaries

A *matroid* is a set system $(E, \mathcal{I})$ with the following properties: (I1) $\emptyset \in \mathcal{I}$, (I2) $X \subseteq Y \in \mathcal{I}$ implies $X \in \mathcal{I}$, and (I3) $X, Y \in \mathcal{I}$, $|X| < |Y|$ implies the existence of $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{I}$. A set $I \subseteq \mathcal{I}$ is said to be *independent*, and an inclusion-wise maximal independent set is called a *base*. We denote the set of bases of $(E, \mathcal{I})$ by $B(\mathcal{I})$. All bases of a matroid have the same cardinality, which is called the *rank* of the matroid and is denoted by $\mathrm{rank}(\mathcal{I})$. For any $B_1, B_2 \in B(\mathcal{I})$ and $e_1 \in B_1 \setminus B_2$, there exists $e_2 \in B_2 \setminus B_1$ such that $B_1 - e_1 + e_2 \in B(\mathcal{I})$ and $B_2 - e_2 + e_1 \in B(\mathcal{I})$.

For a matroid $(E, \mathcal{I})$, a subset $A \subseteq E$, and a nonnegative integer $l \in \mathbb{Z}_+$, define $\mathcal{I}|A = \{X : A \supseteq X \in \mathcal{I}\}$, $\mathcal{I} \setminus A = \{X \setminus A : X \in \mathcal{I}\}$, $\mathcal{I}/A = \{X \subseteq E \setminus A : \mathrm{rank}(X \cup A) - \mathrm{rank}(A) = |X|\}$, and $\mathcal{I}^{(l)} = \{X \in \mathcal{I} : |X| \leq l\}$. We call $(A, \mathcal{I}|A)$, $(E \setminus A, \mathcal{I} \setminus A)$, $(E \setminus A, \mathcal{I}/A)$, and $(E, \mathcal{I}^{(l)})$, respectively, the *restriction*, *deletion*, *contraction*, and *truncation* of $(E,\mathcal{I})$. It is well known that $(A,\mathcal{I}|A)$, $(A,\mathcal{I}\setminus A)$, $(E\setminus A,\mathcal{I}/A)$, and $(E,\mathcal{I}^{(l)})$ are all matroids. Given matroids $\mathcal{M}_1=(E_1,\mathcal{I}_1)$ and $\mathcal{M}_2=(E_2,\mathcal{I}_2)$, we define the *matroid union*, denoted by $\mathcal{M}_1\vee\mathcal{M}_2$, to be $(E_1\cup E_2,\mathcal{I}_1\vee\mathcal{I}_2)$ where $\mathcal{I}_1\vee\mathcal{I}_2=\{I_1\cup I_2 : I_1\in\mathcal{I}_1,\ I_2\in\mathcal{I}_2\}$. Any matroid union is also a matroid.

2.1 Model

Throughout the paper, we assume that every matroid is given by an independence oracle, which checks whether a given set is independent. Let k be a positive integer. We denote $[k]=\{1,\dots,k\}$. Let $(E,\mathcal{I}_i)$ be a matroid and $w_i : E\to\mathbb{R}_+$ be a nonnegative weight function for $i\in[k]$. We denote $n=|E|$. For any k sets $I_1,\dots,I_k\subseteq E$, we call $(I_1,\dots,I_k)$ a *feasible partition* of E if it satisfies that $\bigcup_{i\in[k]} I_i=E$, $I_i\neq\emptyset$ $(\forall i\in[k])$[1], $I_i\cap I_j=\emptyset$ $(\forall i,j\in[k],\ i\neq j)$, and $I_i\in\mathcal{I}_i$ $(\forall i\in[k])$. In particular, $(I_1,\dots,I_k)$ is said to be a *base partition* if it is a feasible partition and $I_i\in B(\mathcal{I}_i)$ for all $i\in[k]$. For two operators $\mathrm{Op}^{(1)}\in\{\max,\min,\sum\}$ and $\mathrm{Op}^{(2)}\in\{\max,\min,\sum\}$, we define the $(\mathrm{Op}^{(1)},\mathrm{Op}^{(2)})$-value of a feasible partition $(I_1,\dots,I_k)$ as $\mathrm{Op}^{(1)}{}_{i\in[k]}\,\mathrm{Op}^{(2)}{}_{e\in I_i}\,w_i(e)$. In this article, we study the following minimization problem:

$$\min\nolimits_{(I_1,\dots,I_k):\text{ feasible partition}} \mathrm{Op}^{(1)}{}_{i\in[k]}\,\mathrm{Op}^{(2)}{}_{e\in I_i}\,w_i(e).$$

We refer to the problem as the *minimum* $(\mathrm{Op}^{(1)},\mathrm{Op}^{(2)})$*-value matroid partitioning problem.* We write a problem instance as $(E,(\mathcal{I}_i,w_i)_{i\in[k]})$. If $(\mathcal{I}_i,w_i)$ are identical for all $i\in[k]$, we write $(E,(\mathcal{I},w),k)$. For the identical case, we can consider the partitioning problem where k is also a variable. This problem can be solved by solving $(E,(\mathcal{I},w),i)$ for $i=1,\dots,n$. Thus it suffices to focus on the problem where k is given.

It is known to be easy to decide whether there exists a feasible partition or not. Moreover, the minimum $(\sum,\sum)$-value matroid partitioning problem can be solved in polynomial time. These facts are useful to show our results later.

▶ **Theorem 1** ([8, 11])**.** *There exists a polynomial-time algorithm that decides whether or not there exists a feasible partition for any given matroids $(E,\mathcal{I}_1),\dots,(E,\mathcal{I}_k)$. Moreover, if it exists, we can find a feasible partition with minimum $(\sum,\sum)$-value in polynomial time.*

2.2 Basic properties

In this subsection, we prove basic properties of the partitioning problems. These properties imply that the minimization and maximization versions of matroid partitioning problems can be reduced to each other.

We first observe that we only need to consider base partitioning problems. Let $\mathcal{M}_i=(E,\mathcal{I}_i)$ be a matroid for $i\in[k]$. We add dummy elements so that any feasible partition is a base partition. To describe this precisely, we denote $r=\sum_{i\in[k]}\mathrm{rank}(\mathcal{I}_i)-|E|$. We remark that $r\geq 0$ if E has a feasible partition, since $|E|=\sum_{i\in[k]}|I_i|\leq\sum_{i\in[k]}\mathrm{rank}(\mathcal{I}_i)$ holds for any feasible partition $(I_1,\dots,I_k)$. Then let $D=\{d_1,\dots,d_r\}$ be a set of dummy elements. Note that $E\cap D=\emptyset$. We define two matroids $\mathcal{M}'_i=(D,\mathcal{I}'_i)$ and $\overline{\mathcal{M}}_i=(E\cup D,\overline{\mathcal{I}}_i)$ for each $i\in[k]$ by $\mathcal{I}'_i=\{D'\subseteq D : |D'|\leq\mathrm{rank}(\mathcal{I}_i)-1\}$ and $\overline{\mathcal{I}}_i=\{I\cup D' : I\in\mathcal{I}_i,\ D'\in\mathcal{I}'_i,\ |I\cup D'|\leq\mathrm{rank}(\mathcal{I}_i)\}$.

[1] We remark that the condition $I_i\neq\emptyset$ $(\forall i\in[k])$ is imposed to make the objective function well-defined. Moreover, if we define $\max_{e\in\emptyset} w_i(e)=0$, $\min_{e\in\emptyset} w_i(e)=\infty$, and $\sum_{e\in\emptyset} w_i(e)=0$, then we can reduce the problem where empty sets are allowed to our problem by adding dummy elements.

Namely, $\mathcal{M}'_i$ is a uniform matroid of rank $(\mathrm{rank}(\mathcal{I}_i) - 1)$, and $\overline{\mathcal{M}}_i$ is the $\mathrm{rank}(\mathcal{I}_i)$-truncation of the matroid union $\mathcal{M}_i \vee \mathcal{M}'_i$. Then, we have the following proposition.

▶ **Proposition 2.** *For any $(E, (\mathcal{I}_i, w_i)_{i\in[k]})$, its minimum $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)})$-value is the same as the minimum $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)})$-value for $(E \cup D, (\overline{\mathcal{I}}_i, \overline{w}_i)_{i\in[k]})$, where*

$$\overline{w}_i(e) = \begin{cases} w_i(e) & (e \in E), \\ \min_{e\in E} w_i(e) & (e \in D,\ \mathrm{Op}^{(2)} = \max), \\ \max_{e\in E} w_i(e) & (e \in D,\ \mathrm{Op}^{(2)} = \min), \\ 0 & (e \in D,\ \mathrm{Op}^{(2)} = \sum). \end{cases}$$

We remark that the same property holds for the maximization problem.

In the following, we assume $|E| = \sum_{i\in[k]} \mathrm{rank}(\mathcal{I}_i)$. We next show that the maximization problems are reducible to the minimization ones.

▶ **Proposition 3.** *For any feasible partition $(I_1, \dots, I_k)$ for $(E, \mathcal{I}_i)_{i\in[k]}$, it is an optimal solution for the minimum $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)})$-value matroid partitioning problem instance $(E, (\mathcal{I}_i, w_i)_{i\in[k]})$ if and only if it is optimal for the maximum $(\widetilde{\mathrm{Op}^{(1)}}, \widetilde{\mathrm{Op}^{(2)}})$-value matroid partitioning problem instance $(E, (\mathcal{I}_i, w'_i)_{i\in[k]})$, where $w^{\max} = \max_{i\in[k]} \max_{e\in E} w_i(e)$,*

$$\begin{cases} \widetilde{\min} = \max, \\ \widetilde{\max} = \min, \\ \widetilde{\sum} = \sum \end{cases} \quad \textit{and} \quad w'_i(e) = \begin{cases} \frac{|E|\cdot w^{\max}}{\mathrm{rank}(\mathcal{I}_i)} - w_i(e) & (\mathrm{Op}^{(1)} \in \{\min, \max\}, \mathrm{Op}^{(2)} = \sum), \\ w^{\max} - w_i(e) & (\textit{otherwise}). \end{cases}$$

We note that these reductions above are not approximation factor preserving. Hence, the (in)approximability of the maximization problems are not deduced from that of the minimization problems.

3 The minimum $(\sum, \max)$-value matroid partitioning problem

In this section, we study the minimum $(\sum, \max)$-value matroid partitioning problem. We first deal with the case where the matroids and weights are respectively identical and then go to the general case.

3.1 Strong NP-hardness of the identical case

We first prove that the minimum $(\sum, \max)$-value matroid partitioning problem is strongly NP-hard even if the matroids and weights are respectively identical.

To prove this, we use the *densest l-subgraph* problem, which is known to be strongly NP-hard [10]. The densest l-subgraph problem is, given a graph G and an integer l, to find a subgraph of G induced on l vertices that contains the largest number of edges.

In our reduction, we use the following property on a partition matroid. Let $(E, \mathcal{I})$ be a partition matroid defined by $\mathcal{I} = \{I : |I \cap S_i| \leq \eta_i\ (i \in [p])\}$, where $(S_1, \dots, S_p)$ is a partition of E, and $\eta_1, \dots, \eta_p$ are positive integers. In addition, we assume that $|S_i| = \eta_i \cdot k$ for each $i \in [p]$ so that E can be partitioned into k bases of $\mathcal{I}$. Then, for any weight w, we can construct greedily an optimal partition to the instance $(E, (\mathcal{I}, w), k)$ of the minimum $(\sum, \max)$-value matroid partitioning problem.

▶ **Lemma 4** ([2]). *Let $(E, \mathcal{I})$ be any partition matroid with $|S_i| = \eta_i \cdot k$ $(\forall i \in [p])$, and let w be any weight. Let $I_{i,j}$ consist of η_i elements with the η_i largest weights in $S_i \setminus (\bigcup_{h=1}^{j-1} I_{i,h})$. Then $(\bigcup_{i\in[p]} I_{i,1}, \dots, \bigcup_{i\in[p]} I_{i,k})$ is an optimal solution to $(E, (\mathcal{I}, w), k)$.*

Table 2 The weight of each element e_{ij}, where each row corresponds to i and each column corresponds to j.

$i\backslash j$	1	$\cdots$	$l-1$	l	$\cdots$	$l+2t-2$	$l+2t-1$	$l+2t$	$\cdots$	$l+2m-1$	$l+2m$	$\cdots$	$n+2m-1$
1	0	$\cdots$	0	0		$t-1$	$t-1$	t			m	$\cdots$	m
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$t-1$	$t-1$	t			$\vdots$		$\vdots$
u_t	$\vdots$		$\vdots$	$\vdots$		$t-1$	t	t			$\vdots$		$\vdots$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$t-1$	$t-1$	t			$\vdots$		$\vdots$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$t-1$	$t-1$	t			$\vdots$		$\vdots$
v_t	$\vdots$		$\vdots$	$\vdots$		$t-1$	t	t			$\vdots$		$\vdots$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$t-1$	$t-1$	t			$\vdots$		$\vdots$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$			$\vdots$		$\vdots$
n	0	$\cdots$	0	0		$t-1$	$t-1$	t			m	$\cdots$	m
$n+1$	0	$\cdots$	0	0	$\cdots$	0	0	0	$\cdots$	0	$2m^2$	$\cdots$	$2m^2$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
$n+2m$	0	$\cdots$	0	0	$\cdots$	0	0	0	$\cdots$	0	$2m^2$	$\cdots$	$2m^2$

▶ **Theorem 5.** *The minimum* $(\sum, \max)$*-value matroid partitioning problem is strongly NP-hard even if the matroids and weights are identical.*

Proof. Let $G = (V, F)$ be an instance of the densest l-subgraph problem. We denote $V = \{1, \dots, n\}$, $F = \{f_1, \dots, f_m\}$, and $f_i = \{u_i, v_i\}$. For any vertex set $T \subseteq V$, we denote $F[T] = \{\{u, v\} \in F : \{u, v\} \subseteq T\}$.

To solve the densest l-subgraph problem, it suffices to find a set of $n - l$ vertices such that the set of the other l vertices attain $\max_{T \subseteq V} |F[T]|$. We construct a matroid so that every feasible partition of the ground set corresponds to some set of $n - l$ vertices in V, and the $(\sum, \max)$-value is the number of edges in the induced subgraph by the other l vertices.

Let $V' = \{n+1, \dots, n+2m\}$ be a set of dummy vertices. For each $i \in V \cup V'$, we define a set E_i of $n + 2m - 1$ elements as $E_i = \{e_{ij} : j \in \{1, \dots, n+2m-1\}\}$. Let

$$E = \bigcup_{i=1}^{n+2m} E_i \quad \text{and} \quad \mathcal{I} = \{I \subseteq E : |I| \le n + 2m - 1,\ |I \cap E_i| \le 1\ (\forall i \in [n+2m])\}.$$

The resulting matroid is denoted by $(E, \mathcal{I})$, which is a $(n + 2m - 1)$-truncation of a partition matroid. We set $k = n + 2m$. The weights of elements are defined as follows:

- for each $j = 1, \dots, l-1$, set $w(e_{ij}) = 0 \quad (\forall i \in [n+2m])$;
- for each $j = l+2m, \dots, n+2m-1$, set $w(e_{ij}) = m$ if $i \le n$, and $w(e_{ij}) = 2m^2$ if $i \ge n+1$;
- set $w(e_{ij})$ $(j = l, l+1, \dots, l+2m-1)$ as follows: for each $f_t = \{u_t, v_t\}$ $(t = 1, \dots, m)$,

$$w(e_{i,l+2t-2}) = \begin{cases} t-1 & (i \in [n]), \\ 0 & (i \ge n+1), \end{cases} \quad \text{and}$$

$$w(e_{i,l+2t-1}) = \begin{cases} t & (i \in \{u_t, v_t\}), \\ t-1 & (i \in [n] \setminus \{u_t, v_t\}), \\ 0 & (i \ge n+1). \end{cases}$$

The weight is illustrated in Table 2.

We remark that $|E| = (n + 2m)(n + 2m - 1)$. By the definition of the matroid, for every $i \in [n + 2m]$, all elements in E_i belong to different independent sets from each other. Thus, for any feasible partition of E, each independent set has $n + 2m - 1$ elements which consist of one element from each E_i except one set.

It remains to show that the resulting instance is equivalent to the densest l-subgraph problem instance $(G = (V, F), l)$.

▶ **Claim 6.** *Let $\alpha \in \{0, \dots, m\}$. The graph G has a vertex set T^* with $|T^*| = l$ and $|F[T^*]| \geq \alpha$ if and only if there exists a feasible partition $(I_1, \dots, I_k)$ of E with $(\sum, \max)$-value at most $2m^2(n - l) + m^2 + m - \alpha$.*

First, we assume that there exists $T^* \subseteq V$ such that $|T^*| = l$ and $|F[T^*]| \geq \alpha$. Without loss of generality, we assume that $T^* = \{1, \dots, l\}$ and $V \setminus T^* = \{l + 1, \dots, n\}$. We show that there exists a partition such that its $(\sum, \max)$-value is at most $2m^2(n - l) + m^2 + m - \alpha$. We denote $E^j[p, q] = \{e_{p,j}, \dots, e_{q,j}\}$. Let $J_1 = \{1, \dots, l\}$, $J_2 = \{l + 1, \dots, l + 2m\}$, and $J_3 = \{l + 2m + 1, \dots, n + 2m\}$. We construct a partition $(I_1^*, \dots, I_{n+2m}^*)$ of E as follows:

$$I_j^* = \begin{cases} E^{j-1}[1, j-1] \cup E^j[j+1, n+2m] & (j \in J_1), \\ E^{j-1}[1, l] \cup E^j[l+1, n+2m+l-j] \cup E^{j-1}[n+2m+l-j+2, n+2m] & (j \in J_2), \\ E^{j-1}[1, j-2m-1] \cup E^j[j-2m+1, n] \cup E^{j-1}[n+1, n+2m] & (j \in J_3). \end{cases}$$

Then, the maximum weight of each independent set is

$$\max_{e \in I_j^*} w(e) = \begin{cases} 0 & (j \in J_1), \\ t-1 & (j = l + 2t - 1 \in J_2,\ t = 1, \dots, m,\ \{u_t, v_t\} \in F[T^*]), \\ t & \left(\begin{array}{l} j = l + 2t - 1 \in J_2,\ t = 1, \dots, m,\ \{u_t, v_t\} \notin F[T^*] \\ j = l + 2t \in J_2,\ t = 1, \dots, m \end{array} \right), \\ 2m^2 & (j \in J_3). \end{cases}$$

Thus, the $(\sum, \max)$-value is at most $0 \cdot l + \sum_{t=1}^m (2t) - |F[T^*]| + 2m^2 \cdot (n - l) \leq 2m^2(n - l) + m^2 + m - \alpha$.

Conversely, we assume that there exists a feasible partition $(I_1, \dots, I_k)$ of E such that $\max_{e \in I_1} w(e) \leq \cdots \leq \max_{e \in I_k} w(e)$, and $\sum_{j \in [k]} \max_{e \in I_j} w(e) \leq 2m^2(n - l) + m^2 + m - \alpha$. All elements in E_k must be contained in different I_j's from each other by definition of $(E, \mathcal{I})$. Hence at least $n - l$ sets contain elements e with $w(e) = 2m^2$. If $\max_{e \in I_j} w(e) \geq 2m^2$ holds for some $j \leq l + 2m$, then the objective value is at least $2m^2(n - l + 1) > 2m^2(n - l) + m^2 + m - \alpha$. Thus, each of $I_{l+2m+1}, \dots, I_k$ contains $2m$ elements with weight $2m^2$, and none of $I_1, \dots, I_{l+2m}$ contains such elements. Let $U = \{i : |E_i \cap I_j| = 0\ (\exists j \in \{l + 2m + 1, \dots, k\})\}$. Note that $|U| = n - l$ and $U \subseteq \{1, \dots, n\}$. Here, we have $2m^2(n - l) + m^2 + m - \alpha \geq \sum_{j \in [k]} \max_{e \in I_j} w(e) = 2m^2(n - l) + \sum_{j \in [l+2m]} \max_{e \in I_j} w(e)$. In order to obtain a lower bound of $\sum_{j \in [l+2m]} \max_{e \in I_j} w(e)$, we define $E' = \{e_{ij} : i \in U,\ j = 1, \dots, l + 2m\}$. Let $(E', \mathcal{I}')$ be a partition matroid where $\mathcal{I}' = \{I' : |I' \cap E_i| \leq 1\ (\forall i \in U)\}$. We observe that $\sum_{j \in [l+2m]} \max_{e \in I_j} w(e) \geq \sum_{j \in [l+2m]} \max_{e \in I_j \cap E'} w(e)$, and $(I_1 \cap E', \dots, I_{l+2m} \cap E')$ is a feasible partition to the $(\sum, \max)$ problem instance $(E', (\mathcal{I}', w), l + 2m)$. By Lemma 4, an optimal solution to $(E', (\mathcal{I}', w), l + 2m)$ can be obtained by a greedy algorithm. Let $(I'_1, \dots, I'_{l+2m})$ be an output solution of the greedy algorithm. Then we have

$$\begin{aligned} \sum_{j \in [l+2m]} \max_{e \in I_j} w(e) \geq \sum_{j \in [l+2m]} \max_{e \in I'_j} w(e) &= m + \sum_{l=1}^{m} 2(l-1) + |\{\{u, v\} : |\{u, v\} \cap U| \geq 1\}| \\ &\geq m^2 + m - |F[V \setminus U]|. \end{aligned}$$

This implies $|F[V \setminus U]| \geq \alpha$. Therefore, $T = V \setminus U$ is a vertex set with $|T| = l$ and $|F[T]| \geq \alpha$. This proves the theorem. ◀

Note that the matroid $(E, \mathcal{I})$ in the above proof is graphic because it can be seen as a matroid corresponding to a cycle with $n+2m$ vertices and each adjacent vertices is connected by $n+2m-1$ multiple edges. Thus, the maximum total capacity spanning tree partition problem is NP-hard.

3.2 PTAS for the identical case

In this subsection, we provide a PTAS for the minimum $(\sum, \max)$-value matroid partitioning problem with identical matroids and weights. This is the best possible result (unless P=NP) because the problem is strongly NP-hard as we proved in the previous subsection.

We start with the following observation, which will be also useful in Section 3.4.

▶ **Proposition 7.** *Let $(E, (\mathcal{I}_i, w_i)_{i\in[k]})$ be any instance of the minimum $(\sum, \max)$-value matroid partitioning problem, and let $(I_1^*, \dots, I_k^*)$ be an optimal solution. When we know $\max_{e\in I_i^*} w_i(e)$ for all $i \in [k]$, we can easily compute a feasible partition $(I_1, \dots, I_k)$ such that $\sum_{i\in[k]} \max_{e\in I_i} w_i(e) \leq \sum_{i\in[k]} \max_{e\in I_i^*} w_i(e)$.*

Proof. The feasible partitions for matroids $(E, \mathcal{I}_i | \{e : w_i(e) \leq \max_{e^*\in I_i^*} w_i(e^*)\})_{i\in[k]}$ satisfy the condition. Thus, we can find one of them in polynomial time by Theorem 1. ◀

Let $(E, (\mathcal{I}, w), k)$ be a problem instance, and let $\varepsilon < 1/2$ be a positive number. We write $w^{\max} = \max_{e\in E} w(e)$. Let $(I_1^*, \dots, I_k^*)$ be an optimal solution.

The idea of the algorithm is to guess the maximum weights. Since the number of possibilities of the maximum weights is at most n^k, we can solve the problem by solving the feasibility of matroid partitioning problems n^k times. Thus, we can solve the problem efficiently when k is small, but not in polynomial time. In order to reduce the possibilities, we guess $\max_{e\in I_i^*} w(e)$ only for some i's. Without loss of generality, we assume that $\max_{e\in I_1^*} w(e) \geq \cdots \geq \max_{e\in I_k^*} w(e)$. We define a set $J = \{i_1, \dots, i_s\}$ of indices by

$$i_j = \begin{cases} j & (j = 1, \dots, \lfloor 1/\varepsilon^2 \rfloor), \\ \lfloor (1+\varepsilon)^t/\varepsilon^2 \rfloor & (j = \lfloor 1/\varepsilon^2 \rfloor + t,\ t = 1, \dots, \lfloor \log_{1+\varepsilon}(k\varepsilon^2) \rfloor). \end{cases}$$

By definition, it holds that $1 = i_1 < i_2 < \cdots < i_s \leq k$, and $s = \lfloor 1/\varepsilon^2 \rfloor + \lfloor \log_{1+\varepsilon}(k\varepsilon^2) \rfloor$. Note that for any $j = \lfloor 1/\varepsilon^2 \rfloor + t$ and $t \geq 1$, we have

$$i_j - i_{j-1} \geq ((1+\varepsilon)^t/\varepsilon^2 - 1) - ((1+\varepsilon)^{t-1}/\varepsilon^2) = (1+\varepsilon)^{t-1}/\varepsilon - 1 \geq 1/\varepsilon - 1 > 1$$

as $\varepsilon < 1/2$. For notational convenience, we denote $i_0 = 0$ and $i_{s+1} = k+1$.

To reduce the number of possibilities more, we round the weights $w(e)$. For all $e \in E$, define

$$w'(e) = \begin{cases} \frac{(1+\varepsilon)^t w^{\max}}{k}\varepsilon & \left(\frac{(1+\varepsilon)^t w^{\max}}{k}\varepsilon \leq w(e) < \frac{(1+\varepsilon)^{t+1} w^{\max}}{k}\varepsilon,\ t \in \{0, 1, \dots, \lfloor \log_{1+\varepsilon}(\frac{k}{\varepsilon}) \rfloor\}\right), \\ 0 & \left(w(e) < \frac{w^{\max}}{k}\varepsilon\right). \end{cases}$$

Our algorithm guesses $\max_{e\in I_{i_j}^*} w'(e)$ for each $i_j \in J$. We write u_j^* for the value. Then, it finds a feasible partition $(I_1, \dots, I_k)$ that satisfies $\max_{e\in I_1} w(e) \geq \cdots \geq \max_{e\in I_k} w(e)$ and $\max_{e\in I_{i_j}} w'(e) \leq u_j^*$ for all $i_j \in J$. The algorithm is summarized in Algorithm 1.

▶ **Theorem 8.** *Algorithm 1 is a PTAS algorithm for the minimum $(\sum, \max)$-value matroid partitioning problem with identical matroids and weights.*

```
Algorithm 1: PTAS for the $(\sum, \max)$ problem with identical matroids and weights
1 foreach $u_1, \dots, u_s \in \{0\} \cup \left\{ \frac{(1+\varepsilon)^t w^{\max}}{k}\varepsilon \,:\, t = 0, \dots, \lfloor \log_{1+\varepsilon}(k/\varepsilon) \rfloor \right\}$ such that
    $u_1 \geq \cdots \geq u_s$ do
2     find a partition $(I_1, \dots, I_k)$ such that $I_i \in (\mathcal{I} | \{e : w'(e) \leq u_j\})$ for each
        $i_j \leq i < i_{j+1},\ j = 1, \dots, s$ if such a partition exists;
3 return the best solution $(I_1, \dots, I_k)$ among the obtained partitions;
```

Proof. Let $(I_1^*, \dots, I_k^*)$ be an optimal solution to the problem and $(I_1, \dots, I_k)$ be the output of Algorithm 1. Without loss of generality, we assume that $\max_{e \in I_1^*} w(e) \geq \cdots \geq \max_{e \in I_k^*} w(e)$. Let $u_j^* = \max_{e \in I_{i_j}^*} w'(e)$ for each $i_j \in J$.

We first analyze the running time of Algorithm 1.

▶ **Claim 9.** *Algorithm 1 runs in polynomial time with respect to k for fixed ε.*

Proof of Claim 9. Let $r = \lfloor \log_{1+\varepsilon}(k/\varepsilon) \rfloor + 2$. We observe that any choice of a possible combination of values $u_1, \dots, u_s$ corresponds a multisubset of size s from the set of r values. Thus the number of possible combinations is $\binom{r+s-1}{s}$. Furthermore, we have

$$\binom{r+s-1}{s} \leq \sum_{l=0}^{r+s-1} \binom{r+s-1}{l} = 2^{r+s-1} \leq 2^{(\log_{1+\varepsilon}(k/\varepsilon)+2)+(1/\varepsilon^2+\log_{1+\varepsilon}(k\varepsilon^2))}$$
$$\leq 2^{2\log_{1+\varepsilon} k + 2 + 1/\varepsilon^2} = 2^{2+1/\varepsilon^2} \cdot k^{\log_{1+\varepsilon} 4}.$$

This is a polynomial with respect to k for fixed ε. Thus, the algorithm runs in polynomial time. ◀

Note that, without the restriction $u_1 \geq \cdots \geq u_s$, the number of possible combinations of values $u_1, \dots, u_s$ is $r^s = k^{\Theta(\log \log k)}$, which is not polynomial with respect to k.

In the remainder, we show the approximation ratio of the algorithm.

▶ **Claim 10.** *Let* OPT *denote the optimal value and let* ALG *denote the* $(\sum, \max)$*-value of* $(I_1, \dots, I_k)$*. Then it holds that* $\mathrm{ALG} \leq (1 + 15.5\varepsilon)\mathrm{OPT}$*.*

Proof of Claim 10. First, OPT is at least

$$\mathrm{OPT} = \sum_{i \in [k]} \max_{e \in I_i^*} w(e) \geq \sum_{i \in [k]} \max_{e \in I_i^*} w'(e) \geq \sum_{j=1}^{s} (i_j - i_{j-1}) u_j^* \ .$$

Let $(I_1', \dots, I_k')$ be a feasible partition of E obtained at line 2 in Algorithm 1 using $u_1^*, \dots, u_s^*$. Then ALG is at most

$$\begin{aligned}
\mathrm{ALG} &= \sum_{i \in [k]} \max_{e \in I_i} w(e) \leq \sum_{i \in [k]} \max_{e \in I_i'} w(e) \\
&\leq \sum_{j=1}^{s} (i_{j+1} - i_j) \max_{e \in I_{i_j}'} w(e) \leq \sum_{j=1}^{s} (i_{j+1} - i_j) \left((1+\varepsilon) u_j^* + \frac{w^{\max}}{k} \varepsilon \right) \\
&\leq \sum_{j=1}^{s} (i_{j+1} - i_j)(1+\varepsilon) u_j^* + k \cdot \frac{w^{\max}}{k} \varepsilon \leq (1+\varepsilon) \sum_{j=1}^{s} (i_{j+1} - i_j) u_j^* + \varepsilon \cdot \mathrm{OPT}. \qquad (1)
\end{aligned}$$

Here, the third inequality holds by the definition of w' and $\max_{e \in I_{i_j}'} w'(e) \leq u_j^*$.

We derive an upper bound on $\sum_{j=1}^{s}(i_{j+1}-i_j)u_j^*$. To simplify notation, let $q=\lfloor 1/\varepsilon^2\rfloor$. First, since $i_{j+1}-i_j=i_j-i_{j-1}=1$ holds for any $j=1,\dots,q-1$, we have

$$\textstyle\sum_{j=1}^{q-1}(i_{j+1}-i_j)u_j^*=\sum_{j=1}^{q-1}(i_j-i_{j-1})u_j^*. \tag{2}$$

Second, we evaluate $(i_{q+1}-i_q)u_q^*$. Note that $i_q=q=\lfloor 1/\varepsilon^2\rfloor$ and $i_{q+1}=\lfloor(1+\varepsilon)/\varepsilon^2\rfloor$. Thus $i_{q+1}-i_q\le(1+\varepsilon)/\varepsilon^2-(1/\varepsilon^2-1)=(1+\varepsilon)/\varepsilon$. Moreover, $u_q^*=\max_{e\in I_q^*}w'(e)\le\max_{e\in I_q^*}w(e)\le\mathrm{OPT}/q$, because $\mathrm{OPT}=\sum_{i\in[k]}\max_{e\in I_i^*}w(e)\ge\sum_{i\in[q]}\max_{e\in I_i^*}w(e)\ge q\cdot\max_{e\in I_q^*}w(e)$. We remark that $1/q=1/\lfloor 1/\varepsilon^2\rfloor\le 1/(1/\varepsilon^2-1)=\varepsilon^2/(1-\varepsilon^2)<\frac{4}{3}\varepsilon^2<2\varepsilon^2$ as $\varepsilon<1/2$. Therefore, it follows that

$$(i_{q+1}-i_q)u_q^*\le 2\varepsilon(1+\varepsilon)\mathrm{OPT}. \tag{3}$$

Lastly, let $j\in\{q+1,\dots,s\}$, and let t (≥ 1) be the integer such that $i_j=\lfloor(1+\varepsilon)^t/\varepsilon^2\rfloor$ (i.e., $t=j-q$). We observe that $i_j-i_{j-1}\ge(1+\varepsilon)^{t-1}/\varepsilon-1$. In addition, we have

$$\begin{aligned}i_{j+1}-i_j&\le\left(\frac{(1+\varepsilon)^{t+1}}{\varepsilon^2}\right)-\left(\frac{(1+\varepsilon)^t}{\varepsilon^2}-1\right)=\frac{(1+\varepsilon)^t}{\varepsilon}+1\\&\le\frac{(1+\varepsilon)/\varepsilon+1}{(1+\varepsilon)^0/\varepsilon-1}\left(\frac{(1+\varepsilon)^{t-1}}{\varepsilon}-1\right)\le\frac{1+2\varepsilon}{1-\varepsilon}(i_j-i_{j-1})<(1+6\varepsilon)(i_j-i_{j-1}),\end{aligned}$$

where the second inequality holds since $\frac{(1+\varepsilon)^x/\varepsilon+1}{(1+\varepsilon)^{x-1}/\varepsilon-1}$ is monotone decreasing for $x\ge 1$ and the last inequality holds since $\varepsilon<1/2$. Therefore, it follows that

$$\textstyle\sum_{j=q+1}^{s}(i_{j+1}-i_j)u_j^*=\sum_{j=q+1}^{s}(1+6\varepsilon)(i_j-i_{j-1})u_j^*. \tag{4}$$

By combining (1), (2), (3), (4), together with $\varepsilon<1/2$, we have

$$\begin{aligned}\mathrm{ALG}&\le(1+\varepsilon)\left((1+6\varepsilon)\textstyle\sum_{j=1}^{s}(i_j-i_{j-1})u_j^*+2\varepsilon(1+\varepsilon)\mathrm{OPT}\right)+\varepsilon\cdot\mathrm{OPT}\\&\le(1+\varepsilon)\left((1+6\varepsilon)+2\varepsilon(1+\varepsilon)\right)\cdot\mathrm{OPT}+\varepsilon\cdot\mathrm{OPT}=(1+10\varepsilon+10\varepsilon^2+2\varepsilon^3)\mathrm{OPT}\\&<(1+10\varepsilon+5\varepsilon+0.5\varepsilon)\mathrm{OPT}=(1+15.5\varepsilon)\mathrm{OPT}.\end{aligned}$$ ◀

3.3 Hardness of the general case

We show a stronger result than the NP-hardness of the minimum $(\sum,\max)$-value matroid partitioning problem by reducing the *set cover* problem. Given a set $V=[n]$ and a collection $\mathcal{S}=\{S_i\subseteq V : i\in[k]\}$, the set cover problem is to find a subset $\mathcal{S}'$ $(\subseteq\mathcal{S})$ of minimum cardinality such that $\mathcal{S}'$ covers V, i.e., $\bigcup_{S\in\mathcal{S}'}S=V$. It is known that the set cover problem cannot be approximated in polynomial time to within a factor of $o(\log k)$ unless P=NP [6, 22].

▶ **Theorem 11.** *Even if either matroids or weights, but not both, are identical, the minimum* $(\sum,\max)$*-value matroid partitioning problem cannot be approximated in polynomial time within a factor of* $o(\log k)$*, unless P=NP.*

3.4 Algorithm for the general case

In this subsection, we provide an εk-approximation algorithm for any $\varepsilon>0$. Let $(E,(\mathcal{I}_i,w_i)_{i\in[k]})$ be an instance of the minimum $(\sum,\max)$-value matroid partitioning problem, and let $(I_1^*,\dots,I_k^*)$ be any optimal partition.

Similarly to the PTAS described in Section 3.2, our algorithm guesses $\max_{e\in I_i^*}w_i(e)$ for each $i\in[k]$. In order to reduce the number of possibilities, we only guess top-$\lceil 1/\varepsilon\rceil$

weights of $\max_{e \in I_i^*} w_i(e)$. For simplicity, let $r = \lceil 1/\varepsilon \rceil$. Let $J^* = \{i_1, \dots, i_r\}$ be the indices of top-r weights, i.e., $\max_{e \in I_i^*} w_i(e) \geq \max_{e \in I_j^*} w_i(e)$ for any $i \in J^*$ and $j \in [k] \setminus J^*$. Let $u_i^* = \max_{e \in I_i^*} w_i(e)$ for each $i \in J^*$. Then it finds a feasible partition $(I_1, \dots, I_k)$ that satisfies $\max_{e \in I_i} w_i(e) \leq u_i^*$ for $i \in J^*$ and $\max_{e \in I_i} w_i(e) \leq \min_{j \in J^*} u_j^*$ for $i \in [k] \setminus J^*$.

▶ **Theorem 12.** *For any positive fixed number $\varepsilon > 0$, there exists a polynomial-time εk-approximation algorithm for the minimum $(\sum, \max)$-value matroid partitioning problem.*

4 Complexity of other optimal matroid partitioning problems

In this section, we prove the other results in Table 1. We first deal with the cases (1) $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)}) = (\min, \min)$, $(\max, \max)$, $(\min, \max)$ or $(\min, \sum)$; (2) $(\mathrm{Op}^{(1)}, \mathrm{Op}^{(2)}) = (\max, \min)$ or $(\sum, \min)$ with identical matroids. For the $(\min, \min)$, $(\max, \max)$, $(\min, \max)$ and $(\min, \sum)$ problems, we show polynomial-time reductions to the matroid partitioning problem. Then we can see that these are polynomially solvable by Theorem 1.

▶ **Theorem 13.** *The minimum $(\min, \min)$-value matroid partitioning problem is solvable in polynomial time.*

▶ **Theorem 14.** *The minimum $(\max, \max)$ and $(\min, \max)$-value matroid partitioning problems $(E, (\mathcal{I}_i, w_i)_{i \in [k]})$ are solvable in polynomial time.*

▶ **Theorem 15.** *The minimum $(\min, \sum)$-value matroid partitioning problem $(E, (\mathcal{I}_i, w_i)_{i \in [k]})$ is solvable in polynomial time.*

Next we consider the $(\max, \min)$ case and the $(\sum, \min)$ case. As we will see later, the optimal matroid partitioning problems for these cases are (strongly) NP-hard even to approximate. We provide polynomial-time algorithms for instances where matroids are identical (weights may differ). The following lemma plays the crucial role for this purpose.

▶ **Lemma 16.** *Let $(E, \mathcal{I})$ be a matroid. If there is a partition $(I_1, \dots, I_k)$ of E such that $I_i \in \mathcal{I}$ for all $i \in [k]$, then for any k elements $e_1, \dots, e_k \in E$, there is a partition $(I'_1, \dots, I'_k)$ of E such that $e_i \in I'_i \in \mathcal{I}$ for all $i \in [k]$,*

We will reduce the problem of finding an optimal partition to the minimum weight perfect bipartite matching problem. It is well-known that this problem is solvable in polynomial time (see, e.g., [16, 23] for basic algorithms). Now we are ready to prove the theorem.

▶ **Theorem 17.** *The minimum $(\max, \min)$ and $(\sum, \min)$-value matroid partitioning problems with identical matroids $(E, (\mathcal{I}, w_i)_{i \in [k]})$ are solvable in polynomial time.*

Proof. Let $(E, \mathcal{I})$ be any matroid. Recall that the existence of a feasible partition is checkable in polynomial time by Theorem 1. Hence, in what follows, we assume that $(E, (\mathcal{I}, w), k)$ has a feasible partition.

We first consider the $(\max, \min)$ problem. By Lemma 16, the minimum $(\max, \min)$-value is at most w if and only if the bipartite graph $(E, [k], \{(e, i) : w_i(e) \leq w\})$ has a right-perfect matching. Thus, we can get the optimal value in polynomial time by setting w for all $\{w_i(e) : i \in [k],\ e \in E\}$ and checking the existence of a right-perfect matching.

Next, we consider the $(\sum, \min)$ problem. By Lemma 16, the minimum $(\sum, \min)$-value is the minimum weight of right-perfect matchings in the weighted bipartite graph $(E, [k], E \times [k]; w)$, where weight w is defined as $w(e, i) = w_i(e)$ for each $(e, i) \in E \times [k]$. Thus, we can find the optimal value in polynomial time. ◀

In addition, for the $(\max, \min)$ case and the $(\sum, \min)$ case, we prove the following hardness result by a reduction from *SAT*, which is an NP-complete problem [12].

▶ **Theorem 18.** *The minimum* $(\max, \min)$ *and* $(\sum, \min)$*-value matroid partitioning problems are both strongly NP-hard. Moreover, there exists no approximation algorithm for the problems unless P=NP.*

References

1 L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem. *Mathematical Methods of Operations Research*, 47:59–82, 1998.

2 R. E. Burkard and E. Yao. Constrained partitioning problems. *Discrete Applied Mathematics*, 28:21–34, 1990.

3 S.-P. Chen, Y. He, and G. Lin. 3-partitioning for maximizing the minimum load. *Journal of Combinatorial Optimization*, 6:67–80, 2002.

4 M. Dell'Amico and S. Martello. Bounds for the cardinality constrained $p||c_{\max}$ problem. *Journal of Scheduling*, 4:123–138, 2001.

5 P. Dell'Olmo, P. Hansen, S. Pallottino, and G. Storchi. On uniform k-partition problems. *Discrete Applied Mathematics*, 150:121–139, 2005.

6 I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, pages 624–633. ACM, 2014.

7 J. Edmonds. Minimum partition of a matroid into independent subsets. *JOURNAL OF RESEARCH of the National Bureau of Standards—B. Mathematicsnd Mathematical Physics*, 69B:67–72, 1965.

8 J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and their Applications*, pages 69–87. Gordon and Breach, New York, 1970.

9 J. Edmonds and D. R. Fulkerson. Transversals and matroid partition. *Journal of Research of the National Bureau of Standards*, 69B:147–153, 1965.

10 U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

11 A. Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, 1981.

12 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman New York, 1979.

13 Y. He, Z. Tan, J. Zhu, and E. Yao. k-partitioning problems for maximizing the minimum load. *Computers and Mathematics with Applications*, 46:1671–1681, 2003.

14 Y. Kawase, K. Kimura, K. Makino, and H. Sumita. Optimal Matroid Partitioning Problems. *arXiv preprints cs.DS/1710.00950*, October 2017. `arXiv:1710.00950`.

15 H. Kellerer and G. Woeginger. A tight bound for 3-partitioning. *Discrete Applied Mathematics*, 45:249–259, 1993.

16 B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer, 2002.

17 E. L. Lawler. Matroids with parity conditions: A new class of combinatorial optimization problems. Memo erl-m334, Electronics Research Laboratory, College of Engineering, UC Berkeley, Berkeley, CA, 1971.

18 J. Lee, M. Sviridenko, and J. Vondrák. Matroid matching: The power of local search. *SIAM Journal on Computing*, 42(1):357–379, 2013.

19 J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

20 W. Li and J. Li. Approximation algorithms for k-partitioning problems with partition matroid constraint. *Optimization Letters*, 8:1093–1099, 2014.

21 L. Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Series B*, 28(2):208–236, 1980.

22 D. Moshkovitz. The projection games conjecture and the NP-hardness of ln n-approximating set-cover. *Theory of Computing*, 11(7):221–235, 2015.

23 A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

24 J. Verschae and A. Wiese. On the configuration-LP for scheduling on unrelated machines. *Journal of Scheduling*, 17:371–383, 2014.

25 B. Wu and E. Yao. k-partitioning problems with partition matroid constraint. *Theoretical Computer Science*, 374:41–48, 2007.

26 B. Wu and E. Yao. Lower bounds and modified LPT algorithm for k-partitioning problems with partition matroid constraint. *Applied Mathematics-A Journal of Chinese Universities*, 23:1–8, 2008.

Improved Bounds for Online Dominating Sets of Trees[*][†]

Koji M. Kobayashi

National Institute of Informatics, Tokyo, Japan
kobaya@nii.ac.jp

Abstract

The online dominating set problem is an online variant of the minimum dominating set problem, which is one of the most important NP-hard problems on graphs. This problem is defined as follows: Given an undirected graph $G = (V, E)$, in which V is a set of vertices and E is a set of edges. We say that a set $D \subseteq V$ of vertices is a *dominating set* of G if for each $v \in V \setminus D$, there exists a vertex $u \in D$ such that $\{u, v\} \in E$. The vertices are revealed to an online algorithm one by one over time. When a vertex is revealed, edges between the vertex and vertices revealed in the past are also revealed. A revealed subtree is connected at any time. Immediately after the revelation of each vertex, an online algorithm can irrevocably choose vertices which were already revealed and must maintain a dominating set of a graph revealed so far. The cost of an algorithm on a given tree is the number of vertices chosen by it, and its objective is to minimize the cost. Eidenbenz (Technical report, Institute of Theoretical Computer Science, ETH Zürich, 2002) and Boyar et al. (SWAT 2016) studied the case in which given graphs are trees. They designed a deterministic online algorithm whose competitive ratio is at most three, and proved that a lower bound on the competitive ratio of any deterministic algorithm is two.

In this paper, we also focus on trees. We establish a matching lower bound for any deterministic algorithm. Moreover, we design a randomized online algorithm whose competitive ratio is exactly $5/2 = 2.5$, and show that the competitive ratio of any randomized algorithm is at least $4/3 \approx 1.333$.

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 algorithm, dominating set, competitive analysis, tree graph, randomized algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.52

1 Introduction

The dominating set problem is one of the most important NP-hard problems on graphs. This problem is defined as follows: Given an undirected graph $G = (V, E)$, in which V is a set of vertices and E is a set of edges. We say that a set $D \subseteq V$ of vertices is a *dominating set* of G if for each vertex $v \in V \setminus D$, there exists a vertex $u \in D$ such that $\{u, v\} \in E$. The objective of the problem is to construct a minimum dominating set. This problem has been extensively studied for many applications, such as communication in ad-hoc networks (see e.g., [16]) and facility location on networks (e.g., [12]).

The dominating set problem has also been studied in online settings [11, 4, 2]. In one of the settings [4, 2], vertices are revealed to an online algorithm one by one, and edges

[*] This work was supported by JSPS KAKENHI Grant Number 26730008.
[†] A full version of the paper is available at https://arxiv.org/abs/1710.11414.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 52; pp. 52:1–52:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

between a revealed vertex and vertices revealed in the past are also revealed. The *input* of this setting is an undirected graph and a sequence consisting of all the vertices of the graph. (This sequence represents an order of the vertices revealed to an online algorithm.) An online algorithm holds the empty set U at the beginning. When a new vertex is revealed, the algorithm can add vertices revealed so far to U, which means that an added vertex is not necessarily the newly revealed one. The algorithm must not remove a vertex from U. The total number of vertices is not known to an online algorithm before the final vertex is revealed. Thus, U must be a dominating set immediately after the revelation of each vertex. The performance of online algorithms is evaluated using *competitive analysis* [1, 14]. The *cost* of an algorithm ALG for an input σ is the size of a dominating set constructed by ALG for σ, which is denoted as $C_{ALG}(\sigma)$. We say that the (strict) competitive ratio of an online algorithm ON is at most c or ON is *c-competitive* if for any input σ, $C_{ON}(\sigma) \leq cC_{OPT}(\sigma)$, in which OPT is an optimal offline algorithm for σ. If ON uses randomization, the expected cost of ON is used.

Previous Results and Our Results

For trees, Eidenbenz [4] and Boyar et al. [2] designed a 3-competitive deterministic algorithm, and proved that the competitive ratio of any deterministic online algorithm is at least two (Boyar et al. showed their results in terms of asymptotic competitive ratios, but the results can hold for strict competitive ratios as well).

In this paper, we show the following three results for trees: (i) We prove that a lower bound on the competitive ratio of any deterministic algorithm is three. This bound matches the above upper bound. (ii) We establish a randomized online algorithm whose competitive ratio is exactly $5/2 = 2.5$. This algorithm is the first non-trivial randomized algorithm for the online dominating set problem for any graph class. (iii) We show that the competitive ratio of any randomized algorithm is at least $4/3 \approx 1.333$. The above results are shown with respect to the *strict* competitive ratio. However, it is easy to see that the same results for the *asymptotic* competitive ratios as (i) and (iii) can be shown in a quite similar way to their proofs. (Note that any upper bound on the strict competitive ratio is an upper bound on that on the asymptotic competitive ratio. That is, (ii) holds for the asymptotic competitive ratio.)

Related Results

For several graph classes, Eidenbenz [4] and Boyar et al. [2] studied online algorithms of a few variants of dominating sets, namely, connected dominating sets, total dominating sets and independent dominating sets. Their results are summarized in the table in Sec. 6 of [4] and Table 2 in Sec. 1 of [2]. For example, they proved that the optimal competitive ratios on a bipartite graph and a planar graph are $n - 1$, in which n is the number of given vertices. Boyar et al. [2] defined an *incremental* algorithm as an algorithm which maintains a dominating set immediately after a new vertex is revealed. An online algorithm is incremental, but an optimal incremental algorithm knows the whole input and can perform better than any online algorithm. They measured the performance of online algorithms compared with an optimal *incremental* algorithm in addition to an optimal offline algorithm. Moreover, they compared the performance of an optimal incremental algorithm with that of an optimal offline algorithm for several graph classes, which is also summarized in Table 1 in Sec. 1 of [2].

King and Tzeng [11] studied two different variants of online dominating sets on general graphs. One variant is the same as the one studied in this paper, except that immediately after a new vertex is revealed, an online algorithm can choose the new one but cannot choose vertices revealed previously. In this setting, they designed a deterministic algorithm whose competitive ratio is at most $n-1$, and proved that the algorithm is the best possible. In the other variant, an online algorithm knows all vertices in advance, and at a time i, all the edges between the i-th vertex v_i and the other vertices are revealed. They showed an upper bound of $3\sqrt{n}/2$ and a lower bound of of $\sqrt{n}$ for this variant.

For the offline setting, the minimum dominating set problem is one of the most significant NP-hard problems on graphs and has been widely studied. One of the most important open problems is to develop exact (exponential) algorithms (see, e.g. [5, 6, 8, 13, 15, 10]). The current fastest algorithm solves this problem in $O(1.4864^n)$ time and polynomial space [10]. Moreover, many variants have been proposed by putting additional constraints on the original dominating set problem and have been extensively studied: for example, connected domination, independent domination and total domination (see, .e.g. [3],[7] and [9], respectively).

2 Preliminaries

2.1 Model Description

We are given an undirected tree and its vertices are revealed to an online algorithm one by one over time. The total number of the vertices is not known to the online algorithm up to the end of the input. When the i-th vertex v_i is revealed to the online algorithm, all the edges between v_i and v_j such that $j < i$ are also revealed. Except for the first revealed vertex, a newly revealed vertex has exactly one edge to a vertex revealed previously. That is, a revealed subtree is connected at any time. An *input* of the problem is a three-tuple of the form (V, E, S), in which V is the set of all the vertices of a given tree, E is the set of all the undirected edges of the tree, and S is a sequence consisting of all the vertices in V. S represents an order of the vertices revealed to an online algorithm. An algorithm has the empty set U before the first vertex is revealed. The algorithm can add vertices into U immediately after the revelation of each vertex, and it is necessary for U to be a dominating set of the given tree at the end of the input. If the algorithm is online, it does not know when the input has ended, and thus U must be a dominating set immediately after each vertex is revealed. Once a vertex is added into U, it must not be removed from U later. The *cost* of the algorithm for an input σ is the number of vertices in U at the end of σ, and the objective of the problem is to minimize the cost. We evaluate the performance of an online algorithm using competitive analysis. We say that the *competitive ratio* of a deterministic online algorithm ON is at most c if for any input σ, $C_{ON}(\sigma) \leq cC_{OPT}(\sigma)$. If ON is a randomized online algorithm, then the expected cost of ON is used, which is denoted by $\mathbb{E}[C_{ON}(\sigma)]$. If for any input σ, $\mathbb{E}[C_{ON}(\sigma)] \leq cC_{OPT}(\sigma)$, then we say that the competitive ratio of a randomized online algorithm ON is at most c against any oblivious adversary.

If the number of vertices in a given tree is one, the cost ratio of any algorithm is clearly one. Thus, we assume that this number is at least two.

2.2 Notation and Definitions

In this section, we give some definitions and notation used throughout this paper. For any $i(=1, 2, \ldots)$, we use v_i to denote the i-th revealed vertex to an online algorithm (the first revealed vertex v_1 appears frequently in this paper). We say that vertices v and u are *adjacent* if $\{v, u\} \in E$, in which E is the set of all the edges of a given graph. When a vertex v is revealed such that v is adjacent to a vertex u which was revealed before v, then we say that v *arrives* at u. For any vertex v and any online algorithm ON, $D_{ON}(v)$ denotes a dominating set constructed by ON of a revealed graph up to the time of the revelation of v. We will omit ON from the notation when it is clear from the context. For an algorithm ALG including an offline algorithm, $D_{ALG}(\sigma)$ denotes a dominating set constructed by ALG after the end of the input σ. We will omit σ from the notation when it is clear from the context. For a vertex v, we say that ALG *selects* v if $v \in D_{ALG}$. For vertices u and v such that u is revealed after v, $deg_u(v)$ denotes the degree of v immediately after u is revealed. $deg(v)$ denotes the degree of v after the end of the input. For a vertex v and a vertex u revealed after v, we say that u is a *descendant* of v if any vertex on the simple path from v to u is revealed after v. The *cost* of a deterministic algorithm ALG for a vertex set U is the number of vertices selected by ALG in U. That is, it is the number of vertices in $U \cap D_{ALG}$. Moreover, if U contains only one vertex, then we simply say the *cost for the vertex*. In the same way, we use the term "the expected cost of ALG for U (or a vertex)" if ALG is a randomized algorithm.

3 Deterministic Lower Bound

Due to page limitations, we omit most of the proofs of the following lemmas and theorems. The full version of this paper is available at `https://arxiv.org/abs/1710.11414`.

3.1 Overview of Proof

We first outline an input to obtain our lower bound. The tree of the input is constructed according to two routines. The tree can be divided into several subtrees satisfying some properties and we evaluate the competitive ratio for each set of subtrees. One of the routines appoints a vertex as the root to construct a subtree, which is called a *base vertex*. The other routine constructs several subtrees with at most two leaves, each of which arises from the base vertex. The set of all the vertices excluding the root in each of the subtrees is called a *T-set*. It depends on the behavior of an online algorithm ON how many T-sets are constructed and how many leaves and inner vertices composing T-sets are. If a T-set contains two leaves, the leaves share the adjacent vertex. For each T-set, OPT selects one vertex for every consecutive three vertices starting with the parent of a leaf in it. If the degree of a vertex selected by OPT is two, ON selects the vertex and the two adjacent vertices. Otherwise, that is its degree is at least three, ON selects at least three vertices from the vertex and all the adjacent vertices.

Let us explain the proof more in detail. If a T-set contains sufficiently many inner vertices, it is called a *T_3-set*. Otherwise, a T-set such that ℓ modulo $3 = i$ is called a *T_i-set*, in which ℓ is the length from the base vertex to a leaf in the T-set. One of the routines tries to force ON to construct one of the following four sets of T-sets from a base vertex (Fig. 1): (1) a set of two T_1-sets and at least zero T_0-set, (2) a set of one T_2-set and at least zero T_0-set, (3) a set of one T_3-set, at most one T_1-set and at least zero T_0-set, and (4) a set of sufficiently many T_0-sets and at most one T_1-set. The cost ratios of these T-sets are three for (1) or (2) and approximately three for (3) or (4), respectively. ON can construct none of these sets. Namely, (5) ON constructs one T_1-set and then does one T_2-set (Further, ON may also construct T_0-sets). In this case, the routine partitions the T_2-set into a vertex u, a T_1-set and a T_0-set. This T_0-set and all the T-sets in (5) except for the partitioned T_2-set compose a set of T-sets of (1). Then, the routine finishes constructing a subtree from the current

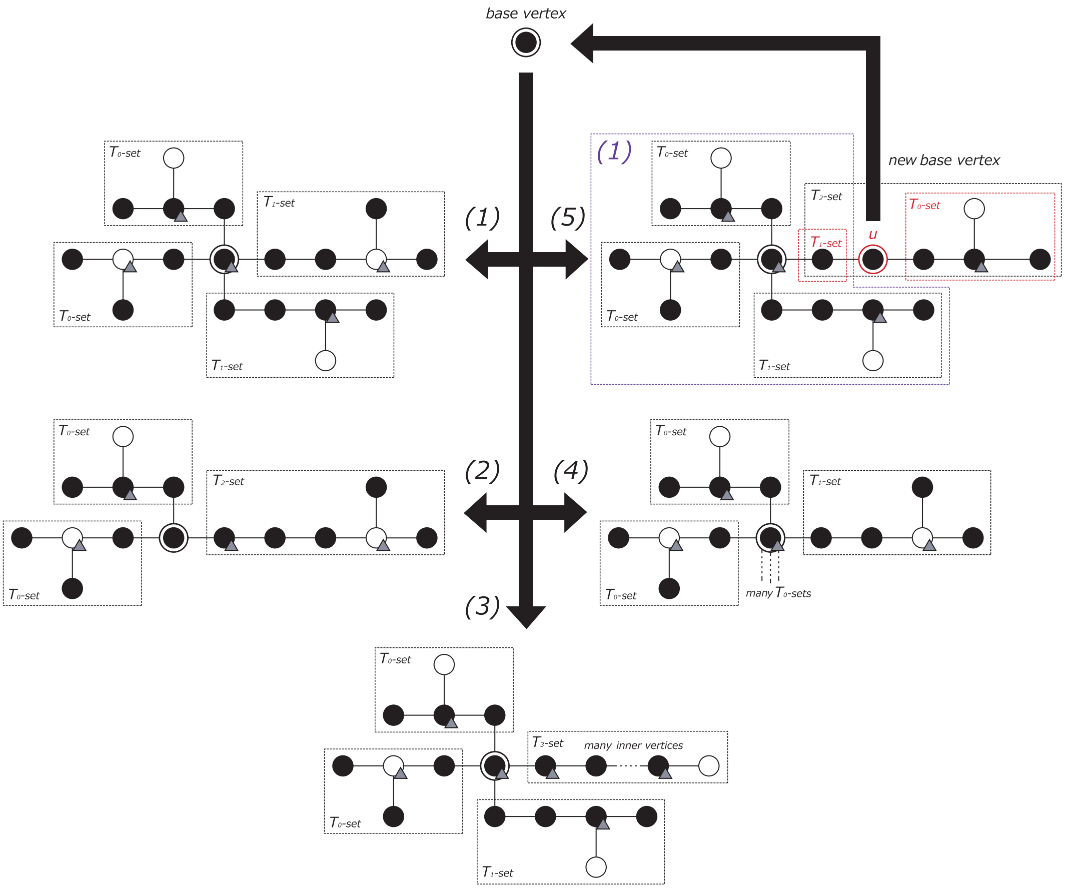


Figure 1 An example of the five sets of T-sets from (1) to (5). Highlighted vertices denote base vertices. Black vertices denote vertices selected by ON. Vertices with a gray triangle denote vertices selected by OPT. One of the five sets of T-sets is constructed for a base vertex. If (5) is constructed, the T_2-set in the set is partitioned into a new base vertex u, a T_1-set and a T_0-set. After that, the routines force ON to construct one of the five sets of T-sets for u recursively. u is not dominated by OPT yet, but is dominated later.

base vertex, whose cost ratio is three, and appoints u as a new base vertex. One T_0-set, which is constructed from the above partition of the T_2-set, belongs to the new base vertex u. Since the set of T-sets of u is not classified into any of the above four categories, the routine continues to construct subtrees for u. This is how the routine tries to construct one of the four sets of T-sets for all base vertices and to achieve a lower bound of (approximately) three. Therefore, we have the following theorem:

▶ **Theorem 1.** *For any $\varepsilon > 0$, the competitive ratio of any deterministic online algorithm is at least $3 - \varepsilon$.*

4 Randomized Upper Bound

4.1 Algorithm

First, we define our algorithm RA. Before the first vertex is revealed, RA chooses to start running one of two deterministic online algorithms A and B, which are defined later, with the probability of 1/2 and thereafter keeps running it up to the end of the input. For a vertex v, $p(v)$ denotes the length of the simple path from v_1 to v. Roughly speaking, the difference between A and B is that for a vertex v, A selects v if $p(v)$ is odd, and B selects v if $p(v)$ is even. Then, A and B try to establish the property that for any vertex u of degree at most two, $u \notin D_A \cap D_B$.

A (B) can select a vertex which A (B) selected previously in the following definition. It means that A (B) does nothing at that time. First, we give the definition of A as follows.

Algorithm 1: Algorithm A

```
Suppose that the $i$-th vertex $v_i$ is revealed.
Case 1 ($i = 1$): Select $v_1$.
Case 2 ($i \geq 2$): Suppose that $v_i$ arrives at a vertex $u$.
  Case 2.1 ($deg_{v_i}(u) \geq 3$): Select $u$.
  Case 2.2 ($deg_{v_i}(u) \leq 2$):
    Case 2.2.1 ($p(v_i)$ modulo $2 = 0$): Select $u$.
    Case 2.2.2 ($p(v_i)$ modulo $2 = 1$): Select $v_i$.
```

Since A selects either a revealed vertex v_i or the vertex adjacent to v_i, the set of vertices selected by A is a dominating set of a revealed graph immediately after each of A's selections.

The definition of B is quite the same as that of A except for Case 2.2. The process of B in Case 2.2.1 (2.2.2) is the same as that of A in Case 2.2.2 (2.2.1). That is, B selects v_i and u in Cases 2.2.1 and 2.2.2, respectively. Thus, the set of vertices selected by B is also a dominating set at any time. We omit its formal definition due to page limitation.

4.2 Basic Properties of RA

In this section, we show several basic properties of dominating sets by A and B.

► **Lemma 2.** *The following properties hold for a vertex v:*
(1) *If $v = v_1$, $v \in D_A$ and $v \in D_B$.*
Suppose that $v \neq v_1$.
(2) *If $deg(v) \geq 3$, $v \in D_A$ and $v \in D_B$.*
(3) *Suppose that $deg(v) = 2$.*
(3-e) *If $p(v)$ modulo $2 = 0$, $v \notin D_A$ and $v \in D_B$.*
(3-o) *If $p(v)$ modulo $2 = 1$, $v \in D_A$ and $v \notin D_B$.*
(4) *Suppose that $deg(v) = 1$ and let $\tilde{u}$ be the vertex adjacent to v.*
(4-1) *Suppose that $deg(\tilde{u}) \geq 3$ and $deg_v(\tilde{u}) \leq 2$.*
(4-1-e) *If $p(v)$ modulo $2 = 0$, $v \notin D_A$ and $v \in D_B$.*
(4-1-o) *If $p(v)$ modulo $2 = 1$, $v \in D_A$ and $v \notin D_B$.*
(4-2) *If $deg(\tilde{u}) \geq 3$ and $deg_v(\tilde{u}) \geq 3$, then $v \notin D_A$ and $v \notin D_B$.*
(4-3) *Suppose that $deg(\tilde{u}) \leq 2$.*
(4-3-e) *If $p(v)$ modulo $2 = 0$, $v \notin D_A$ and $v \in D_B$.*
(4-3-o) *If $p(v)$ modulo $2 = 1$, $v \in D_A$ and $v \notin D_B$.*

► **Lemma 3.** *The expected cost of RA for v is as follows:*
(1) *If $v = v_1$, it is one.*
Suppose that $v \neq v_1$.
(2) *If $deg(v) \geq 3$, it is one.*
(3) *If $deg(v) = 2$, it is $1/2$.*
(4) *Suppose that $deg(v) = 1$ and v is adjacent to a vertex u.*

(4-1) *If $deg(u) \geq 3$ and $deg_v(u) \leq 2$, it is $1/2$.*
(4-2) *If $deg(u) \geq 3$ and $deg_v(u) \geq 3$, it is zero.*
(4-3) *If $deg(u) \leq 2$, it is $1/2$.*

We say that a vertex v *dominates* vertices adjacent to v if OPT selects v. We also say that v *dominates* v itself. If a vertex u arrives at a vertex v, (v, u) denotes the edge between v and u. Suppose that a vertex u arrives at a vertex v. Also, suppose that u is dominated by a vertex in U and v is dominated by a vertex not in U, in which U is the set of u and all the descendants of u. Then, we say that the edge (v, u) is *free*. We say that a free edge (v, u) is *fixed* if this edge satisfies the following three conditions: (i) $v \neq v_1$, (ii) $deg(u) \geq 3$, and (iii) either $deg(v) = 3$ or $deg(v) = 2$, $deg(v') \geq 3$ and $deg_v(v') \geq 3$, in which $v'(\neq u)$ is the vertex at which v arrives. We say that a vertex triplet (u_1, u_2, u_3) is *good* if the vertices u_1, u_2 and u_3 satisfy the following three conditions: (i) both u_1 and u_3 are adjacent to u_2, (ii) $deg(u_1) = deg(u_2) = deg(u_3) = 3$, and (iii) OPT selects u_1 and u_3.

In the rest of this section, we will show the following lemma.

▶ **Lemma 4.** *There exists an input σ which maximizes $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)}$ and satisfies the following seven properties.*

(P1) *Any free edge is fixed (Lemmas 7 and 9).*
(P2) *The degree of any vertex is at most three (Lemma 8).*
(P3) *The degree of any vertex selected by OPT is three (Lemma 10).*
(P4) *For any free edge (v, u), OPT does not select v (Lemma 11).*
(P5) *Good vertex triplets are not contained (Lemma 12).*
(P6) *For any free edge (v, u), the degree of v is not two (Lemma 14).*
(P7) *The degree of any vertex is either one or three (Lemma 13).*

This lemma shows that we only have to consider an input satisfying the properties from (P1) to (P7) to evaluate the competitive ratio of RA. It is easy to see that if (P7) holds, both (P2) and (P6) clearly hold. However, we must prove some lemmas including ones about the both properties before showing (P7).

To prove the above lemma and the following lemmas, we give a few definitions about transformations of an input. First, we "divide" an input into two inputs. For an input $\sigma = (V, E, S)$ and a vertex $v \in V$, we define the input $f_1(\sigma, v) = (V_1, E_1, S_1)$ such that $V_1 = V \setminus U$, in which U is the set of v and all the descendants of v, $E_1 = \{\{u, u'\} \in E \mid u, u' \in V_1\}$. That is, (V_1, E_1) is the subgraph of (V, E) induced by V_1, and S_1 is the subsequence of S consisting of all the vertices of V_1 . Also, we define the input $f_2(\sigma, v) = (U, E_2, S_2)$ such that $E_2 = \{\{u, u'\} \in E \mid u, u' \in U\}$, that is, (U, E_2) is the subgraph of (V, E) induced by U, and S_2 is the subsequence of S consisting of all the vertices of U.

Moreover, we "connect" two inputs. For an input $\sigma' = (V', E', S')$, a vertex $v' \in V'$ and an input $\sigma'' = (V'', E'', S'')$, we define $f_3(\sigma', v', \sigma'') = (V_3, E_3, S_3)$ such that $V_3 = V' \cup V''$, $E_3 = E' \cup E'' \cup \{\{v', u''_1\}\}$, in which u''_i is the $i(\in [1, n''])$-th vertex in S'' and n'' is the number of vertices in S'', and $S_3 = (u'_1, \ldots, u'_{n'}, u''_1, \ldots, u''_{n''})$, in which n' is the number of vertices in S' and u'_i is the $i(\in [1, n'])$-th vertex in S'.

▶ **Lemma 5.** *Suppose that the vertex set of an input σ contains two vertices v and u such that the edge (v, u) is free. Then, there exists OPT such that $D_{OPT}(f_1(\sigma, u)) \cup D_{OPT}(f_2(\sigma, u)) = D_{OPT}(\sigma)$.*

▶ **Lemma 6.** *Suppose that the graph in an input σ contains a vertex v and OPT selects v. Then, there exists OPT such that $D_{OPT}(f_3(\sigma, v, \hat{\sigma})) = D_{OPT}(\sigma)$, in which $\hat{\sigma} = (\{u\}, \varnothing, u)$ and u is a vertex not in the vertex set of the graph in σ.*

▶ **Lemma 7.** *Suppose that the graph of an input σ contains at least one free edge which is not fixed. Then, there exits an input σ' such that (a) any free edge in the graph of σ' is fixed, that is (P1) holds, and (b) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.*

▶ **Lemma 8.** *Suppose that an input σ satisfies the following conditions: (i) the graph in σ contains at least one vertex of degree at least four, and (ii) (P1) holds.*

Then, there exists an input σ' such that (a) the degree of any vertex of the graph in σ' is at most three, that is, (P2) holds, and (b) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.

▶ **Lemma 9.** *Suppose that an input σ satisfies the following conditions: (i) the graph in σ contains at least one free edge which is not fixed, and (ii) (P2) holds.*

Then, there exists an input σ' such that (a) (P1) and (P2) hold, and (b) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.

▶ **Lemma 10.** *Suppose that an input σ satisfies the following conditions: (i) OPT selects at least one vertex of degree at most two, and (ii) (P1) and (P2) hold.*

Then, there exists an input σ' such that (a) the degree of any vertex selected by OPT is three, that is, (P3) holds, (b) (P1) and (P2) hold, and (c) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.

▶ **Lemma 11.** *Suppose that an input σ satisfies the following conditions: (i) there exists at least one free edge (v, u) such that OPT selects v, and (ii) (P1), (P2) and (P3) hold.*

Then, there exists an input σ' such that (a) for any free edge (v, u), OPT does not select v, that is, (P4) holds, (b) (P1), (P2) and (P3) hold, and (c) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.

▶ **Lemma 12.** *Suppose that an input σ satisfies the following conditions: (i) the graph in σ contains at least one good vertex triplet, and (ii) the properties from (P1) to (P4) inclusive hold.*

Then, there exists an input σ' such that (a) the graph in σ' contains no good vertex triplets, that is, (P5) holds, (b) the properties from (P1) to (P4) inclusive hold, and (c) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.

▶ **Lemma 13.** *Consider the graph in an input satisfying the properties from (P1) to (P6) inclusive. Then, the degree of any vertex is one or three.*

▶ **Lemma 14.** *Suppose that an input σ satisfies the following conditions: (i) there exists at least one free edge (v, u) such that $deg(v) = 2$, and (ii) the properties from (P1) to (P5) inclusive hold.*

Then, there exists an input σ' such that (a) for any free edge (v, u), $deg(v) = 3$, (b) the properties from (P1) to (P5) inclusive hold, and (c) $\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} \leq \frac{\mathbb{E}[C_{RA}(\sigma')]}{C_{OPT}(\sigma')}$.

Now we can show Lemma 4 using Lemmas 13 and 14. In the next section, we analyze only inputs satisfying the properties from (P1) through (P7).

4.3 Analysis of RA

We assign a positive integer to each vertex of a given tree according to the below routine. We call the set of all the vertices with the same assigned value a *block*. All the vertices in a block are on a path of at most length three. We obtain the competitive ratio of RA by evaluating the costs of RA and OPT for each block. For a vertex v, $N(v)$ denotes the set of vertices adjacent to v. That is, $N(v) = \{u \mid \{v, u\} \in E\}$, in which E is the set of all the edges of a given graph.

Algorithm 2: BlockRoutine

```
Step 1: $\ell := 0$ and $U := \{v_i \mid i = 1, \dots, n\}$, in which $n$ is the number of all the vertices of a given graph.
Step 2: $\ell := \ell + 1$. If $U = \varnothing$, then finish. Otherwise, $i_1 := \min\{i \mid v_i \in U\}$, $U := U \setminus \{v_{i_1}\}$ and assign $\ell$ to the vertex $v_{i_1}$.
Step 3: If $U \cap N(v_{i_1}) = \varnothing$, then go to Step 2. Otherwise, $i_2 := \min\{i \mid v_i \in U \cap N(v_{i_1})\}$, $U := U \setminus \{v_{i_2}\}$ and assign $\ell$ to the vertex $v_{i_2}$.
Step 4: $U' := U \cap \{N(v_{i_1}) \cup N(v_{i_2})\}$. If $U' = \varnothing$, then go to Step 2. Otherwise, $i_3 := \min\{i \mid v_i \in U'\}$, $U := U \setminus \{v_{i_3}\}$, assign $\ell$ to the vertex $v_{i_3}$ and go to Step 2.
```

▶ **Lemma 15.** *The number of all the vertices of a given tree is at least four.*

By the definition of BlockRoutine, this lemma leads to the fact that at least one vertex in a block is adjacent to a vertex in another block. Also, by (P7) in the previous section, the degree of a vertex is one or three, and hence blocks which a given graph can contain are classified into the following four categories: A $\boldsymbol{B_1}$*-block* is a set consisting of one vertex u_1 such that $deg(u_1) = 1$. The following three blocks are sets consisting of three vertices u_1, u_2 and u_3. Suppose that both u_1 and u_3 are adjacent to u_2. A $\boldsymbol{B_2}$*-block* is a set such that $deg(u_1) = 3$, $deg(u_2) = 3$ and $deg(u_3) = 1$, a $\boldsymbol{B_3}$*-block* is a set such that $deg(u_1) = 1$, $deg(u_2) = 3$ and $deg(u_3) = 1$, and a $\boldsymbol{B_4}$*-block* is a set such that $deg(u_1) = 3$, $deg(u_2) = 3$ and $deg(u_3) = 3$.

For each block, we discuss vertices selected by OPT and classify B_1, B_2, B_3 and B_4 into the following eleven categories. Then the next lemma shows that we only have to consider six categories. u_1, u_2 and u_3 to classify B_i are used in the same definitions as those of u_1, u_2 and u_3 to define B_i.

B_1-blocks are classified into two categories: A $\boldsymbol{B_1^0}$*-block* in which OPT does not select any vertex, and a $\boldsymbol{B_1^1}$*-block* in which OPT selects only u_1.

B_2-blocks are classified into two categories: A $\boldsymbol{B_2^{010}}$*-block* in which OPT selects only u_2, and a $\boldsymbol{B_2^{110}}$*-block* in which OPT selects only u_1 and u_2.

B_3-blocks are not classified. OPT selects only u_2 in a B_3-block.

B_4-blocks are classified into six categories: A $\boldsymbol{B_4^{000}}$*-block* in which OPT selects no vertices, a $\boldsymbol{B_4^{100}}$*-block* in which OPT selects only u_1, a $\boldsymbol{B_4^{010}}$*-block* in which OPT selects only u_2, a $\boldsymbol{B_4^{110}}$*-block* in which OPT selects only u_1 and u_2, a $\boldsymbol{B_4^{101}}$*-block* in which OPT selects only u_1 and u_3, and a $\boldsymbol{B_4^{111}}$*-block* in which OPT selects all the vertices.

▶ **Lemma 16.** *An input can contain at most six kinds of blocks:* $B_1^0, B_2, B_3, B_4^{000}, B_4^{100}$ *and* B_4^{010}.

A B_1-block consists of one vertex v of degree one, and (4) in Lemma 3 shows that the expected cost of RA for v depends on the adjacent vertex u. Then, we classify B_1-blocks into the following two categories in terms of RA: A $B_{1,0}$-block of v such that $deg_v(u) = 3$ and a $B_{1,1}$-block of v such that $deg_v(u) \leq 2$.

▶ **Lemma 17.** *Consider a block without* v_1 *and then the expected costs of* RA *are as follows: (i) zero for a* $B_{1,0}$*-block, (ii)* $1/2$ *for a* $B_{1,1}$*-block, (iii) at most* $5/2$ *for a* B_2*-block, (iv) at most* $3/2$ *for a* B_3*-block and (v) at most three for a* B_4*-block.*

Next, we evaluate the expected cost of RA for each block with v_1. Since the number of all the vertices of a given graph is at least four, no B_1-block contains v_1 by the definition of BlockRoutine.

▶ **Lemma 18.** *Consider a block with v_1 and then the expected costs of RA are as follows: (i) at most three for a B_2-block, (ii) at most $5/2$ for a B_3-block and (iii) at most three for a B_4-block.*

Let $b_{1,0}, b_{1,1}, b_2, b_3, b_4^{000}, b_4^{100}$ and b_4^{010} denote the numbers of $B_{1,0}$-blocks, $B_{1,1}$-blocks, B_2-blocks, B_3-blocks, B_4^{000}-blocks, B_4^{100}-blocks and B_4^{010}-blocks, respectively. We define $b_4 = b_4^{000} + b_4^{100} + b_4^{010}$.

▶ **Lemma 19.** *If the number of all the vertices of a given graph is at least five,*

$$b_{1,0} \le b_2 + b_4^{100} + b_4^{010} \tag{1}$$

and

$$b_{1,1} \le b_4^{100}. \tag{2}$$

▶ **Lemma 20.** $b_{1,0} + b_{1,1} + b_3 = b_2 + 3b_4 + 2$.

▶ **Theorem 21.** *The competitive ratio of RA is at most* $5/2$.

Proof. First, we consider an input σ of which the number of vertices of a given tree is four. A combination of blocks composing a tree with four vertices consists of one B_1-block C_1 and one B_3-block C_3 by Lemma 16. Since C_1 does not contain v_1 by the definition of BLOCKROUTINE, C_3 contains v_1. Thus, the expected of RA for C_3 is at most $5/2$ by Lemma 18. Let v denote the vertex in C_1, and let u denote the vertex adjacent to v in C_3. $deg_v(u) = 3$ by the definition of B_3-blocks, which means that C_1 is a $B_{1,0}$-block. Thus, the expected cost for C_1 is zero by Lemma 17. By the above argument, $\mathbb{E}[C_{RA}(\sigma)] = 5/2$. On the other hand, OPT clearly selects at least one vertex, that is, $C_{OPT}(\sigma) \ge 1$. Therefore, we have shown the statement of the theorem in the case of a tree with four vertices.

Next, we consider the case in which of a graph with at least five vertices. The expected costs of RA for a B_2-block and a B_3-block with v_1 are greater than those for a B_2-block and a B_3-block without v_1 by $1/2$ and one, respectively, by Lemmas 17 and 18. Also, v_1 does not affect the expected cost for B_4-blocks. Thus, let b'_2 and b'_3 denote the numbers of B_2-blocks and B_3-blocks with v_1, respectively. By definition, $b'_2, b'_3 \in \{0, 1\}$ and $b'_2 + b'_3 \le 1$. Then, using Lemma 17, we have

$$\mathbb{E}[C_{RA}(\sigma)] \le b_{1,1}/2 + 5b_2/2 + 3b_3/2 + 3b_4 + b'_2/2 + b'_3 \le b_{1,1}/2 + 5b_2/2 + 3b_3/2 + 3b_4 + 1.$$

By the definitions of blocks, we have

$$C_{OPT}(\sigma) = b_2 + b_3 + b_4^{100} + b_4^{010}.$$

By the inequality and the equality, we have

$$\begin{aligned}
\frac{\mathbb{E}[C_{RA}(\sigma)]}{C_{OPT}(\sigma)} &\le \frac{b_{1,1}/2 + 5b_2/2 + 3b_3/2 + 3b_4 + 1}{b_2 + b_3 + b_4^{100} + b_4^{010}} \\
&= \frac{-3b_{1,0}/2 - b_{1,1} + 4b_2 + 15b_4/2 + 4}{-b_{1,0} - b_{1,1} + 2b_2 + 3b_4 + b_4^{100} + b_4^{010} + 2} \quad \text{(by the substitution for } b_3 \text{ by Lemma 20)} \\
&\le \frac{-3b_{1,0}/2 + 4b_2 + 15b_4/2 - b_4^{100} + 4}{-b_{1,0} + 2b_2 + 3b_4 + b_4^{010} + 2} \quad \text{(by the substitution for } b_{1,1} \text{ by Eq. (2))} \\
&\le \frac{-5b_{1,0}/2 + 5b_2 + 15b_4/2 + b_4^{010} + 4}{-b_{1,0} + 2b_2 + 3b_4 + b_4^{010} + 2} \quad \text{(by the substitution for } b_4^{100} \text{ by Eq. (1))} \\
&= \frac{5}{2} \cdot \frac{-b_{1,0} + 2b_2 + 3b_4 + 2b_4^{010}/5 + 8/5}{-b_{1,0} + 2b_2 + 3b_4 + b_4^{010} + 2} < \frac{5}{2}.
\end{aligned}$$

◀

Our analysis of RA is exact by the following theorem.

▶ **Theorem 22.** *The competitive ratio of RA is at least $5/2$.*

5 Randomized Lower Bound

▶ **Lemma 23.** *Consider a randomized online algorithm RON. Suppose that a vertex v arrives at a vertex u. Let $p_u(p_v)$ denote the probability that $u \in D_{RON}(v)(v \in D_{RON}(v))$. Then, $p_u + p_v \geq 1$.*

Proof. Let p'_u be the probability that $u \in D_{RON}(u)$. Since RON's selection is irrevocable, the probability that $u \in D_{RON}(u)$ and $u \in D_{RON}(v)$ is greater than or equal to p'_u (Fact (a)). Next, we consider the case in which $u \notin D_{RON}(u)$. RON must select u or v to construct a dominating set immediately after v is revealed. Thus, the probability that either $u \in D_{RON}(v)$ or $v \in D_{RON}(v)$ is one. By the definition of p'_u, the probability that $u \notin D_{RON}(u)$ is $1 - p'_u$. Hence, the probability that both $u \notin D_{RON}(u)$ and either $u \in D_{RON}(v)$ or $v \in D_{RON}(v)$ is at least $1 - p'_u$. This probability together with Fact (a) shows that $p_u + p_v \geq p'_u + 1 - p'_u = 1$. ◀

▶ **Theorem 24.** *The competitive ratio of any randomized online algorithm is at least $4/3$.*

Proof. Consider a randomized online algorithm RON for the following input σ. Let m be any positive integer. We sketch an adversary constructing σ. First, the adversary gives a line of $2m$ vertices to ON. Then, for every two consecutive vertices on the line, the adversary determines whether an additional vertex will arrive at one of the two vertices. Specifically, if the probability that RON selects at least one of the two vertices is low, the adversary makes a new vertex arrive at the vertex.

For each $i = 1, 2, \ldots, 2m$, the i-th vertex v_i arrives at v_{i-1}. For each $j = 1, 2, \ldots, 2m$, let p_j be the probability that $v_j \in D_{RON}(v_{2m})$. Next, for each $\ell = 1, 2, \ldots, m$, vertices are revealed after the revelation of v_{2m} in the following two cases.

Case 1 ($\min\{p_{2\ell-1}, p_{2\ell}\} \geq 2/3$): A new vertex does not arrive at either $v_{2\ell-1}$ or $v_{2\ell}$. Since RON's selection is irrevocable, the expected cost of RON for $v_{2\ell-1}$ and $v_{2\ell}$ is at least $2 \cdot 2/3 = 4/3$.

Case 2 ($\min\{p_{2\ell-1}, p_{2\ell}\} < 2/3$): If $p_{2\ell-1} \leq p_{2\ell}$, then define $\ell_1 = 2\ell - 1$ and $\ell_2 = 2\ell$. Otherwise, define $\ell_2 = 2\ell - 1$ and $\ell_1 = 2\ell$. Then, a vertex u_{ℓ_1} arrives at v_{ℓ_1}. By Lemma 23, the probability that $v_{\ell_1} \in D_{RON}(u_{\ell_1})$ or $u_{\ell_1} \in D_{RON}(u_{\ell_1})$ is at least one (Fact (a)). Moreover, $p_{2\ell-1} + p_{2\ell} = p_{\ell_1} + p_{\ell_2} \geq 1$ also holds. Since $p_{\ell_1} < 2/3$ by the condition of Case 2, $p_{\ell_2} \geq 1/3$. Hence, the expected cost for v_{ℓ_1}, v_{ℓ_2} and u_{ℓ_1} is at least $1 + 1/3 = 4/3$.

Let x be the number of such ℓ with applying Case 1. Thus, the number of such ℓ with applying Case 2 is $m - x$. $\mathbb{E}[C_{RON}(\sigma)] \geq 4x/3 + 4(m - x)/3 = 4m/3$ by the above argument.

Next, we consider an offline algorithm OFF to give an upper bound on the cost of OPT. For such ℓ with applying Case 1, OFF selects $v_{2\ell-1}$ and for such ℓ with applying Case 2, selects v_{ℓ_1}. Thus, OFF selects m vertices, and the set of the m selected vertices is clearly a dominating set. By the optimality of OPT, $C_{OPT}(\sigma) \leq C_{OFF}(\sigma) = m$. Therefore, $\mathbb{E}[C_{RON}(\sigma)]/C_{OPT}(\sigma) \geq 4/3$. ◀

6 Conclusions

In this paper, we have conducted research on algorithms for an online variant of the minimum dominating set problem on trees and obtained the following results: First, we have shown that the competitive ratio of any deterministic algorithm is at least 3, which matches the upper bound shown in [4, 2]. Then, we have designed an algorithm whose competitive ratio is exactly 5/2 using randomization. Furthermore, we have shown that the competitive ratio of any randomized algorithm is at least 4/3.

We conclude this paper by providing open questions: (i) Online algorithms for dominating sets on several graph classes have been discussed in [4, 2] and optimal online algorithms have not yet known on some classes. Then, it is interesting to consider online algorithms on other classes in addition to them. (ii) Our algorithm *RA* is the first randomized algorithm for the online dominating set problem on trees and can achieve a competitive ratio smaller than that of any deterministic algorithm. Can we also obtain a better ratio on other classes using randomization? (iii) The gap between the randomized bounds shown in this paper is still large and thus, it is an obvious open problem to close the gap.

References

1 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

2 J. Boyar, S. J. Eidenbenz, L. M. Favrholdt, M. Kotrbčik, and K. S. Larsen. Online dominating set. In *Proc. of the 15th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 21:1–21:15, 2016.

3 D.-Z. Du and P.-J. Wan. *Connected Dominating Set: Theory and Applications.* Springer, 2013.

4 S. J. Eidenbenz. Online dominating set and variations on restricted graph classes. Technical report, Institute of Theoretical Computer Science, ETH Zürich, 2002.

5 F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *Proc. of the 30th international conference on Graph-Theoretic Concepts in Computer Science*, pages 245–256, 2004.

6 F.V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. In *Bulletin of the EATCS*, pages 47–77, 2005.

7 W. Goddard and M. A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313(7):839–854, 2013.

8 F. Grandoni. Independent domination in graphs: A survey and recent results. *Journal of Discrete Algorithms*, 4(2):209–214, 2006.

9 M. Henning and A. Yao. *Total Domination in Graphs.* Springer, 2013.

10 Y. Iwata. A faster algorithm for dominating set analyzed by the potential method. In *Proc. of the 6th international conference on Parameterized and Exact Computation*, pages 41–54, 2011.

11 G. H. King and W. G. Tzeng. On-line algorithms for the dominating set problem. *Information Processing Letters*, 61:11–14, 1997.

12 F. Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

13 I. Schiermeyer. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 156(17):3291–3297, 2008.

14 D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

15 J. M. M. van Rooij and H. L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011.

16 J. Y. Yu and P. H. J. Chong. A survey of clustering schemes for mobile ad hoc networks. *IEEE Communications Surveys and Tutorials*, 7(1):32–48, 2005.

Maximizing the Strong Triadic Closure in Split Graphs and Proper Interval Graphs

Athanasios L. Konstantinidis[1] **and Charis Papadopoulos**[2]

1 **Department of Mathematics, University of Ioannina, Greece**
skonstan@cc.uoi.gr

2 **Department of Mathematics, University of Ioannina, Greece**
charis@cs.uoi.gr

Abstract

In social networks the Strong Triadic Closure is an assignment of the edges with strong or weak labels such that any two vertices that have a common neighbor with a strong edge are adjacent. The problem of maximizing the number of strong edges that satisfy the strong triadic closure was recently shown to be NP-complete for general graphs. Here we initiate the study of graph classes for which the problem is solvable. We show that the problem admits a polynomial-time algorithm for two unrelated classes of graphs: proper interval graphs and trivially-perfect graphs. To complement our result, we show that the problem remains NP-complete on split graphs, and consequently also on chordal graphs. Thus we contribute to define the first border between graph classes on which the problem is polynomially solvable and on which it remains NP-complete.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.2.2 Graph Theory

Keywords and phrases strong triadic closure, polynomial-time algorithm, NP-completeness, split graphs, proper interval graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.53

1 Introduction

Predicting the behavior of a network is an important concept in the field of social networks [9]. Understanding the strength and nature of social relationships has found an increasing usefulness in the last years due to the explosive growth of social networks (see e.g., [2]). Towards such a direction the Strong Triadic Closure principle enables us to understand the structural properties of the underlying graph: it is not possible for two individuals to have a strong relationship with a common friend and not know each other [12]. Such a principle stipulates that if two people in a social network have a "strong friend" in common, then there is an increased likelihood that they will become friends themselves at some point in the future. Satisfying the Strong Triadic Closure is to characterize the edges of the underlying graph into weak and strong such that any two vertices that have a strong neighbor in common are adjacent. Since users interact and actively engage in social networks by creating strong relationships, it is natural to consider the MaxSTC problem: maximize the number of strong edges that satisfy the Strong Triadic Closure. The problem has been shown to be NP-complete for general graphs while its dual problem of minimizing the number of weak edges admits a constant factor approximation ratio [28].

In this work we initiate the computational complexity study of the MaxSTC problem in important classes of graphs. If the input graph is a P_3-free graph (i.e., a graph having no induced path on three vertices which is equivalent with a graph that consists of vertex-disjoint union of cliques) then there is a trivial solution by labeling strong all the edges. Such an observation might falsely lead into a graph modification problem, known as Cluster Deletion problem (see e.g., [3, 14]), in which we want to remove the minimum number of edges that correspond to the weak edges, such that the resulting graph does not contain a P_3 as an induced subgraph. More precisely the obvious reduction would consist in labeling the deleted edges in the instance of Cluster Deletion as weak, and the remaining ones as strong. However, this reduction fails to be correct due to the fact that the graph obtained by deleting the weak edges in an optimal solution of MaxSTC may contain an induced P_3, so long as those three vertices induce a triangle in the original graph (prior to deleting the weak edges). We stress that there are examples on split graphs (Figure 1) and proper interval graphs (Figure 3) showing such a difference.

To the best of our knowledge, no previous results were known prior to our work when restricting the input graph for the MaxSTC problem. It is not difficult to see that for bipartite graphs the MaxSTC problem has a simple polynomial-time solution by considering a maximum matching that represent the strong edges [15]. In fact such an argument regarding the maximum matching generalizes to the larger class of triangle-free graphs. Also notice that for triangle-free graphs a set of edges is a maximum matching if and only if it is formed by a solution for the Cluster Deletion problem. It is well-known that a maximum matching of a graph corresponds to a maximum independent set of its line graph that represents the adjacencies between the edges [10]. As previously noted, for general graphs it is not necessarily the case that a maximum matching corresponds to the optimal solution for MaxSTC. Here we show a similar characterization for MaxSTC by considering the adjacencies between the edges of a graph that participate in induced P_3's. Such a characterization allows us to exhibit properties towards an optimal solution of MaxSTC.

Due to the nature of the P_3 existence that enforce the labeling of weak edges, there is an interesting connection to problems related to the *square root* of a graph; a graph H is a *square root* of a graph G and G is the *square* of H if two vertices are adjacent in G whenever they are at distance one or two in H. Any graph does not have a square root (for example consider a simple path), but every graph contains a subgraph that has a square root. Although it is NP-complete to determine if a given chordal graph has a square root [21], there are polynomial-time algorithms when the input is restricted to bipartite graphs [20], or proper interval graphs [21], or trivially-perfect graphs [25]. Among several square roots that a graph may have, one can choose the square root with the maximum or minimum number of edges [5, 23]. The relationship between MaxSTC and to that of determining square roots can be seen as follows. In the MaxSTC problem we are given a graph G and we want to select the maximum possible number of edges, at most one from each induced P_3 in G. Thus we need to find the largest subgraph (in terms of the number of its edges) H of G such that the square of H is a subgraph of G. However the known results related to square roots were concerned with deciding if the whole graph has a (maximum or minimum) square root and there are no such equivalent formulations related to the largest square root.

Our main motivation is to understand the complexity of the problem on subclasses of chordal graphs, since the class of chordal graphs (i.e., graphs having no chordless cycle of length at least four) finds important applications in both theoretical and practical areas related to social networks [1, 19, 26]. More precisely two famous properties can be found in social networks. For most known social and biological networks their diameter, that is, the length of the longest shortest path between any two vertices of a graph, is known to be a small constant [17]. On the other hand it has been shown that the most prominent social network

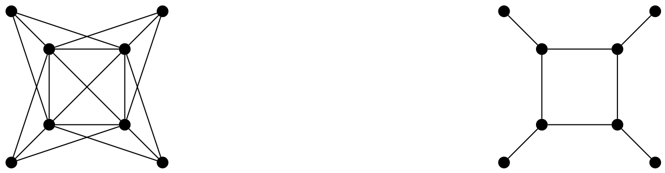


Figure 1 A split graph G is shown to the left side. The right side depicts a solution for MaxSTC on G where the weak edges are exactly the edges of G that are not shown.

subgraphs are cliques, whereas highly infrequent induced subgraphs are cycles of length four [29]. Thus it is evident that subclasses of chordal graphs are close related to such networks, since they have rather small diameter (e.g., split graphs or trivially-perfect graphs) and are characterized by the absence of chordless cycles (e.g., proper interval graphs). Towards such a direction we show that MaxSTC is NP-complete on split graphs and consequently also on chordal graphs. On the positive side, we present the first polynomial-time algorithm for computing MaxSTC on proper interval graphs. Proper interval graphs, also known as unit interval graphs or indifference graphs, form a subclass of interval graphs and they are unrelated to split graphs [27]. By our result they form the first graph class, other than triangle-free graphs, for which MaxSTC is shown to be polynomial time solvable. In order to obtain our algorithm, we take advantage of their clique path (consecutive arrangement of maximal cliques) and apply a dynamic programming on subproblems defined by passing the clique path in its natural ordering. Our structural proofs on proper interval graphs can be seen as useful tools towards settling the complexity of MaxSTC on interval graphs. Furthermore by considering the characterization of the induced P_3's mentioned earlier, we show that MaxSTC admits a simple polynomial-time solution on trivially-perfect graphs (i.e., graphs having no induced P_4 or C_4).

2 Preliminaries

We refer to [4] for our standard graph terminology. Given a graph $G = (V, E)$, a *strong-weak labeling* on the edges of G is a function λ that assigns to each edge of $E(G)$ one of the labels *strong* or *weak*; i.e., $\lambda : E(G) \to \{\text{strong}, \text{weak}\}$. An edge that is labeled strong (resp., weak) is simply called *strong* (resp. *weak*). The *strong triadic closure* of a graph G is a strong-weak labeling λ such that for any two strong edges $\{u, v\}$ and $\{v, w\}$ there is a (weak or strong) edge $\{u, w\}$. In other words, in a strong triadic closure there is no pair of strong edges $\{u, v\}$ and $\{v, w\}$ such that $\{u, w\} \notin E(G)$. The problem of computing a maximum strong triadic closure, denoted by MaxSTC, is to find a strong-weak labeling on the edges of $E(G)$ that satisfies the strong triadic closure and has the maximum number of strong edges. Note that its dual problem asks for the minimum number of weak edges. Here we focus on maximizing the number of strong edges in a strong triadic closure.

Let G be a strong-weak labeled graph. We denote by (E_S, E_W) the partition of $E(G)$ into strong edges E_S and weak edges E_W. The graph spanned by E_S is the graph $G \setminus E_W$. For a vertex $v \in V(G)$ we say that the *strong neighbors* of v are the other endpoints of the strong edges incident to v. We denote by $N_S(v) \subseteq N(v)$ the strong neighbors of v. Similarly we say that a vertex u is *strongly adjacent* to v if u is adjacent to v and $\{u, v\}$ is strong.

▶ **Observation 1.** *Let $G = (E_S, E_W)$ be a strong-weak labeled graph. G satisfies the strong triadic closure if and only if for every P_3 in $G \setminus E_W$, the vertices of P_3 induce a K_3 in G.*

3 MaxSTC on split graphs

Here we provide an NP-hardness result for MaxSTC on split graphs. A graph $G = (V, E)$ is a *split graph* if V can be partitioned into a clique C and an independent set I, where (C, I) is called a *split partition* of G. Split graphs form a subclass of the larger and widely known graph class of *chordal graphs*, which are the graphs that do not contain induced cycles of length 4 or more as induced subgraphs. It is known that split graphs are self-complementary, that is, the complement of a split graph remains a split graph. Hereafter for two vertices u and v we say that u *sees* v if $\{u, v\} \in E(G)$; otherwise, we say that u *misses* v.

▶ **Lemma 2.** *Let $G = (V, E)$ be a split graph with a split partition (C, I). Let E_S be the set of strong edges in an optimal solution for* MaxSTC *on G and let I_W be the vertices of I that are incident to at least one edge of E_S.*

1. *If every vertex of I_W misses at least three vertices of C in G then $E_S = E(C)$.*
2. *If every vertex of I_W misses exactly one vertex of C in G then $|E_S| \leq |E(C)| + \lfloor \frac{|I_W|}{2} \rfloor$.*

Proof. Let w_i be a vertex of I and let B_i be the set of vertices in C that are non-adjacent to w_i. Let A_i be the strong neighbors of w_i in an optimal solution. For the edges of the clique, there are $|A_i||B_i|$ weak edges due to the strong triadic closure. Moreover any vertex w_j of $I \setminus \{w_i\}$ cannot have a strong neighbor in A_i. This means that $A_i \cap A_j = \emptyset$. Notice, however, that both sets $B_i \cap B_j$ and $A_i \cap B_j$ are not necessarily empty.

Observe that I_W contains the vertices of I that are incident to at least one strong edge. Let $E(A, B)$ be the set of weak edges that have one endpoint in A_i and the other endpoint in B_i, for every $1 \leq i \leq |I_W|$. We show that $2|E(A, B)| \geq \sum_{w_i \in I_W} |A_i||B_i|$. Let $\{a, b\} \in E(A, B)$ such that $a \in A_i$ and $b \in B_i$. Assume that there is a pair A_j, B_j such that $\{a, b\}$ is an edge between A_j and B_j, for $j \neq i$. Then a cannot belong to A_j since $A_i \cap A_j = \emptyset$. Thus $a \in B_j$ and $b \in A_j$. Therefore for every edge $\{a, b\} \in E(A, B)$ there are at most two pairs (A_i, B_i) and (A_j, B_j) for which $a \in A_i \cup B_j$ and $b \in B_i \cup A_j$. This means that every edge of $E(A, B)$ is counted at most twice in $\sum_{w_i \in I_W} |A_i||B_i|$.

For any two edges $\{u, v\}, \{v, z\} \in E(C) \setminus E(A, B)$, observe that they satisfy the strong triadic closure since there is the edge $\{u, z\}$ in G. Thus the strong edges of the clique are exactly the set of edges $E(C) \setminus E(A, B)$. In total by counting the number of strong edges between the independent set and the clique, we have $|E_S| = |E(C) \setminus E(A, B)| + \sum_{w_i \in I_W} |A_i|$. Since $2|E(A, B)| \geq \sum_{w_i \in I_W} |A_i||B_i|$, we get

$$|E_S| \leq |E(C)| + \sum_{w_i \in I_W} |A_i| \left(1 - \left\lfloor \frac{|B_i|}{2} \right\rfloor\right).$$

Now the first claim of the lemma holds because $|B_i| \geq 3$ so that $I_W = \emptyset$. For the second claim we show that for every vertex of I_W, $|A_i| = 1$. Let $w_i \in I_W$ such that $|A_i| \geq 2$ and let $B_i = \{b_i\}$. Recall that no other vertex of I_W has strong neighbors in A_i. Also note that there is at most one vertex w_j in I_W that has b_i as a strong neighbor. If such a vertex w_j exist and for the vertex b_j of the clique that misses w_j it holds $b_j \in A_i$, then we let $v = b_j$; otherwise we choose v as an arbitrary vertex of A_i. Observe that no vertex of $I \setminus \{w_i\}$ has a strong neighbor in $A_i \setminus \{v\}$ and only $w_j \in I_W$ is strongly adjacent to b_i. Then we label weak the $|A_i| - 1$ edges between w_i and the vertices of $A_i \setminus \{v\}$ and we label strong the $|A_i| - 1$ edges between b_i and the vertices of $A_i \setminus \{v\}$. Making strong the edges between b_i and the vertices of $A_i \setminus \{v\}$ does not violate the strong triadic closure since every vertex of $C \cup \{w_j\}$ is adjacent to every vertex of $A_i \setminus \{v\}$. Therefore for every vertex $w_i \in I_W$, $|A_i| = 1$ and by substituting $|B_i| = 1$ in the formula for $|E_S|$ we get the claimed bound. ◀

In order to give the reduction, we introduce the following problem that we call *maximum disjoint non-neighborhood*: given a split graph (C, I) where every vertex of I misses three vertices from C, we want to find the maximum subset S_I of I such that the non-neighborhoods of the vertices of S_I are pairwise disjoint. In the corresponding decision version, denoted by MaxDisjointNN, we are also given an integer k and the problem asks whether $|S_I| \geq k$. The polynomial-time reduction to MaxDisjointNN is given from the classical NP-complete problem 3-Set Packing [18]: given a universe $\mathcal{U}$ of n elements, a family $\mathcal{F}$ of triplets of $\mathcal{U}$, and an integer k, the problem asks for a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \geq k$ such that all triplets of $\mathcal{F}'$ are pairwise disjoint.

▶ **Corollary 3.** MaxDisjointNN *is NP-complete on split graphs.*

Now we turn to our original problem MaxSTC. The decision version of MaxSTC takes as input a graph G and an integer k and asks whether there is strong-weak labeling of the edges of G that satisfies the strong triadic closure with at least k strong edges.

▶ **Theorem 4.** *The decision version of* MaxSTC *is NP-complete on split graphs.*

Proof. Given a strong-weak labeling (E_S, E_W) of a split graph $G = (C, I)$, checking whether (E_S, E_W) satisfies the strong triadic closure amounts to check in $G \setminus E_W$ whether there is a non-edge in G between the endvertices of every P_3 according to Observation 1. Thus by listing all P_3's of $G \setminus E_W$ the problem belongs to NP. Next we give a polynomial-time reduction to MaxSTC from the MaxDisjointNN problem on split graphs which is NP-complete by Corollary 3. Let (G, k) be an instance of MaxDisjointNN where $G = (C, I)$ is a split graph such that every vertex of the independent set I misses exactly three vertices from the clique C. For a vertex $w_i \in I$, we denote by B_i the set of the three vertices in C that are non-adjacent to w_i. Let $n = |C|$. We extend G and construct another split graph G' as follows (see Figure 2):

- We add n vertices $y_1, \ldots, y_n$ in the clique that constitutes the set C_Y.
- We add n vertices $x_1, \ldots, x_n$ in the independent set that constitutes the set I_X.
- For every $1 \leq i \leq n$, y_i is adjacent to all vertices of $(C \cup C_Y \cup I \cup I_X) \setminus \{x_i\}$.
- For every $1 \leq i \leq n$, x_i is adjacent to all vertices of $(C \cup C_Y) \setminus \{y_i\}$.

Thus w_i misses only the vertices of B_i from the clique. By construction it is clear that G' is a split graph with a split partition $(C \cup C_Y, I \cup I_X)$. Notice that the clique $C \cup C_Y$ has $2n$ vertices and $G = G'[I \cup C]$.

We claim that G has a solution for MaxDisjointNN of size at least k if and only if G' has a strong triadic closure with at least $n(2n-1) + \lfloor \frac{n}{2} \rfloor + \lceil \frac{k}{2} \rceil$ strong edges. Due to space restriction, we only show the one direction.

Assume that $\{w_1, \ldots, w_k\} \subseteq I$ is a solution for MaxDisjointNN on G of size at least k. Since the sets $B_1, \ldots, B_k$ are pairwise disjoint, there are k distinct vertices $y_1, \ldots, y_k$ in C_Y such that $k \leq n$. We will give a strong-weak labeling for the edges of G' that fulfills the strong triadic closure and has at least the claimed number of strong edges. For simplicity, we describe only the strong edges; the edges of G' that are not given are all labeled weak. We label the edges between each vertex w_i, y_i, x_i and the three vertices of each set B_i, for $1 \leq i \leq k$ as follows:

- The edges of the form $\{y_i, v\}$ are labeled strong if $v \in (C \cup C_Y) \setminus B_i$ or $v = w_i$.
- The edges between x_i and the three vertices of B_i are labeled strong.

Next we label the edges incident to the rest of the vertices. No edge incident to a vertex of $I \setminus \{w_1, \ldots, w_k\}$ is labeled strong. For every vertex $u \in C \setminus (B_1 \cup \cdots \cup B_k)$ we label the edge $\{u, v\}$ strong if $v \in (C \cup C_Y)$. Let $C'_Y = \{y_{k+1}, \ldots, y_n\}$ and let $I'_X = \{x_{k+1}, \ldots, x_n\}$.

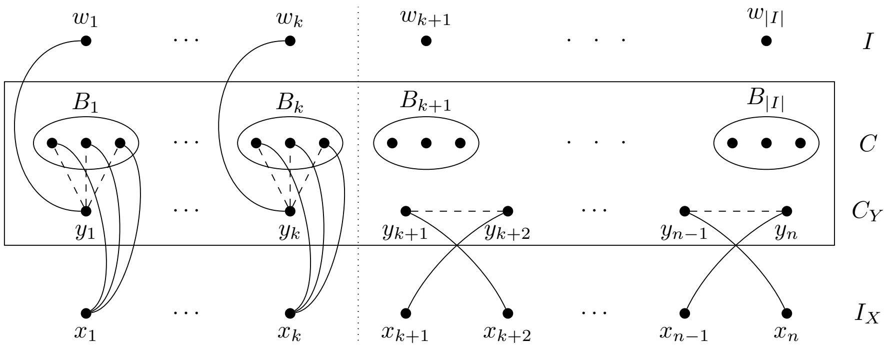


Figure 2 The split graph $(C \cup C_Y, I \cup I_X)$ given in the polynomial-time reduction. Every vertex w_i misses the vertices of B_i and sees the vertices of $(C \cup C_Y) \setminus B_i$. Every vertex x_i misses y_i and sees the vertices of $(C \cup C_Y) \setminus \{y_i\}$. The sets $B_1, \ldots, B_k$ are pairwise disjoint whereas for every set B_j, $k < j \leq |I|$, there is a set B_i, $1 \leq i \leq k$, such that $B_i \cap B_j \neq \emptyset$. The drawn edges correspond to the strong edges between the independent set and the clique, and the dashed edges are the only weak edges in the clique $C \cup C_Y$.

Recall that every vertex x_{k+j} is adjacent to every vertex of $C'_Y \setminus \{y_{k+j}\}$. Let $\ell = \lfloor \frac{n-k}{2} \rfloor$. Let $M = \{e_1, \ldots, e_\ell\}$ be a maximal set of pairwise non-adjacent edges in $G'[C'_Y]$ where $e_j = \{y_{k+2j-1}, y_{k+2j}\}$, for $j \in \{1, \ldots, \ell\}$; note that M is a maximal matching of $G'[C'_Y]$. For every vertex $y \in C'_Y$, we label the edge $\{y, v\}$ strong if $v \in (C \cup C_Y) \setminus \{y'\}$ such that $\{y, y'\} \in M$. Moreover, for $j \in \{1, \ldots, \ell\}$, the edges $\{x_{k+2j-1}, y_{k+2j}\}$ and $\{x_{k+2j}, y_{k+2j-1}\}$ are labeled strong. Note that if $n - k$ is odd then no edge incident to the unique vertex y_n belongs to M and all edges between y_n and the vertices of $C \cup C_Y$ are labeled strong; in such a case also note that no edge incident to x_n is strong.

Let us show that such a labeling fulfills the strong triadic closure. Any labeling for the edges inside $G'[C \cup C_Y]$ is satisfied since $G'[C \cup C_Y]$ is a clique. Also note that there are no two adjacent strong edges that have a common endpoint in the clique $C \cup C_Y$ and the two other endpoints in the independent set $I \cup I_X$. If there are two strong edges incident to the same vertex v of the independent set then $v \in \{x_1, \ldots, x_k\}$ and $N_S[v] = B_i$ which is a clique. Assume that there are two adjacent strong edges $\{u, v\}$ and $\{v, z\}$ such that $u \in I \cup I_X$, and $v, z \in C \cup C_Y$.

- If $u \in \{w_1, \ldots, w_k\}$ then $\{u, z\} \in E(G')$ since every w_i misses only the vertices of B_i.
- If $u \in \{x_1, \ldots, x_k\}$ then $v \in B_i$ and $\{u, z\} \in E(G')$ since every vertex x_i misses only y_i.
- If $u \in I_X \setminus \{x_1, \ldots, x_k\}$ then the strong neighbors of v in $C \cup C_Y$ are adjacent to u in G' since for the only non-neighbor of u in $C \cup C_Y$ there is a weak edge incident to v.

Recall that there is no strong edge incident to the vertices of $I \setminus \{w_1, \ldots, w_k\}$. Therefore the given strong-weak labeling fulfills the strong triadic closure.

Observe that the number of vertices in $C \cup C_Y$ is $2n$. There are exactly $3k + \ell$ weak edges in $G'[C \cup C_Y]$. Thus the number of strong edges in $G'[C \cup C_Y]$ is $n(2n-1) - 3k - \ell$. There are k strong edges incident to $\{w_1, \ldots, w_k\}$, $3k$ strong edges incident to $\{x_1, \ldots, x_k\}$, and 2ℓ strong edges incident to $I_X \setminus \{x_1, \ldots, x_k\}$. Thus the total number of strong edges is $n(2n-1) - 3k - \ell + k + 3k + 2\ell = n(2n-1) + \ell + k$ and by substituting $\ell = \lfloor \frac{n-k}{2} \rfloor$ we get the claimed bound. ◀

4 Computing MaxSTC on proper interval graphs

Due to the NP-completeness on split graphs given in Theorem 4, it is natural to consider interval graphs that form another well-studied subclass of chordal graphs. However besides few observations of this section that may be applied for interval graphs, we found several unresolved technicalities. Moreover, to the best of our knowledge, the complexity of the close-related Cluster Deletion problem remains unresolved on interval graphs [3]. Thus we further restrict the input to the class of proper interval graphs that form a proper subclass of interval graphs. Our polynomial solution for MaxSTC on proper interval graphs can be seen as a first step towards determining its complexity on interval graphs.

A graph is a *proper interval graph* if there is a bijection between its vertices and a family of closed intervals of the real line such that two vertices are adjacent if and only if the two corresponding intervals overlap and no interval is properly contained in another interval. A vertex ordering σ is a linear arrangement $\sigma = \langle v_1, \ldots, v_n \rangle$ of the vertices of G. For a vertex pair x, y we write $x \preceq y$ if $x = v_i$ and $y = v_j$ for some indices $i \leq j$; if $x \neq y$ which implies $i < j$ then we write $x \prec y$. The first position in σ will be referred to as the *left end* of σ, and the last position as the *right end*. We will use the expressions *to the left of*, *to the right of*, *leftmost*, and *rightmost* accordingly.

A vertex ordering σ for G is called a *proper interval ordering* if for every vertex triple x, y, z of G with $x \prec y \prec z$, $\{x, z\} \in E(G)$ implies $\{x, y\}, \{y, z\} \in E(G)$. Proper interval graphs are characterized as the graphs that admit such orderings, that is, a graph is a proper interval graph if and only if it has a proper interval ordering [24]. We only consider this vertex ordering characterization for proper interval graphs. Moreover it can be decided in linear time whether a given graph is a proper interval graph, and if so, a proper interval ordering can be generated in linear time [24]. It is clear that a vertex ordering σ for G is a proper interval ordering if and only if the reverse of σ is a proper interval ordering. Two adjacent vertices u and v are called *twins* if $N[u] = N[v]$. A connected proper interval graph without twin vertices has a unique proper interval ordering σ up to reversal [8, 16]. Figure 3 shows a proper interval graph with its proper interval ordering.

Let us turn our attention to the MaxSTC problem. Instead of maximizing the strong edges of the original graph G, we will look at the maximum independent set of the following graph that we call the *line-incompatibility* graph $\widehat{G}$ of G: for every edge of G there is a node in $\widehat{G}$ and two nodes of $\widehat{G}$ are adjacent if and only if the vertices of the corresponding edges induce a P_3 in G. In a different context the notion of line-incompatibility has already been considered under the term *Gallai graph* in [22] or as an auxiliary graph in [5]. Note that the line-incompatibility graph of G is a subgraph of the line graph[1] of G. Moreover observe that for a graph G, its line graph and its line-incompatibility graph coincide if and only if G is a triangle-free graph.

▶ **Proposition 5.** *A subset S of edges $E(G)$ is an optimal solution for* MaxSTC *of G if and only if S is a maximum independent set of $\widehat{G}$.*

Therefore we seek for the optimal solution of G by looking at a solution for a maximum independent set of $\widehat{G}$. As a byproduct, if we are interested in minimizing the number of weak edges then we ask for the minimum vertex cover of $\widehat{G}$. We denote by $I_{\widehat{G}}$ the maximum independent set of $\widehat{G}$. To distinguish the vertices of $\widehat{G}$ with those of G we refer to the

[1] The *line graph* of G is the graph having the edges of G as vertices and two vertices of the line graph are adjacent if and only if the two original edges are incident in G.

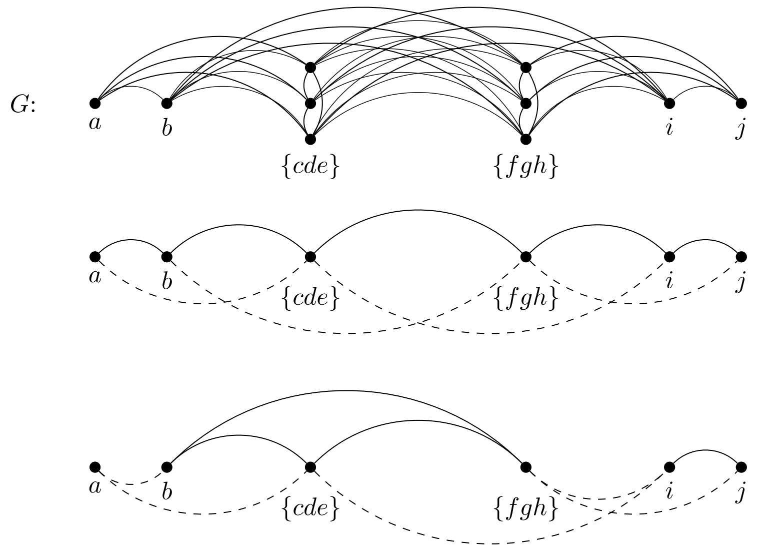


Figure 3 A proper interval graph G and its proper interval ordering. The vertices $\{c, d, e\}$ and $\{f, g, h\}$ form twin sets in G. The two lower orderings depict two solutions for MaxSTC on G. A solid edge corresponds to a strong edge, whereas a dashed edge corresponds to a weak edge. Observe that the upper solution contains larger number of strong edges than the lower one. Also note that the lower solution consists an optimal solution for the Cluster Deletion problem on G.

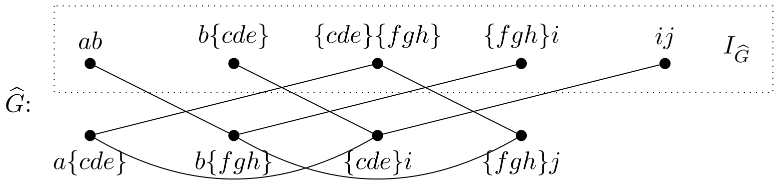


Figure 4 The line-incompatibility graph $\widehat{G}$ of the proper interval graph G given in Figure 3. The set $I_{\widehat{G}}$ is a maximum weighted independent set of $\widehat{G}$, by taking into account the weight of each node (i.e., an edge of G) that corresponds to the number of the twin vertices of its endpoints in G (see Lemma 6).

former as nodes and to the latter as vertices. For an edge $\{u, v\}$ of G we denote by uv the corresponding node of $\widehat{G}$. Figure 4 shows the line-incompatibility graph of the proper interval graph given in Figure 3.

A natural contraction for several graph problems is to group twin vertices since they play the same role on the given graph. With the next result, we show that this is indeed the case for the MaxSTC problem.

▶ **Lemma 6.** *Let x and y be twin vertices of a graph G. Then there is an optimal solution $I_{\widehat{G}}$ such that $xy \in I_{\widehat{G}}$ and for every vertex $u \in N(x)$, $xu \in I_{\widehat{G}}$ if and only if $yu \in I_{\widehat{G}}$.*

Lemma 6 suggests to consider a graph G that has no twin vertices as follows. We partition $V(G)$ into sets of twins. For every twin set W_x we choose an arbitrary vertex x and remove all its twin vertices except x from G. Let G' be the resulting graph that has no twin vertices. For every edge $\{x, y\}$ of G' we assign a weight equal to the product $|W_x| \cdot |W_y|$. This value corresponds to all edges of the original graph G between the vertices of W_x and W_y. The line-incompatibility graph $\widehat{G'}$ of G' is constructed as defined above with the only difference that a node of $\widehat{G'}$ has weight equal to the weight of its corresponding edge in G'. Let $I_{\widehat{G'}}$ be a *maximum weighted independent set*, that is an independent set of $\widehat{G'}$ such that the sum of the weights of its nodes is maximized. Then by Lemma 6 we have $I_{\widehat{G}} = I_{\widehat{G'}} \cup S(W)$ where $S(W)$ contains $|W_x|(|W_x|-1)/2$ nodes for every twin set W_x. Therefore we are interested in computing a maximum weighted independent set of $\widehat{G'}$. Also note that G' is an induced subgraph of the original graph G. In order to avoid heavier notation we refer to $\widehat{G'}$ as $\widehat{G}$ by assuming that G has no twin vertices and every vertex of G has a positive weight.

Before reaching the details of our algorithm for proper interval graphs, let us highlight the difference between the optimal solution for MaxSTC and the optimal solution for the Cluster Deletion. As already explained in the Introduction a solution for Cluster Deletion satisfies the strong triadic closure, though the converse is not necessarily true. In fact such an observation carries out for the class of proper interval graphs as shown in the example given in Figure 3. For the Cluster Deletion problem twin vertices can be grouped together following a similar characterization with Lemma 6, as proved in [3]. This means that the P_3-free graph depicted in the lower part of Figure 3 that is obtained by removing its weak edges (i.e., the dashed drawn lines) is an optimal solution for Cluster Deletion problem on the given proper interval graph. Therefore when restricted to proper interval graphs the optimal solution for Cluster Deletion does not necessarily imply an optimal solution for MaxSTC.

Let G be a proper interval graph and let σ be a proper interval ordering for G. We say that a solution $I_{\widehat{G}}$ has the *consecutive strong property* with respect to σ if for any three vertices x, y, z of G with $x \prec y \prec z$ the following holds: $xz \in I_{\widehat{G}}$ implies $xy, yz \in I_{\widehat{G}}$. Our task is to show that such an optimal ordering exists. We start by characterizing the optimal solution $I_{\widehat{G}}$ with respect to the proper interval ordering σ.

▶ **Lemma 7.** *Let x, y, z be three vertices of a proper interval graph G such that $x \prec y \prec z$. If $xz \in I_{\widehat{G}}$ then $xy \in I_{\widehat{G}}$ or $yz \in I_{\widehat{G}}$.*

Proof. We show that at least one of xy or yz belongs to $I_{\widehat{G}}$. Assume towards a contradiction that neither xy nor yz belong to $I_{\widehat{G}}$. Consider the node xy in $\widehat{G}$. If xy is adjacent to a node $xx_\ell \in I_{\widehat{G}}$ then $\{x_\ell, y\} \notin E(G)$. Then observe that $x_\ell \prec y$ because $x \prec y$ and $\{x_\ell, y\} \notin E(G)$. Since both xx_ℓ and xz belong to $I_{\widehat{G}}$, $\{x_\ell, z\} \in E(G)$. This however contradicts the proper interval ordering because $x_\ell \prec y \prec z$, $\{x_\ell, z\} \in E(G)$ and y is non-adjacent to x_ℓ. Thus xy is non-adjacent to any node $xx_\ell \in I_{\widehat{G}}$ and, in analogous fashion, yz is non-adjacent to any node $zz_r \in I_{\widehat{G}}$.

Now assume that xy is adjacent to a node $yy_r \in I_{\widehat{G}}$ and yz is adjacent to a node $y_\ell y \in I_{\widehat{G}}$. This means that $\{x, y_r\} \notin E(G)$ and $\{z, y_\ell\} \notin E(G)$. Since $\{x, z\} \in E(G)$, by the proper interval ordering we have $y_\ell \prec x \prec y \prec z \prec y_r$. Then notice that $\{y_\ell, y_r\} \in E(G)$, because both $yy_r, yy_\ell \in I_{\widehat{G}}$. By the proper interval ordering we know that both x and z are adjacent to y_ℓ, y_r, leading to a contradiction to the assumptions $\{x, y_r\} \notin E(G)$ and $\{z, y_\ell\} \notin E(G)$. Therefore at least one of xy or yz belongs to $I_{\widehat{G}}$. ◀

Thus by Lemma 7 we have two symmetric cases to consider. The next characterization suggests that there is a fourth vertex with important properties in each corresponding case.

▶ **Lemma 8.** *Let x, y, z be three vertices of a proper interval graph G such that $x \prec y \prec z$ and $xz \in I_{\widehat{G}}$.*

- *If $xy \notin I_{\widehat{G}}$ and $yz \in I_{\widehat{G}}$ then xy is non-adjacent to any node $x_\ell x \in I_{\widehat{G}}$ and there is a vertex w such that $yw \in I_{\widehat{G}}$, $\{x, w\} \notin E(G)$, and $z \prec w$.*
- *If $xy \in I_{\widehat{G}}$ and $yz \notin I_{\widehat{G}}$ then yz is non-adjacent to any node $zz_r \in I_{\widehat{G}}$ and there is a vertex w such that $wy \in I_{\widehat{G}}$, $\{w, z\} \notin E(G)$ and $w \prec x$.*

Now we are ready to show that there is an optimal solution that has the described properties with respect to the given proper interval ordering.

▶ **Lemma 9.** *There exists an optimal solution $I_{\widehat{G}}$ that has the consecutive strong property with respect to σ.*

Lemma 9 suggests to find an optimal solution that has the consecutive strong property with respect to σ. In fact by Proposition 5 and the proper interval ordering, this reduces to computing the largest proper interval subgraph H of G such that the vertices of every P_3 of H induce a clique in G.

Let G be a proper interval graph and let $\sigma = \langle v_1, \ldots, v_n \rangle$ be its proper interval ordering. For a vertex v_i we denote by $\ell(i)$ and $r(i)$ the positions of its leftmost and rightmost neighbors, respectively, in σ. Observe that for any two vertices $v_i \prec v_j$ in σ, $v_{\ell(i)} \preceq v_{\ell(j)}$ and $v_{r(i)} \preceq v_{r(j)}$ [8]. For $1 \leq j \leq r(1)$, let $V_j = \{v_1, \ldots, v_j\}$, that is, V_j contains the *first* j vertices in σ. Observe that any subset of vertices of V_j induces a clique in G. For the set V_j we denote by $B(V_j)$ the value that corresponds to the total weight of the edges incident to v_1 and each of $v_2, \ldots, v_j$.

Let $A(G)$ be the value of an optimal solution $I_{\widehat{G}}$ for G. For technical reasons we assume that $v_i v_i$ is an edge of G with weight equal to zero. For every vertex v_i we denote by $L[i] = i$ and $R[i] = r(i)$. The vectors L and R are called the *rightmost limits* of the vertices. Let $A(G, L, R)$ be the value of the optimal solution $I(G, L, R)$ such that for every vertex v_i its rightmost strong neighbor v_k lies between the positions $L[i]$ and $R[i]$. That is, for every vertex v_i with $v_i v_k \in I(G, L, R)$ and k as large as possible, $L[i] \leq k \leq R[i]$ holds. The key idea is that we try all positions j among the rightmost limits of the first vertex v_1. This is achieved through the consecutive strong property by making v_1 strongly adjacent to every vertex of V_j. Then, however, we need to update accordingly the rightmost limits of each vertex of V_j in order to obey the consecutive strong property. As a trivial case observe that if G contains exactly one vertex then $A(G) = 0$.

▶ **Lemma 10.** *Let G be a proper interval graph and let L and R be the rightmost limits of the vertices with respect to σ. Then $A(G) = A(G, L, R)$ and*

$$A(G, L, R) = \max_{L[1] \leq j \leq R[1]} \{A(G - \{v_1\}, L_j, R_j) + B(V_j)\},$$

$$\text{where } L_j[i] = \begin{cases} j & \text{if } i \leq j, \\ L[i] & \text{otherwise} \end{cases} \quad \text{and} \quad R_j[i] = \begin{cases} \min\{r(1), R[i]\} & \text{if } i \leq j, \\ R[i] & \text{otherwise.} \end{cases}$$

Now we are equipped with our necessary tools in order to obtain our main result, namely a polynomial-time algorithm that solves the MaxSTC problem on proper interval graphs.

▶ **Theorem 11.** *There is a polynomial-time algorithm that computes the* MaxSTC *of a proper interval graph.*

5 Concluding remarks

Given the first study with positive and negative results for the MaxSTC problem on restricted input, there are some interesting open problems. As we pointed out MaxSTC is more difficult than Cluster Deletion in the following sense: a solution for Cluster Deletion forms a solution for MaxSTC but the converse is not necessarily true. We have given examples showing that such an observation carries out for split graphs as well as for proper interval graphs. Despite the structural difference of both problems, our result on split graphs points out an important and interesting complexity difference between the two problems: on split graphs Cluster Deletion has already been shown to be polynomially solvable [3] whereas we prove that MaxSTC remains NP-complete. It is interesting to explore other graph classes that exhibit the same behavior. Towards such a direction observe that every problem expressible in monadic second order logic (MSOL) with quantification over the vertices and vertex sets can be solved in linear time for graphs of bounded treewidth [7]. Indeed, MaxSTC can be formulated in MSOL: (i) the edges are partitioned into two subsets E_S, E_W (i.e., a strong-weak labeling), (ii) the endpoints of every path of length two spanned by the edges of E_S have an edge (i.e., satisfy the strong triadic closure), and (iii) $|E_S|$ is as large as possible. Therefore there is a linear-time algorithm for MaxSTC on graphs of bounded treewidth [7].

Apart from the structural properties that we proved for the solution on proper interval graphs, the complexity of MaxSTC on interval graphs is still open. Moreover it is natural to characterize the graphs for which their line-incompatibility graph is perfect. Such a characterization will lead to further polynomial cases of MaxSTC, since the problem of finding a maximum independent set of perfect graphs admits a polynomial solution [13]. A typical example is the class of bipartite graphs for which their line graph coincides with their line-incompatibility graph and it is known that the line graph of a bipartite graph is perfect (see for e.g., [4]). As we show next, another paradigm of this type is the class of trivially-perfect graphs.

A graph G is called *trivially-perfect* (also known as *quasi-threshold*) if for each induced subgraph H of G, the number of maximal cliques of H is equal to the maximum size of an independent set of H. It is known that the class of trivially-perfect graphs coincides with the class of (P_4, C_4)-free graphs, that is every trivially-perfect graph has no induced P_4 or C_4 [11]. A *cograph* is a graph without an induced P_4, that is a cograph is a P_4-free graph. Hence trivially-perfect graphs form a subclass of cographs.

▶ **Theorem 12.** *The line-incompatibility graph of a trivially-perfect graph is a cograph.*

By Theorem 12 and the fact that the maximum independent set of a cograph can be computed in linear time [6], MaxSTC can be solved in polynomial time on trivially-perfect graphs. We would like to note that the line-incompatibility graph of a cograph or a proper interval graph is not necessarily a perfect graph.

More general there are extensions and variations of the MaxSTC problem that are interesting to consider as proposed in [28]. An interesting and realistic problem is to allow multiple types of strong edges $S_0, S_1, \ldots, S_k$ that do not allow violating "ordered" P_3's. More precisely the objective is to partition the edges of G into $S_0, S_1, \ldots, S_k$ with $k \geq 1$ so that there is no pair of edges $\{u, v\} \in S_i$ and $\{v, w\} \in S_i$ such that $\{u, w\} \notin E(G)$ and $|S_1| + \cdots + |S_k|$ is as large as possible.

References

1 A. B. Adcock, B. D. Sullivan, and M. W. Mahoney. Tree decompositions and social graphs. *Internet Mathematics*, 12:315–361, 2016.

2 L. Backstrom and J. Kleinberg. Romantic partnerships and the dispersion of social ties: a network analysis of relationship status on facebook. In *Proceedings of CSCW 2014*, pages 831–841, 2014.

3 F. Bonomo, G. Durán, and M. Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, 600:59–69, 2015.

4 A. Brandstädt, V. B. Le, and J. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.

5 M. Cochefert, J.-F. Couturier, P. A. Golovach, D. Kratsch, and D. Paulusma. Parameterized algorithms for finding square roots. *Algorithmica*, 74:602–629, 2016.

6 D.G. Corneil, H. Lerchs, and L.K. Stewart. Complement reducible graphs. *Discrete Applied Mathematics*, 3:163–174, 1981.

7 B. Courcelle. The monadic second-order logic of graphs i: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

8 X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25:390–403, 1996.

9 D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

10 J. Edmonds. Paths, trees and flowers. *Canad. J. Math.*, 17:449–467, 1965.

11 M.C. Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24:105–107, 1978.

12 M. Granovetter. The strength of weak ties. *American J. of Sociology*, 78:1360–1380, 1973.

13 M. Grötschel. Polynomial algorithms for perfect graphs. *North-Holland Mathematics Studies*, 21:325–356, 1984.

14 P. Heggernes, D. Lokshtanov, J. Nederlof, C. Paul, and J. A. Telle. Generalized graph clustering: recognizing (p, q)-cluster graphs. In *Proceedings of WG 2010*, pages 171–183, 2010.

15 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

16 L. Ibarra. The clique-separator graph for chordal graphs. *Discrete Applied Mathematics*, 157:1737–1749, 2009.

17 M. O. Jackson. *Social and economic networks*. Princeton University press, vol. 3, 2008.

18 R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

19 D. J. Kleitman and R. V. Vohra. Computing the bandwidth of interval graphs. *SIAM J. Disc. Math.*, 3:373–375, 1990.

20 L. C. Lau. Bipartite roots of graphs. *ACM Transactions on Algorithms*, 2:178–208, 2006.

21 L. C. Lau and D. G. Corneil. Recognizing powers of proper interval, split, and chordal graphs. *SIAM J. Disc. Math.*, 18:83–102, 2004.

22 V. B. Le. Gallai graphs and anti-gallai graphs. *Discrete Mathematics*, 159:179–189, 1996.

23 V. B. Le, A. Oversberg, and O. Schaudt. Polynomial time recognition of squares of ptolemaic graphs and 3-sun-free split graphs. *Theoretical Computer Science*, 602:39–49, 2015.

24 P. J. Looges and S. Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25:15–25, 1993.

25 M. Milanič and O. Schaudt. Computing square roots of trivially perfect and threshold graphs. *Discrete Applied Mathematics*, 161:1538–1545, 2013.

26 J. L. Pfaltz. Chordless cycles in networks. In *Proceedings of ICDE Workshops 2013*, pages 223–228, 2013.

27 F. S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory, Academic Press, New York*, pages 139–146, 1969.

28 S. Sintos and P. Tsaparas. Using strong triadic closure to characterize ties in social networks. In *Proceedings of KDD 2014*, pages 1466–1475, 2014.

29 J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of WWW 2013*, pages 1307–1318, 2013.

Non-Crossing Geometric Steiner Arborescences

Irina Kostitsyna[*1], **Bettina Speckmann**[†2], **and Kevin Verbeek**[‡3]

1 **TU Eindhoven, Eindhoven, The Netherlands**
`i.kostitsyna@tue.nl`
2 **TU Eindhoven, Eindhoven, The Netherlands**
`b.speckmann@tue.nl`
3 **TU Eindhoven, Eindhoven, The Netherlands**
`k.a.b.verbeek@tue.nl`

Abstract

Motivated by the question of simultaneous embedding of several flow maps, we consider the problem of drawing multiple geometric Steiner arborescences with no crossings in the rectilinear and in the angle-restricted setting. When terminal-to-root paths are allowed to turn freely, we show that two rectilinear Steiner arborescences have a non-crossing drawing if neither tree necessarily completely disconnects the other tree and if the roots of both trees are "free". If the roots are not free, then we can reduce the decision problem to 2SAT. If terminal-to-root paths are allowed to turn only at Steiner points, then it is NP-hard to decide whether multiple rectilinear Steiner arborescences have a non-crossing drawing. The setting of angle-restricted Steiner arborescences is more subtle than the rectilinear case. Our NP-hardness result extends, but testing whether there exists a non-crossing drawing if the roots of both trees are free requires additional conditions to be fulfilled.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Steiner arborescences, non-crossing drawing, rectilinear, angle-restricted

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.54

1 Introduction

Flow maps are used in cartography to visualize the movement of objects between different locations. Generally, multiple sources are connected to multiple destinations with curves of varying thickness indicating the amount of flow. A good layout of a flow map consists of "simple" aesthetically pleasing curves, and avoids unnecessary intersections. Specifically, in this paper we are interested in drawing flow maps with no crossings at all. That is, given a set of source points and a set of destination points, we are looking to connect the source points to their corresponding destinations without intersections. For the sake of readability, the curves of a flow map should roughly be oriented from the source to the destination (or vice versa). This poses restrictions on the curves, which in related work [5] has been formalized using the notion of *angle-restricted* paths: for every point p on the path, the angle γ between the tangent vector at p and the vector from p to the source can be at most a prescribed angle α (see Figure 1).

[*] I.K. was partially supported by the Netherlands Organisation for Scientific Research (NWO) under grant number 639.023.208. I.K. was also supported by F.R.S.-FNRS.
[†] B.S. was partially supported by the Netherlands Organisation for Scientific Research (NWO) under grant number 639.023.208.
[‡] K.V. was supported by the Netherlands Organisation for Scientific Research (NWO) under grant number 639.021.541.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 54; pp. 54:1–54:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

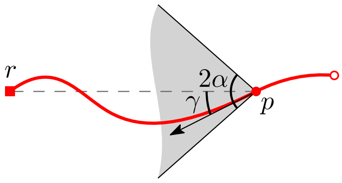


Figure 1 Angle-restricted path: γ is bounded by α.

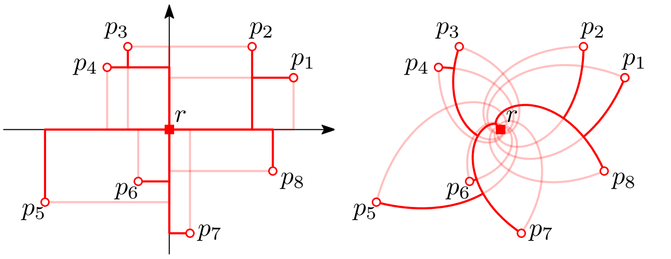


Figure 2 A rectilinear Steiner arborescence and a flux tree with eight terminals.

The notion of angle-restricted paths is closely related to that of so-called *generalized self-approaching* paths. A φ-self-approaching path, as defined by Aichholzer et al. in [2], is a path such that for any point p on it, the rest of the path lies inside some wedge with apex in p and with angle φ. Accordingly, any angle-restricted path with angular constraint α is also generalized self-approaching for angle $\varphi = 2\alpha$: for every point p on the path from some destination to its source r, the remainder of the path is contained in a wedge with angle 2α, with apex at p, and with r lying on its bisector. On the other hand, any α-self-approaching path is also angle-restricted with constraint α.

In a way, angle-restricted paths behave similarly to (xy-monotone) rectilinear paths. In this case, for every point p on the path, the subpath between p and the source r is bounded by an axis-aligned 90° wedge with the apex at p that contains r. We therefore study the following problems in this paper. Given a set of source points and a set of corresponding destination points, is it possible to connect all sources to their respective destinations using only angle-restricted paths (or xy-monotone paths in the rectilinear case) such that there are no intersections? The rectilinear case offers a somewhat simpler setting that is more amenable to analysis and allows us to clearly illustrate our main techniques.

In practice, a flow map will often have a small number of source points connected to multiple destination points to show the comparison of a certain commodity flow between several geographic locations, or to compare several types of commodities. Thus, we can group the flows in a flow map by a common source point to represent the out-flow of a given commodity from a specific location. Our problem is equivalent to drawing non-intersecting flow trees with angle-restricted (or xy-monotone for rectilinear) leaf-to-root paths. This approach has an important advantage of considering relevant flows together, and thus allowing for the possibility of merging similar flows which are going to the same source. A resulting merge point will then be a Steiner point, and the flow tree will be a Steiner *arborescence* (a tree with directed edges in the direction from the root). Buchin et al. [5] introduced angle-restricted Steiner arborescences, or *flux trees*, as a new variant of drawing flow trees. They study the problem of drawing a flux tree of minimal total length, and, among other results, show that the branches of an optimal flux tree consist of arcs of logarithmic spirals. Figure 2 shows an example of a rectilinear Steiner arborescence and a flux tree.

For two or more sets of input points, non-crossing Steiner arborescences need not even exist. Nonetheless, they are very relevant in practice. A single flux tree can show information about only one source, but ideally multiple sources should be shown simultaneously, in such a way that the corresponding flux trees have few or no crossings. To the best of our knowledge, these problems have not yet been studied. In this paper we are therefore studying the decision question of whether there exists a simultaneous non-crossing drawing of multiple geometric Steiner arborescences. Specifically, given a set of k roots (sources) $r_1, \ldots, r_k \in \mathbb{R}^2$, and k sets of terminals (destinations) $T_1, \ldots, T_k \subset \mathbb{R}^2$, do there exist k non-crossing Steiner arborescences which connect each set of terminals T_i to its root r_i, such that the leaf-to-root paths are angle-restricted (or xy-monotone for rectilinear)?

When talking about flows in a flow map, we assume the (standard in the literature) root-to-terminal orientation of the flows[1]. However, when drawing a flow tree, we start from a terminal and move towards the root. Thus, the direction of the paths in the construction of the trees is opposite to the flows in the flow map. When needed, we will explicitly state which orientation is considered to avoid any ambiguity.

Related work. The Euclidean Steiner tree problem and its variants have been studied extensively. Although most of these problems are NP-hard, many efficient approximation algorithms are known [3, 10]. However, if we want to compute multiple Steiner trees for multiple point sets, such that the Steiner trees have no or few crossings, then there are very few results. Aichholzer *et al.* [1] give an algorithm that, given two sets of n points in the plane, computes in $O(n \log n)$ time two spanning trees (not Steiner trees) such that the diameters of the trees and the number of intersections between the trees are small. Similar (weaker) results have also been obtained for drawing more than two plane spanning trees with few crossings [7, 9]. Recently, Bereg *et al.* [4] presented approximation algorithms for computing k disjoint Steiner trees for k point sets, with approximation ratios $O(\sqrt{n} \log k)$ and $k + \varepsilon$ for general k, $(5/3 + \varepsilon)$ for $k = 3$, and a PTAS for $k = 2$. Other relevant related work considers obtaining particular subgraphs of given geometric or topological graphs with few or no crossings [6, 8, 11, 12]. These problems are often NP-hard, except for certain special cases [12], but they differ from general Steiner tree problems, as the selected edges must be part of the input graph.

Preliminaries. In this paper, we focus mostly on the case of drawing two flow trees, i.e., when $k = 2$. When considering only two trees, we refer to the first tree as the *red* tree, with root r_1 and terminals $T_1 = \{p_1, \ldots, p_n\}$, and the second tree as the *blue* tree with root r_2 and terminals $T_2 = \{q_1, \ldots, q_m\}$. When studying multiple rectilinear Steiner arborescences, we generally allow the use of different axes for different trees. This way, the rectilinear problem is more similar to the more involved angle-restricted (flux trees) version of the problem.

It follows from the restriction on the paths that the path between a terminal and its root must completely lie in a particular region. For rectilinear Steiner arborescences this is the axes-aligned rectangle spanned by the root and the terminal. For flux tree this region is bounded by two curves traced by points for which the angle between the tangent and the direction to the destination is exactly α. These curves are in fact logarithmic spirals, and hence the above region is called the *spiral region* [5] (see Figure 3). Here we refer to these regions as $\mathcal{R}$-*regions*, and denote the $\mathcal{R}$-region given by a root r and a terminal t by $\mathcal{R}(r, t)$.

▶ **Definition 1.** Two $\mathcal{R}$-regions $\mathcal{R}(r_1, p_i)$ and $\mathcal{R}(r_2, q_j)$ fully intersect if $r_1, p_i \notin \mathcal{R}(r_2, q_j)$, $r_2, q_j \notin \mathcal{R}(r_1, p_i)$, and segments r_1p_i and r_2q_j intersect.

It is easy to verify that two non-crossing Steiner arborescences do not exist if there are two $\mathcal{R}$-regions $\mathcal{R}(r_1, p_i)$ and $\mathcal{R}(r_2, q_j)$ that fully intersect for $p_i \in T_1$ and $q_j \in T_2$ (see Figure 3): any two paths routed within the respective $\mathcal{R}$-regions must intersect.

[1] The same flow map can as well be used to represent an in-flow of a product with the flows oriented from terminals to roots.

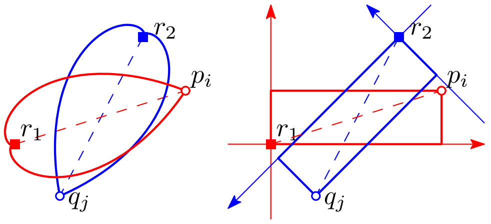


Figure 3 Two $\mathcal{R}$-regions fully intersect.

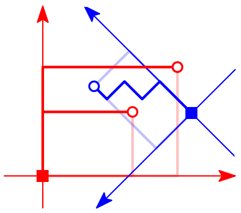


Figure 4 Limited turns: no drawing.

When drawing (the paths of) Steiner arborescences, we will consider two models. In the *limited turns* model, we restrict the segments of a terminal-to-root path of a flux tree to only follow one of the two logarithmic spirals implied by the location of the corresponding root, and prohibit the path from making a turn anywhere except for maybe at a merging point with another path. Similarly, in the *limited turns* model in the rectilinear case, we restrict terminal-to-root paths to be rectilinear, and prohibit the paths from making a turn except for at a merging point with another path (or at a corner of an $\mathcal{R}$-region). In the *free turns* model, we allow the paths to follow any angle-restricted (or xy-monotone) curves as long as there is no crossings. The limited turns model can be quite restrictive. Figure 4 shows an example where a non-crossing drawing of two rectilinear Steiner arborescences exists only if free turns are allowed.

Results. In Section 2.1 we show that two rectilinear Steiner arborescences, in the case when the roots are not contained inside any $\mathcal{R}$-regions, have a non-crossing drawing in the free turns model if and only if no two $\mathcal{R}$-regions fully intersect. In Section 2.2 we lift the constraint on the roots and show how to reduce the decision problem to 2SAT. For flux trees the problem is more involved: $\mathcal{R}$-regions of flux trees can have more complicated interactions. Contrary to rectilinear Steiner arborescences, it is not sufficient for flux trees to consider only full intersections of $\mathcal{R}$-regions if the roots are not contained in $\mathcal{R}$-regions. Nonetheless we can extend our arguments for rectilinear Steiner arborescences to show that, in the case when the roots are not contained in $\mathcal{R}$-regions, we can decide in polynomial time if two flux trees have a non-crossing drawing. Due to space constraints we provide only a sketch of this extension in Section 3; refer to the full version of this paper for details. In the limited turns model we can show that it is NP-hard to decide whether an arbitrary number of rectilinear Steiner arborescences or flux trees have a non-crossing drawing. This result, as well as all the omitted proofs, can be found in the full version of this paper.

2 Two rectilinear Steiner arborescences

In this section we show how to decide if a non-crossing drawing of two rectilinear Steiner arborescences in the free turns model exists and how to construct such a drawing if the answer is positive. We consider the general case, when the axes of the arborescences are not aligned. The free turn model implies that, in principle, the paths of the trees can approximate any xy-monotone curve. We show that we can restrict the directions of the paths to the 8 directions implied by the axes of the two rectilinear Steiner arborescences.

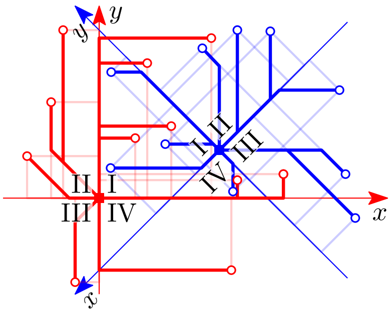


Figure 5 Non-crossing drawings of two Steiner arborescences.

2.1 Roots not contained in $\mathcal{R}$-regions

Consider the four quadrants of the coordinate system of the red arborescence ordered counterclockwise, and the four quadrants of the blue arborescence ordered clockwise. Let the first quadrants be the ones containing the other root (see Figure 5). In the arrangement of the four coordinate axes there are eleven faces, to which we refer by the two corresponding quadrants. For simplicity of presentation, we assume that no terminal lies on an axis of the other color. Let $\mathcal{C}_b$ be a cone with angle range $[0, \frac{\pi}{2}]$ in the red coordinate system with the apex in the blue root, and let $\mathcal{C}_r$ be a cone with angle range $[0, \frac{\pi}{2}]$ in the blue coordinate system with the apex in the red root. If the roots are not contained in the $\mathcal{R}$-regions of the other tree then there are no red terminals in $\mathcal{C}_b$, and there are no blue terminals in $\mathcal{C}_r$.

Given a red terminal p, and some xy-monotone path π_p connecting p to r_1, define a *dead region* $\mathcal{D}_2(\pi_p)$ with respect to the blue root r_2 to be the union of all points q such that path π_p intersects region $\mathcal{R}(q, r_2)$ and disconnects q from r_2. Analogously, given a blue terminal q and some xy-monotone path π_q connecting q to r_2, define a *dead region* $\mathcal{D}_1(\pi_q)$ to be the union of all points p such that path π_q disconnects p from r_1 in region $\mathcal{R}(p, r_1)$.

Observe that π_p is on the boundary of $\mathcal{D}_2(\pi_p)$, and that the rest of the boundary consists of lines parallel to blue axes. For example, in Figure 6, $\mathcal{D}_2(\pi_p)$ is bounded on one side by a line that goes through r_1 that is parallel to the blue y-axis. On the other side $\mathcal{D}_2(\pi_p)$ is bounded by a line parallel to the blue y-axis that goes through p, as p is in the blue quadrant II. If p were, for example, in blue quadrant I, than the bounding line would be parallel to blue x-axis. Also note that there are terminals p such that $\mathcal{D}_2(\pi_p)$ only consists of the points of π_p.

▶ **Definition 2.** Given a red terminal p such that $r_2 \notin \mathcal{R}(p, r_1)$, define the *dead region* $\mathcal{D}_2(p)$ with respect to r_2 to be the intersection of dead regions $\mathcal{D}_2(\pi_p)$ for all possible paths π_p connecting p to r_1:

$$\mathcal{D}_2(p) = \bigcap_{\pi_p} \mathcal{D}_2(\pi_p) .$$

Define the *dead region* $\mathcal{D}_1(q)$ of a blue terminal q analogously.

▶ **Proposition 3.** *For a red terminal $p \notin \mathcal{R}(q, r_2)$ and a blue terminal $q \notin \mathcal{R}(p, r_1)$, the following three statements are equivalent: (a) $q \in \mathcal{D}_2(p)$, (b) $p \in \mathcal{D}_1(q)$, (c) $\mathcal{R}(p, r_1)$ and $\mathcal{R}(q, r_2)$ fully intersect.*

Note that there can exist terminals whose dead regions are empty. For example, if $p \in \mathrm{I} \cap \mathrm{I}$ then there is a path connecting p to r_1 that does not obstruct routing of any possible blue

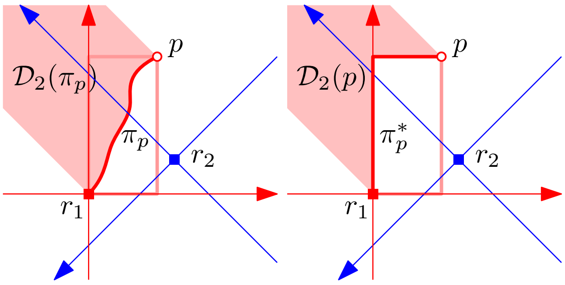
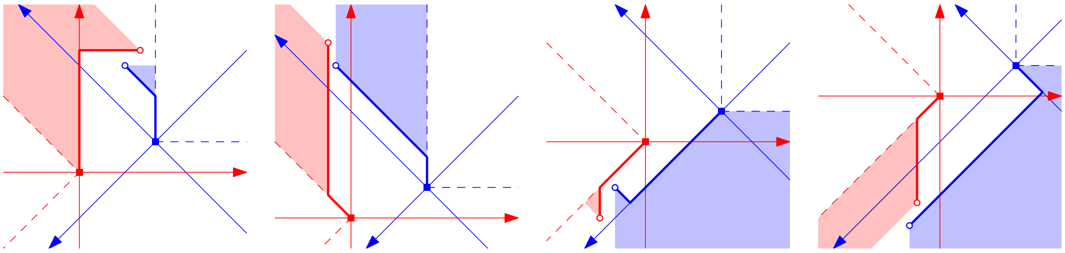


Figure 6 Dead regions: of a path π_p (left) and of a terminal p (right).



(a) I ∩ II. **(b)** II ∩ II. **(c)** III ∩ I. **(d)** III ∩ IV.

Figure 7 Red terminal p, blue terminal q, the corresponding dead regions $\mathcal{D}_2(p)$ (light red) and $\mathcal{D}_1(q)$ (light blue), and paths π_p^* and π_q^* connecting p to r_1 and q to r_2. Cones $\mathcal{C}_r$ and $\mathcal{C}_b$ are denoted with dashed red and blue lines respectively.

terminal. Consider the eight faces of the arrangement of the four axes except for faces I ∩ I, I ∩ IV, and IV ∩ I. For terminals p and q in them, $\mathcal{D}_2(p)$ and $\mathcal{D}_1(q)$ are not empty. Moreover, in these faces $p \in \mathcal{D}_2(p)$ and $q \in \mathcal{D}_1(q)$. Denote π_p^* to be the path that connects p to r_1 along the boundary of $\mathcal{D}_2(p)$ (refer to Figure 6 (right)). Similarly, denote π_q^* to be the path that connects q to r_2 along the boundary of $\mathcal{D}_1(q)$. We can show that:

▶ **Proposition 4.** *Paths π_p^* and π_q^* are xy-monotone in the red and blue coordinate systems, respectively.*

Therefore π_p^* and π_q^* are valid paths connecting p to r_1 and q to r_2. From Proposition 3 it follows that if a blue terminal $q \notin \mathcal{D}_2(p)$ then π_p^* does not intersect π_q^*. Figure 7 illustrates some of the possible placements of p and q such that their dead regions $\mathcal{D}_2(p)$ and $\mathcal{D}_1(q)$ are not empty.

Routing rules. Note that two cases, when there is a red terminal p in I ∩ II and when there is a blue terminal q in II ∩ I, are mutually exclusive. Otherwise there is no crossing free drawing of the arborescences. Table 1 gives a full list of all mutually exclusive cases. We will prove that, given two roots and two sets of terminals such that no two $\mathcal{R}$-regions of opposite colors fully intersect, there exists a non-crossing drawing of two Steiner arborescences. We can draw two non-crossing Steiner arborescences using the following routing rules:

Rule 1. If a red terminal $p \in (\mathrm{II} \cup \mathrm{III}) \backslash \mathcal{C}_r$ (refer to Figure 7 (b, c, d)), or p is in faces I ∩ II, I ∩ III, IV ∩ III, or IV ∩ IV (refer to Figure 7 (a)), then connect p to r_1 along π_p^*.

Rule 2. If a blue terminal $q \in (\mathrm{II} \cup \mathrm{III}) \backslash \mathcal{C}_b$ (refer to Figure 7 (a, b)), or q is in faces II ∩ I, III ∩ I, III ∩ IV, or IV ∩ IV (refer to Figure 7 (c, d) respectively), then connect q to r_2 along π_q^*.

Table 1 Mutually exclusive cases of locations of red and blue terminals.

red terminals		blue terminals
(a) in I ∩ II	vs.	(b) in II ∩ I,
(c) in I ∩ III	vs.	(d) in III ∩ I,
(e) in I ∩ IV	vs.	(f) in IV ∩ I,
(g) in III ∩ IV	vs.	(h) in IV ∩ III,
(i) in I ∩ III	vs.	(j) in IV ∩ IV,
(k) in IV ∩ IV	vs.	(l) in III ∩ I.

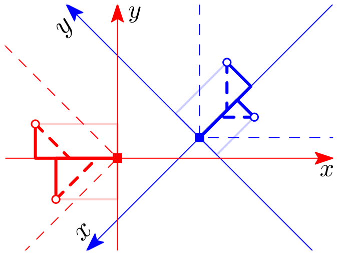
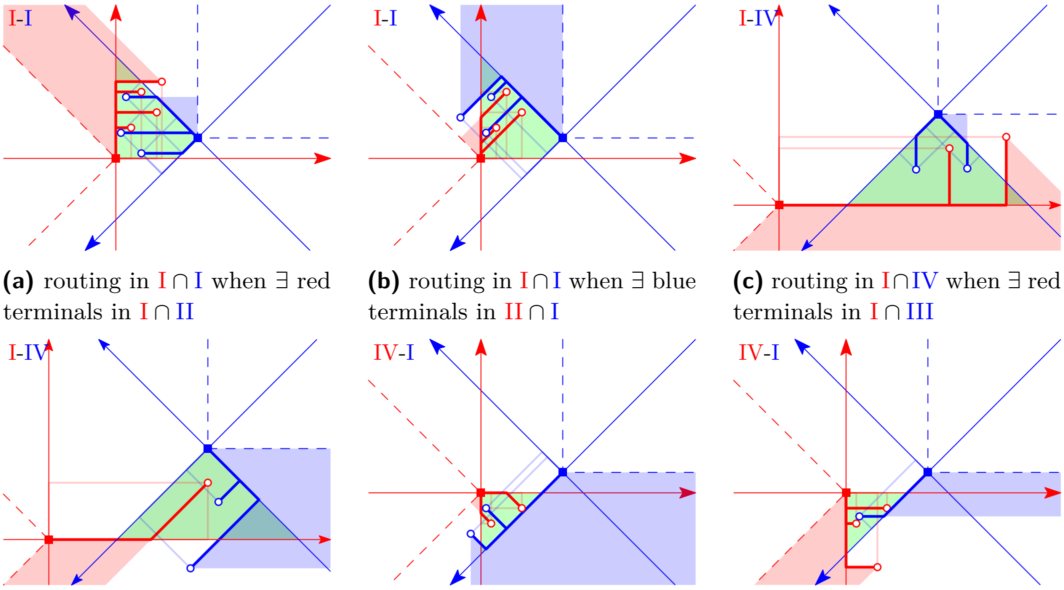


Figure 8 Rule 3.



(a) routing in I ∩ I when ∃ red terminals in I ∩ II

(b) routing in I ∩ I when ∃ blue terminals in II ∩ I

(c) routing in I ∩ IV when ∃ red terminals in I ∩ III

(d) routing in I ∩ IV when ∃ blue terminals in IV ∩ IV

(e) routing in IV ∩ I when ∃ blue terminals in III ∩ I

(f) routing in IV ∩ I when ∃ red terminals in IV ∩ IV

Figure 9 Routing rules for drawing rectilinear Steiner arborescences. Shaded regions denote dead regions.

Rule 3. Route the red terminals in cone $\mathcal{C}_r$ parallel to the red y-axis until reaching the x-axis, then along it. Route the blue terminals in $\mathcal{C}_b$ parallel to the blue y-axis until the x-axis, then along it. For aesthetics, we can add a shortcut in the direction of one of the axes of the opposite color (see Figure 8).

After applying Rules 1–3, all the terminals outside of I ∩ I, I ∩ IV, and IV ∩ I are connected to the roots. We use the following routing rules for the remaining terminals (see Figure 9).

Rule 4. Face I ∩ I: in case (a) (in Table 1), red edges are drawn parallel to the red x-axis until the red y-axis, then follow it, blue edges are drawn parallel to the red x-axis, until a blue axis, then follow it to the root; in case (b), blue edges are drawn parallel to the blue x-axis until the blue y-axis, then follow it, red edges are drawn parallel to the blue x-axis until a red axis, then follow it.

Rule 5. Face I ∩ IV: in case (i) (in Table 1), red edges are drawn parallel to the red y-axis then along the red x-axis, blue edges are drawn parallel to the red y-axis then along the blue y-axis; in case (j), red edges are drawn parallel to the blue x-axis then along the red x-axis, blue edges are drawn parallel to the blue x-axis then along the blue y-axis.

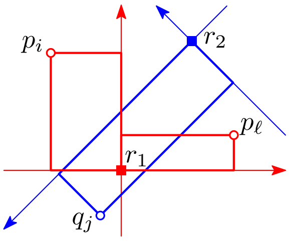


Figure 10 Definition 6.

Rule 6. Face IV ∩ I: in case (k) (in Table 1), red edges are drawn parallel to the red x-axis then along the red y-axis, blue edges are drawn parallel to the red x-axis then along the blue x-axis; in case (l), blue edges are drawn parallel to the blue y-axis then along the blue x-axis, red edges are drawn parallel to the blue y-axis then along a red axis.

Theorem 5 now follows from the definition of the dead regions and a case analysis over faces containing red and blue terminals.

▶ **Theorem 5.** *If the roots are not contained in $\mathcal{R}$-regions, then two rectilinear Steiner arborescences can be drawn with no crossings in the free turn model if and only if no two $\mathcal{R}$-regions fully intersect.*

2.2 Roots contained in $\mathcal{R}$-regions

We now relax the restriction that the roots cannot be contained in $\mathcal{R}$-regions. Hence, for any $\mathcal{R}$-region that contains the root of the other color, we need to make a choice of how to route the terminal-to-root path around the other root. This choice clearly can affect later decisions.

Before we proceed, we introduce some additional definitions. Points r and t split the boundary of $\mathcal{R}(t, r)$ into two pieces that we call the *left* and the *right* sides (with respect to moving from t to r).

▶ **Definition 6.** We say that $\mathcal{R}(p_i, r_1)$ cuts the left (right) side of $\mathcal{R}(q_j, r_2)$, if $r_1 \in \mathcal{R}(q_j, r_2)$, and both sides of $\mathcal{R}(p_i, r_1)$ intersect the left (right) side of $\mathcal{R}(q_j, r_2)$ (refer to Figure 10).

We can define a dead region of a terminal p for a fixed direction a p-to-r_1 path must take around r_2:

▶ **Definition 7.** A left (right) dead region $\mathcal{D}_2(p, \text{left})$ ($\mathcal{D}_2(p, \text{right})$) with respect to r_2, for a given red terminal p such that $r_2 \in \mathcal{R}(p, r_1)$, is the intersection of dead regions $\mathcal{D}_2(\pi_p)$ for all possible paths π_p connecting r_1 to p that pass between r_2 and the left (right) side of $\mathcal{R}(p, r_1)$:

$$\mathcal{D}_2(p, \text{left}) = \bigcap_{\text{left } \pi_p} \mathcal{D}_2(\pi_p)\,, \qquad \mathcal{D}_2(p, \text{right}) = \bigcap_{\text{right } \pi_p} \mathcal{D}_2(\pi_p)\,.$$

Analogously, define $\mathcal{D}_1(q, \text{left})$ and $\mathcal{D}_1(q, \text{right})$. Note that we can make a similar observation for left and right dead regions as for dead regions. Let blue root $r_2 \in \mathcal{R}(p, r_1)$. A blue terminal q lies in $\mathcal{D}_2(p, \text{left})$ ($\mathcal{D}_2(p, \text{right})$) if and only if $\mathcal{R}(q, r_2)$ cuts the left (right) side of $\mathcal{R}(p, r_1)$.

We reduce the problem of choosing the direction of the path with respect to the other root by reducing it to 2SAT. We assign a boolean variable to each $\mathcal{R}$-region containing the root of the other color, which takes its value according to the direction in which the terminal-to-root

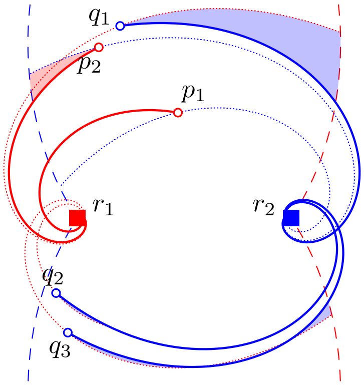


Figure 11 Dead regions for two flux trees.

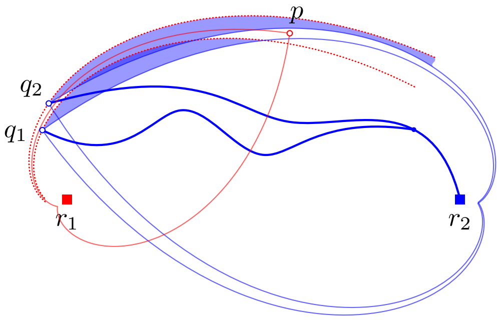


Figure 12 $\mathcal{R}(p, r_q)$ does not fully intersect $\mathcal{R}(q_1, r_2)$ nor $\mathcal{R}(q_2, r_2)$. Nevertheless, there is no angle-restricted path from p to r_1 that does not intersect paths from q_1 and q_2 to r_2. Areas highlighted in light-blue are the dead regions of q_1 and q_2.

path goes around the other root. Given a 2SAT formula solution, we can apply the routing rules and connect the terminals to their roots along the boundaries of the dead regions.

▶ **Theorem 8.** *We can decide in polynomial time whether two rectilinear Steiner arborescences can be drawn without crossings in the free turn model.*

3 Two flux trees

In this section we sketch how to draw two flux trees with no root containment in $\mathcal{R}$-regions in the model when free turns are allowed. The details can be found in the full version of this paper. Similarly to the rectilinear case, free turns imply that a terminal-to-root path can be any angle-restricted curve. Any angle-restricted curve can be approximated with a curve following only four types of logarithmic spirals: left-handed and right-handed, or simply left and right, spirals (left spirals spiral in clockwise direction when moving towards the root, right spirals spiral in counter-clockwise direction) with their origins in the red and blue roots. Thus we can restrict our drawing to these four types of spirals.

Similarly to the rectilinear case, we define the areas $\mathcal{C}_r$ and $\mathcal{C}_b$ which should be empty of blue and red terminals respectively (to fulfill the no-root-containment requirement). These areas are bounded by the spirals centered at one root and going through the other root.

Analogously to the rectilinear case, we can define a dead region of a path, and a dead region of a terminal point. Figure 11 shows an example of several terminals and their dead regions. The dead regions are bounded by two logarithmic spirals going through a terminal and centered at the two roots. Consider, for example, red terminal p_2 in Figure 11. Part of the blue spiral that goes through p_2 is hidden from root r_2 by the red spiral connecting p_2 to r_1. Therefore, for any terminal q above the red spiral, but below the blue spiral (area shaded light-red in the figure), $\mathcal{R}(q, r_2)$ will fully intersect $\mathcal{R}(p_2, r_1)$.

A red and a blue logarithmic spiral can intersect more than once inside the area $\mathbb{R}^2 \backslash (\mathcal{C}_r \cup \mathcal{C}_b)$. This fact can cause some dead regions to consist of several connected components (for example, blue terminal q_3 in Figure 11). Moreover, we no longer can consider the dead regions independently, as we did in the rectilinear setting. Consider the example in Figure 12. Point p does not belong to the dead region of q_1 nor of q_2 ($\mathcal{R}(p, r_1)$ does not fully intersect $\mathcal{R}(q_1, r_2)$ nor $\mathcal{R}(q_2, r_2)$). Nevertheless, no angle-restricted path connecting p to r_1 can avoid paths from q_1 and q_2 to r_2. Indeed, any angle-restricted path from p to r_1 will intersect

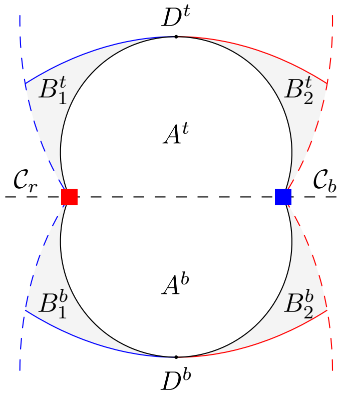


Figure 13 Regions A^t, A^b, B_1^t, B_2^t, B_1^b, B_2^b, D^t, and D^b.

either the dead region of q_1 or the dead region of q_2. Thus, when two or more dead regions intersect, they block some area outside of them that becomes forbidden for the terminals of the other color. We will call this area an *extended dead region.*

To formally define the extended dead region, we need to introduce some notation. Let $s_r^+(p)$ and $s_r^-(p)$ be respectively the spiral segments of the right and left logarithmic spirals, centered at r_1 and going through p, which are bounded by $\mathbb{R}^2 \backslash (\mathcal{C}_r \cup \mathcal{C}_b)$; and let $s_b^+(q)$ and $s_b^-(q)$ be respectively the spiral segments of the right and left logarithmic spirals, centered at r_2 and going through q, which are bounded by $\mathbb{R}^2 \backslash (\mathcal{C}_r \cup \mathcal{C}_b)$. Let $\mathbb{R}^2_+$ be the half-plane to the left of $r_1 r_2$, and $\mathbb{R}^2_-$ be the half-plane to the right of $r_1 r_2$.

▶ **Definition 9.** Given k blue terminals $Q = \{q_1, q_2, \ldots, q_k\}$ in $\mathbb{R}^2_+ \backslash (\mathcal{C}_r \cup \mathcal{C}_b)$, such that the component of the dead region $\mathcal{D}_1(q_i)$ containing q_i intersects the left side of $\mathcal{R}(q_{i+1}, r_2)$ for all $1 \leq i < k$, and Q is maximal, define the extended dead region $\mathcal{D}_1(Q)$ with respect to the red root r_1 to be:

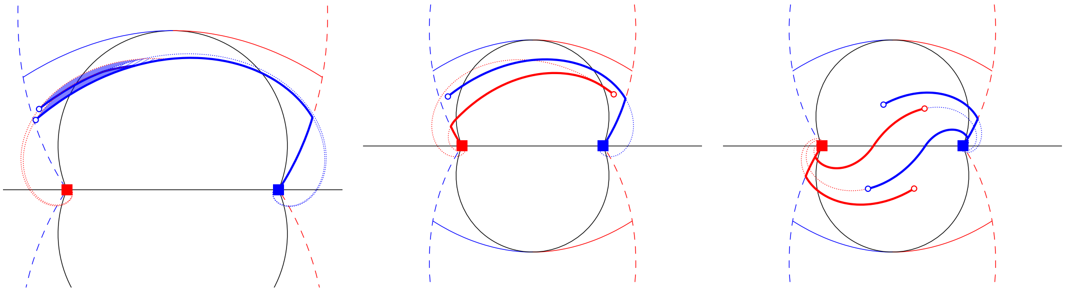
$$\mathcal{D}_1(Q) = \bigcup_{1 \leq i < k} F_i\,,$$

where F_i is the area enclosed between the left sides of $\mathcal{R}(q_i, r_2)$ and $\mathcal{R}(q_{i+1}, r_2)$, and the two red spiral segments $s_r^+(q_i)$ and $s_r^+(q_k)$.

Similarly define the extended dead region $\mathcal{D}_1(Q)$ of a set Q of blue terminals lying in the bottom half-plane, and the extended dead regions $\mathcal{D}_2(P)$ of a set P of red terminals for the top and the bottom half-planes.

We will show that there exists a non-crossing drawing of two flux trees, given that the roots are not contained in any $\mathcal{R}$-region, if and only if no terminal lies inside a dead region or an extended dead region of the other color.

Routing rules. We can partition $\mathbb{R}^2$ into several regions such that we can specify the routing rules for terminals within each region separately (see Figure 13): A^t and A^b are bounded by $r_1 r_2$ and two circular arcs with an angle subtended by the chord $r_1 r_2$ equal to $\pi - 2\alpha$; B_1^t, B_2^t, B_1^b, and B_2^b are bounded by the arcs of A^t and A^b, the boundaries of the regions $\mathcal{C}_r$ and $\mathcal{C}_b$, and by a spiral going through the topmost or the bottommost point of the arcs; $D^t = \mathbb{R}^2_+ \backslash (\mathcal{C}_r \cup \mathcal{C}_b \cup A^t \cup B_1^t \cup B_2^t)$, and $D^b = \mathbb{R}^2_- \backslash (\mathcal{C}_r \cup \mathcal{C}_b \cup A^b \cup B_1^b \cup B_2^b)$. An important observation is that in A^t and A^b red paths can be routed along blue spirals and vice versa.



(a) Rule 2. **(b)** Rule 4. **(c)** Rules 4 and 5.

Figure 14 Routing rules for drawing flux trees.

We can now introduce the following routing rules to draw two non-crossing flux trees if there were no terminals of the other color in the dead regions. Refer to Figure 14 for the illustrations of the rules. Note that Rules 4 and 5 can introduce some intersections, that we will uncross afterwards.

Rule 1. Route red and blue terminals in D^t in D^b along their respective spiral segments s_r^+, s_b^-, s_r^-, or s_b^+ until reaching the boundary of $\mathcal{C}_r$ or $\mathcal{C}_b$;

Rule 2. For all blue extended dead regions, route the corresponding sets of blue terminals $Q = \{q_1, \ldots, q_k\}$ in $\mathbb{R}^2_+$ in the following way: route q_1 along a blue spiral segment $s_b^-(q_1)$ until it reaches $\mathcal{C}_b$; for every $1 < i \leq k$, route q_i along the boundary of its dead region and then along the boundary of the extended dead region until reaching $s_b^-(q_1)$ (merging with the path from q_1 to r_2) or the boundary of the cone $\mathcal{C}_b$;

Rule 3. For all red extended dead regions, route the corresponding sets of red terminals $P = \{p_1, \ldots, p_k\}$ in $\mathbb{R}^2_-$ in the following way: route p_1 along a red spiral segment $s_r^-(p_1)$ until it reaches $\mathcal{C}_r$; for every $1 < i \leq k$, route p_i along the boundary of its dead region, then along the boundary of the extended dead region until reaching $s_r^-(p_1)$ (merging with the path from p_1 to r_1) or the boundary of the cone $\mathcal{C}_r$;

Rule 4. The rest of the blue terminals in $\mathbb{R}^2_+$ route along their left blue spiral segments until reaching the boundary of cone $\mathcal{C}_b$; the rest of the red terminals in $\mathbb{R}^2_+$ route along a right red spiral within region B_2^t, along a left blue spiral within A^t, and along right red spiral within B_1^t until reaching the cone $\mathcal{C}_r$;

Rule 5. The rest of the red terminals in $\mathbb{R}^2_-$ route along their left red spiral segments until reaching the boundary of cone $\mathcal{C}_r$; the rest of the blue terminals in $\mathbb{R}^2_-$ route along a right blue spiral within B_1^b, along a left red spiral within A^b, and along right blue spiral within B_2^b until reaching the cone $\mathcal{C}_b$;

Rule 6. Finally, route all the blue paths along the boundary of $\mathcal{C}_b$ to r_2 and all the red paths along the boundary of $\mathcal{C}_r$ to r_1. Red terminals in $\mathcal{C}_r$ and blue terminals in $\mathcal{C}_b$ can be routed arbitrarily (joining when necessary) within those cones towards their respective roots.

As mentioned, after applying Rules 4 and 5, some intersections are possible. Specifically, red and blue paths can intersect within regions B_1^t, B_2^t, B_1^b, or B_2^b. Red and blue paths do not intersect within A^t or A^b, as they follow non-intersecting spirals; and red and blue paths do not intersect within D^t or D^b, otherwise the corresponding $\mathcal{R}$-regions would fully intersect.

Consider a red terminal p and a blue terminal q in $\mathbb{R}^2_+$ such that their paths, constructed by the presented routing rules, intersect. These paths can intersect only once, due to the

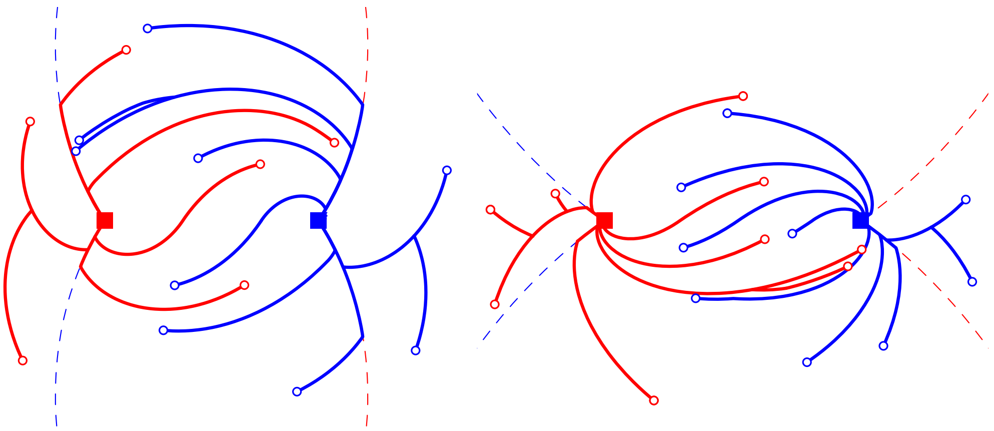


Figure 15 Two non-crossing drawings of flux trees for $\alpha = 60°$ (left) and $\alpha = 30°$ (right).

difference of curvatures of spirals in B_1^t and B_2^t, and because within A^t these paths follow non-intersecting spirals. There can be two cases: (a) the intersection point is in B_1^t; and (b) the intersection point is in B_2^t.

In the first case, the blue terminal q is in B_1^t, and the red path crosses its dead region $\mathcal{D}_1(q)$. Then reroute the red path along the boundary of the dead region $\mathcal{D}_1(q)$, when it first encounters it. If q is a part of a set of blue terminals that define an extended dead region, then reroute the red path along the boundary of the extended dead region when it first encounters it. The new red path will not intersect any other blue paths, otherwise these paths would be a part of the set defining the extended dead region.

In the second case, when intersection point is inside B_2^t, the red terminal p is in B_2^t. Let f be the intersection point of the red path with the boundary between A^t and B_2^t. Consider the left side of $\mathcal{R}(p, r_q)$. It intersects the blue path exactly two times, otherwise the $\mathcal{R}$-regions of p and q would fully intersect. Let g be the intersection point of the left side of $\mathcal{R}(p, r_q)$ and the blue path inside A^t. Consider all the blue paths that intersect the red spiral segment $s_r^+(p)$ between points f and g. Reroute all these paths along the red spiral segment $s_r^+(p)$ when they first encounter it, until they reach point g, then route the merged path along a new blue spiral segment $s_b^-(g)$ within B_2^t. Let the red path continue following the red spiral segment $s_r^+(p)$ when it enters A^t until it reaches point f, then let the red path follow the blue spiral segment $s_b^-(f)$ “parallel” to the rest of the paths in A^t. Note, that if there was a part of the red path in $\mathbb{R}^2_-$, the new path may completely lie in $\mathbb{R}^2_+$. This procedure essentially brings the part of the blue path(s) that was above the red path under it.

The symmetrical cases in the bottom half-plane can be dealt with similarly. And if the terminals lie in the different half-planes, their paths can intersect once or twice. However, the method for uncrossing such paths completely mirrors the cases for when p and q lie in the same half-plane. Figure 15 shows the final result of the procedure. In the full version of this paper we prove the following theorem.

▶ **Theorem 10.** *A drawing of two non-crossing flux trees with no root containment in $\mathcal{R}$-regions exists if and only if no terminal lies in a dead region or an extended dead region of the other color. If it exists, such a drawing can be constructed in polynomial time.*

4 Conclusion and Future Work

In this paper we study the problem of drawing a flow map with non-crossing curves that have to be oriented approximately towards the source. We have shown that we can efficiently decide if two rectilinear Steiner arborescences can be drawn without crossings, if we require the paths to simply be xy-monotone with no other restrictions. Similarly, we show how to draw two non-intersecting flux trees in the case when their roots are not contained in the other tree's $\mathcal{R}$-regions.

With an extra restriction on the paths that prohibits free turns, the problem becomes NP-hard for k Steiner arborescences, where k is part of the input. We conjecture that this problem is also NP-hard for $k = 2$. Whether the problem is NP-hard for more than two Steiner arborescences in the free turn model is left as an open problem.

References

1 Oswin Aichholzer, Franz Aurenhammer, Thomas Hackl, and Clemens Huemer. Connecting colored point sets. *Discrete Applied Mathematics*, 155(3):271–278, 2007. doi:10.1016/j.dam.2006.06.010.

2 Oswin Aichholzer, Franz Aurenhammer, Christian Icking, Rolf Klein, Elmar Langetepe, and Günter Rote. Generalized self-approaching curves. *Discrete Applied Mathematics*, 109(1-2):3–24, apr 2001. doi:10.1016/S0166-218X(00)00233-X.

3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

4 Sergey Bereg, Krzysztof Fleszar, Philipp Kindermann, Sergey Pupyrev, Joachim Spoerhase, and Alexander Wolff. Colored non-crossing Euclidean Steiner forest. In *Proc. 26th International Symposium on Algorithms and Computation (ISAAC)*, pages 429–441, 2015. URL: http://arxiv.org/abs/1509.05681.

5 Kevin Buchin, Bettina Speckmann, and Kevin Verbeek. Angle-restricted Steiner arborescences for flow map layout. *Algorithmica*, 72(2):656–685, 2015. doi:10.1007/s00453-013-9867-z.

6 Klaus Jansen and Gerhard J. Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT Numerical Mathematics*, 33(4):580–595, 1993. doi:10.1007/BF01990536.

7 M. Kano, C. Merino, and J. Urrutia. On plane spanning trees and cycles of multicolored point sets with few intersections. *Information Processing Letters*, 93(6):301–306, 2005.

8 Christian Knauer, Étienne Schramm, Andreas Spillner, and Alexander Wolff. Configurations with few crossings in topological graphs. *Computational Geometry*, 37(2):104–114, 2007. doi:10.1016/j.comgeo.2006.06.001.

9 J. Leaños, C. Merino, G. Salazar, and J. Urrutia. Spanning trees of multicoloured point sets with few intersections. In *Proc. Indonesia-Japan Joint Conference on Combinatorial Geometry and Graph Theory (IJCCGGT)*, LNCS 3330, pages 113–122, 2005.

10 Bing Lu and Lu Ruan. Polynomial time approximation scheme for the rectilinear Steiner arborescence problem. *Journal of Combinatorial Optimization*, 4(3):357–363, 2000. doi:10.1023/A:1009826311973.

11 Julia Schüler and Andreas Spillner. Crossing-free spanning trees in visibility graphs of points between monotone polygonal obstacles. In *Proc. 9th International Computer Science Symposium in Russia (CSR)*, LNCS 8476, pages 337–350, 2014.

12 Shin-ichi Tokunaga. Intersection number of two connected geometric graphs. *Information Processing Letters*, 59(6):331–333, 1996. doi:10.1016/0020-0190(96)00124-X.

Precedence-Constrained Min Sum Set Cover*

Jessica McClintock[1], Julián Mestre[2], and Anthony Wirth[†3]

1 School of Computing and Information Systems, The University of Melbourne, Parkville, Australia
jessica.mcclintock@unimelb.edu.au

2 School of Information Technologies, The University of Sydney, Darlington, Australia
julian.mestre@sydney.edu.au

3 School of Computing and Information Systems, The University of Melbourne, Parkville, Australia
awirth@unimelb.edu.au

Abstract

We introduce a version of the Min Sum Set Cover (MSSC) problem in which there are "AND" precedence constraints on the m sets. In the Precedence-Constrained Min Sum Set Cover (PCMSSC) problem, when interpreted as directed edges, the constraints induce an acyclic directed graph. PCMSSC models the aim of scheduling software tests to prioritize the rate of fault detection subject to dependencies between tests.

Our greedy scheme for PCMSSC is similar to the approaches of Feige, Lovász, and, Tetali for MSSC, and Chekuri and Motwani for precedence-constrained scheduling to minimize weighted completion time. With a factor-4 increase in approximation ratio, we reduce PCMSSC to the problem of finding a maximum-density precedence-closed sub-family of sets, where density is the ratio of sub-family union size to cardinality. We provide a greedy factor-$\sqrt{m}$ algorithm for maximizing density; on forests of in-trees, we show this algorithm finds an optimal solution. Harnessing an alternative greedy argument of Chekuri and Kumar for Maximum Coverage with Group Budget Constraints, on forests of out-trees, we design an algorithm with approximation ratio equal to maximum tree height.

Finally, with a reduction from the Planted Dense Subgraph detection problem, we show that its conjectured hardness implies there is no polynomial-time algorithm for PCMSSC with approximation factor in $O(m^{1/12-\varepsilon})$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases planted dense subgraph, min sum set cover, precedence constrained

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.55

1 Introduction

In this paper, we introduce the Precedence-Constrained Min Sum Set Cover problem, which has connections to Min Sum Set Cover, Densest Subgraph, Precedence-Constrained Scheduling to Minimize Total Weighted Completion Time, Scheduling with AND/OR Precedence Constraints, and several other problems.

* This work was partially supported by the Australian Research Council.

† Corresponding author. Anthony Wirth was supported by an Australian Research Council Future Fellowship.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 55; pp. 55:1–55:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Problem definition. Just like Set Cover, or indeed Min Sum Set Cover (aka MSSC), the input to Precedence-Constrained Min Sum Set Cover (aka PCMSSC) is a family of sets, $\mathcal{F}$, whose union is the *universe*, U. In addition, there is a binary precedence relation on $\mathcal{F}$, represented by $\prec$. Let $G(\mathcal{F}, \prec)$ stand for the directed graph that $\prec$ induces on $\mathcal{F}$. Throughout this presentation, we assume G is *acyclic*. The output is a permutation, π, of the family of sets, $\mathcal{F}$, that obeys the precedence relation $\prec$. That is, if $A \prec B$, then A must precede B in the permutation: $\pi^{-1}(A) < \pi^{-1}(B)$. The objective value, to be minimized, is the sum over every item in U of its *first* covering time in the permutation. That is, for each item u, let $\tau(u) = \min_{A \in \mathcal{F}}\{\pi^{-1}(A) : u \in A\}$ be the index of the earliest set in π that includes item u; the objective is $\sum_{u \in U} \tau(u)$. For convenience, let n be the number of items in the universe $|U|$, and let m be the cardinality of the family of sets $|\mathcal{F}|$, and index the permutation π from 1 to m.

1.1 Application

In the software testing context, we would like to schedule test sequences to prioritize the rate of fault detection. However, there may be inherent dependencies between the tests – some test cases need to be scheduled before others – complicating the process of ordering the test suite [18]. Though they perform well, existing algorithms for test case prioritization subject to dependencies, are heuristic in their effectiveness [18, 17]. The PCMSSC problem crystallizes the aims and constraints of this software test prioritization problem in a way that admits analysis and approximation, yet is realistic. PCMSSC is a small extension of one existing combinatorial optimization question, MSSC, and a refinement of another, Scheduling with AND/OR Precedence Constraints (aka SAOPC).

1.2 Theoretical context

The original MSSC problem has the same objective as PCMSSC, to minimize $\sum_{u \in U} \tau(u)$, but it permits every ordering π of $\mathcal{F}$. Feige, Lovász and Tetali's greedy algorithm for MSSC [10] starts from an empty ordering π and is simply: *While $|\pi| < m$, append to π a set maximizing the number of (yet) uncovered items.* Via a clever pricing and histogram argument, they show that this is a 4-approximation; they also show that this is the best possible unless P equals NP.

Chekuri and Motwani attack the Precedence-Constrained Scheduling to Minimize Total Weighted Completion Time (aka PCSTW) problem, which has the same precedence structure as PCMSSC. Here, however, the total weighted completion times objective is additive, whereas set coverage is (only) monotone submodular. Chekuri and Motwani's factor-2 algorithm for PCSTW repeatedly (and optimally) solves the subproblem Minimum-Rank Precedence-Closed Subgraph (aka MRPCS) [7]. The *rank* of a family of jobs is the ratio of its total processing time to its total weight.

Key sub-problem definition. To produce an approximation algorithm for PCMSSC, we combine the ideas of Feige et al. and Chekuri and Motwani. We study a problem we call Max-Density Precedence-Closed Subfamily (aka MDPCS): in some sense, *density* is the reciprocal of rank. We let the *coverage* of sub-family, $\mathcal{A}$ of $\mathcal{F}$, be the union of the sets in the sub-family: $\text{cov}(\mathcal{A}) \equiv \cup_{A \in \mathcal{A}} A$. Often, we consider the coverage of a sub-family on some subset X of the universe: $\text{cov}(\mathcal{A}, X) = \text{cov}(\mathcal{A}) \cap X$. The density, Δ, of a non-empty sub-family on subset X is the ratio of the size of its coverage to its cardinality: $\Delta(\mathcal{A}, X) \equiv |\text{cov}(\mathcal{A}, X)|/|\mathcal{A}|$. (When it is obvious, we omit the second argument of Δ.)

Algorithm 1 Algorithm PCMSSC-Greedy.

```
function PCMSSC-Greedy($\mathcal{F}, \prec$)
    $\pi \leftarrow$ an “empty permutation”
    $\mathcal{G} \leftarrow \mathcal{F}$; $R \leftarrow U$
    while $R \neq \varnothing$ do
        $\mathcal{A} \leftarrow D(\mathcal{G}, \prec, R)$
        Append to $\pi$ some permutation of $\mathcal{A}$ consistent with $\prec$
        $\mathcal{G} \leftarrow \mathcal{G} \setminus \mathcal{A}$; $R \leftarrow R \setminus \mathrm{cov}(\mathcal{A})$
    Return $\pi$
```

For convenience' sake, $\Delta(\varnothing, X)$ is defined to be negative. The MDPCS problem seeks a sub-family $\mathcal{A}$ of some input family of sets $\mathcal{G}$ that maximizes density on a remaining set of items to be covered, R, with $R \subseteq \mathrm{cov}(\mathcal{G})$, and is *precedence closed*. That is, the aim is to maximize $\Delta(\mathcal{A}, R)$, with the requirement that if $A \prec B$ and $B \in \mathcal{A}$, then $A \in \mathcal{A}$. Were sets in $\mathcal{G}$ pairwise disjoint, we could adopt Chekuri and Motwani's approach for MRPCS, but in general, it is highly unlikely that such a polynomial-time optimal algorithm exists for MDPCS. Indeed, our hardness-of-approximation result for MDPCS arises from a connection to Densest k-Subgraph, whereas Chekuri and Motwani's max-flow algorithm solves MRPCS in polynomial time, similar to the approach for Densest Subgraph [12].

1.3 Our results

We first show that an approximately good solution to MDPCS provides, within factor 4, an approximately good solution to PCMSSC.

We describe a greedy algorithm, MDPCS-Greedy, for MDPCS that obtains a $\sqrt{m}$ approximation, and show that (up to a factor 2) this analysis is tight. We extend MDPCS-Greedy to be iteratively greedy, and again show that $O(\sqrt{m})$ is the best approximation we can obtain. If the precedence relation, $\prec$, induces a forest of in-trees, we show that MDPCS-Greedy in fact solves MDPCS optimally in polynomial time. If the precedence relation, $\prec$, induces a forest of out-trees, we introduce a polynomial-time approximation with factor equal to the largest tree *height*.

Consistent with the large approximation factors found in our algorithms, we show there is no approximation algorithm for PCMSSC with factor in $O(m^{1/12-\varepsilon})$. This result assumes the Planted Dense Subgraph Conjecture (aka PDSC), which states that it is hard to find inside an Erdős-Rényi graph a planted dense E-R component. Recently, hardness of approximation of the Target Set Selection problem was shown via a reduction from Planted Densest Subgraph and its conjectured hardness [5].

2 Reduction to Max-Density Precedence-Closed Subfamily

In this section, we show that PCMSSC *reduces* to MDPCS. Suppose we have an algorithm, D, that returns a factor-α approximation solution to MDPCS. Consider the greedy scheme in Algorithm 1 for PCMSSC, which we call PCMSSC-Greedy.

Since D is a polynomial-time algorithm (and X only gets smaller), since the while loop runs at most n times, and since topological sorting takes polynomial time, PCMSSC-Greedy runs in polynomial time. We now prove that this scheme is in fact an approximation algorithm.

Although factor α might be a function of both m and n, we assume α is monotonically non-decreasing in both, so we can safely let α stand for $\alpha(m, n)$ in the following.

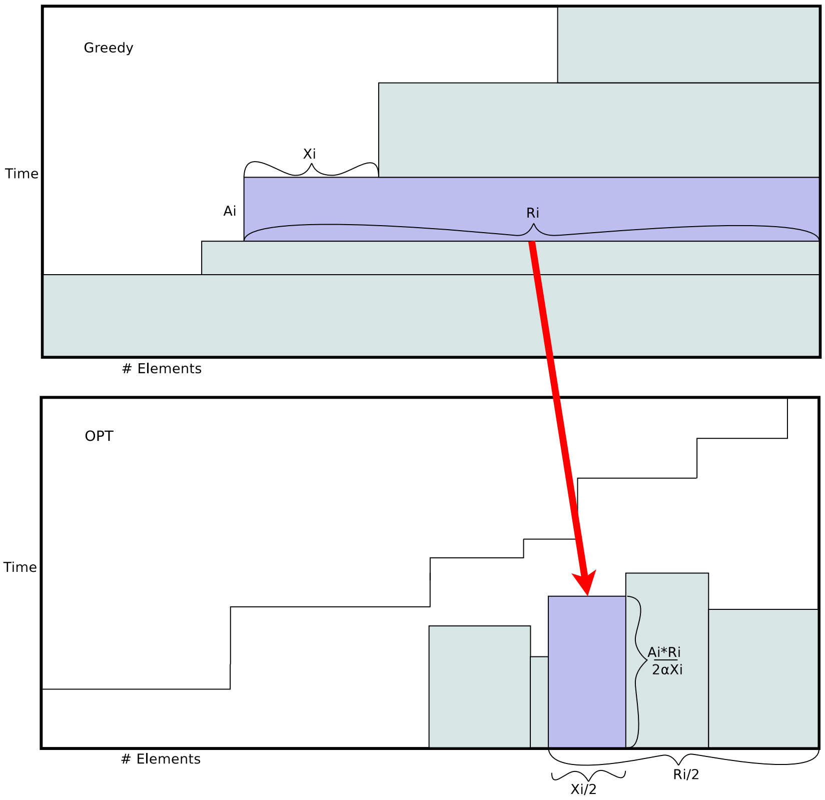


Figure 1 Mapping from upper-bound plot $\mathcal{B}$ to one with area a factor $4 \cdot \alpha$ smaller.

▶ **Lemma 1.** PCMSSC-Greedy *is a* $4 \cdot \alpha$ *approximation algorithm to* PCMSSC.

Proof. Let $\mathcal{A}_i$ be the sub-family returned by D in iteration i of PCMSSC-Greedy, and let $m_i = \sum_{j=1}^{i} |\mathcal{A}_j|$. Also, let R_i be the subset of U not yet covered after $i-1$ iterations, $\{u : \tau(u) > m_{i-1}\}$, and let $X_i \subseteq R_i$ be the subset of U that is first covered by some set in $\mathcal{A}_i$, $\{u : m_{i-1} < \tau(u) \leq m_i\}$.

We can upper-bound the cost of PCMSSC-Greedy by $\sum_i |R_i||\mathcal{A}_i|$. At worst, each of the X_i items is covered by the last set in $\mathcal{A}_i$, number m_i; hence, at iteration i, appending (a permutation of) $\mathcal{A}_i$ to π increases the cover time of all items in R_i by (at most) $|\mathcal{A}_i|$.

To prove the approximation factor, we adapt the argument of Feige et al. [10]. Consider a plot of cover time against item number, where we order $u \in U$ by $\tau(u)$ (Figure 1).

That is, on $(u-1, u]$ the plot has height $\tau(u)$, and the plot is non-decreasing on $(0, n]$. The PCMSSC solution cost is the area under plot on $(0, n]$. The upper bound for the PCMSSC-Greedy solution in the previous paragraph can be viewed as a series of horizontal slices, of height $|\mathcal{A}_i|$ and width $|R_i|$, with slices "right-aligned". That is, slice i is the rectangle $(n - |R_i|, n] \times (m_{i-1}, m_i]$. The plot defined by the upper boundary of this series of slices, which we call $\mathcal{B}$, lies not below the plot for PCMSSC-Greedy. We show that, with area shrunk by factor of $1/(4\alpha)$, a mapping of plot $\mathcal{B}$ lies not above the plot for (every) optimal solution, OPT, on $(0, n]$. Since the area under the plot represents solution cost, we conclude that PCMSSC-Greedy is a $4 \cdot \alpha$ approximation.

Mapping. We map slice i to a column of height $h_i \equiv |\mathcal{A}_i||R_i|/(2\alpha|X_i|)$ and width $|X_i|/2$, positioned between $|R_i|/2$ and $|R_{i+1}|/2$ "elements" from the right-hand end of the curve for

Algorithm 2 Algorithm MDPCS-Greedy.

```
function MDPCS-Greedy($\mathcal{G}$, $\prec$, $R$)
    $\mathcal{A} \leftarrow \mathcal{G}$
    for each $S \in \mathcal{G}$ do
        if $\Delta(\mathcal{P}[S], R) > \Delta(\mathcal{A}, R)$ then
            $\mathcal{A} \leftarrow \mathcal{P}[S]$
    return $\mathcal{A}$
```

OPT. This column is the rectangle $(n - |R_i|/2, n - |R_{i+1}|/2] \times (0, h_i]$. (Sets appended to π after all elements of U are covered do not contribute to the solution cost, so we omit them from this analysis, and thus assume $|X_i| > 0$.) Since $h_i|X_i|/2 = |\mathcal{A}_i||R_i|/(4\alpha)$, this mapping produces a plot whose area is a factor 4α smaller than plot $\mathcal{B}$. The following claim suffices to prove Lemma 1, where OPT$[j]$ is the prefix of j sets in permutation OPT (of $\mathcal{F}$).

▶ **Claim 2.** *For all i with $|R_i| > 0$, $|\text{cov}(\text{OPT}[\lfloor h_i \rfloor], R_i)| \le |R_i|/2$.*

The proof of Claim 2 follows soon. Meanwhile, finalizing the proof of Lemma 1, Claim 2 shows that even after the first $\lfloor h_i \rfloor$ sets of solution OPT, there are at least $|R_i|/2$ uncovered items (of R_i). Therefore the plot for OPT rises to a height at least $\lfloor h_i \rfloor + 1 \ge h_i$, at a horizontal position at most $n - |R_i|/2$. For all i, the top-left corner of the i^{th} mapped column is at position $(h_i, n - |R_i|/2)$; since the plot for OPT is non-decreasing, this rectangle fits entirely within the plot for OPT, and we have the desired shrunken plot. ◀

Proof of Claim 2. Let $\pi \circ j$ stand for π after $j-1$ iterations of the loop in PCMSSC-Greedy. Abusing notation (as OPT and π are sequences, not families of sets), if $\text{OPT}[\lfloor h_i \rfloor] \subseteq \pi \circ i$, the claim is trivially true, since $|\text{cov}(\text{OPT}[\lfloor h_i \rfloor], R_i)| = 0$. We hence assume that $\text{OPT}[\lfloor h_i \rfloor] \setminus \pi \circ i$ is non-empty. On its i^{th} instantiation, algorithm D returns a sub-family, $\mathcal{A}_i$, whose density $\Delta(\mathcal{A}_i, R_i)$ is $\ge 1/\alpha$ times the maximum-density precedence-closed subfamily of sets. Since sub-family $\text{OPT}[\lfloor h_i \rfloor]$ is precedence closed with respect to $\mathcal{F}$, so is $\text{OPT}[\lfloor h_i \rfloor] \setminus \pi \circ i$ with respect to $\mathcal{G}_i = \mathcal{F} \setminus \pi \circ i$, and it was thus "considered" by D, so

$$\frac{|\text{cov}(\text{OPT}[\lfloor h_i \rfloor] \setminus \pi \circ i), R_i)|}{|\text{OPT}[\lfloor h_i \rfloor] \setminus \pi \circ i|} \le \alpha \cdot \frac{|X_i|}{|\mathcal{A}_i|}\,. \tag{1}$$

Now, R_i is exactly those items not in $\text{cov}(\pi \circ i)$, and $|\text{OPT}[\lfloor h_i \rfloor] \setminus \pi \circ i| \le \lfloor h_i \rfloor$, so inequality (1) leads to $|\text{cov}(\text{OPT}[\lfloor h_i \rfloor], R_i)| \le \alpha\, h_i\, |X_i|/|\mathcal{A}_i|$. Substituting $h_i = |\mathcal{A}_i||R_i|/(2\alpha|X_i|)$ into this, we obtain $|\text{cov}(\text{OPT}[\lfloor h_i \rfloor], R_i)| \le |R_i|/2$. ◀

3 Algorithms for Max-Density Precedence-Closed Subfamily

In this section, we introduce a general greedy method for MDPCS, optimal on in-trees, and an alternative approach for out-tree forests, with factor equal to maximum tree height.

3.1 Greedy

Recall that the input to MDPCS is a family of sets $\mathcal{G}$ and a set of items to be covered R. Let $\mathcal{P}[S]$ be the minimal precedence-closed sub-family of $\mathcal{G}$ containing $S \in \mathcal{G}$: that is, the *ancestors* of S (including S itself). Our greedy algorithm for MDPCS, in Algorithm 2 (which we call MDPCS-Greedy), returns the denser of $\mathcal{G}$ and the best of the $\mathcal{P}[S]$ solutions.

By the definition of $\mathcal{P}$, MDPCS-Greedy returns a feasible solution. We let $\delta_{\mathcal{P}\text{-max}}$ stand for the maximum of $\Delta(\mathcal{P}[S], R)$ over all $S \in \mathcal{G}$. If every set in $\mathcal{G}$ covers at least one item in R, then $\delta_{\mathcal{P}\text{-max}} \geq 1$. However, since PCMSSC-Greedy could involve a sequence of depleting instances of MDPCS, there is no guarantee that each set in $\mathcal{G}$ contains an item in R.

▶ **Lemma 3.** MDPCS-Greedy *is a* $\sqrt{m}$ *approximation to* MDPCS. *If* $\delta_{\mathcal{P}\text{-max}} \geq 1$*,* MDPCS-Greedy *is a* $\sqrt{n}$ *approximation to* MDPCS.

Proof. Consider some optimal solution sub-family, OPT, and let k stand for its cardinality. For each $S_1, S_2, \ldots, S_k$ in OPT, the density of $\mathcal{P}[S_i]$ is at most $\delta_{\mathcal{P}\text{-max}}$ and, by definition, $\mathcal{P}[S_i]$ is a sub-family of OPT. Therefore,

$$|\text{cov}(\text{OPT}, R)| = |\cup_{i=1}^{k} \text{cov}(\mathcal{P}[S_i], R)| \leq \sum_{i=1}^{k} |\text{cov}(\mathcal{P}[S_i], R)| \leq \delta_{\mathcal{P}\text{-max}} \sum_{i=1}^{k} |\mathcal{P}[S_i]| \leq \delta_{\mathcal{P}\text{-max}} \cdot k^2 \,,$$

where the first inequality observes the definition of union, while the remarks above justify the second and third inequalities. Therefore $\Delta(\text{OPT}, R)/\delta_{\mathcal{P}\text{-max}} \leq k$.

On the other hand, $\Delta(\text{OPT}, R) \leq |R|/k$, but MDPCS-Greedy returns $\mathcal{A}$ with $\Delta(\mathcal{A}, R) \geq \Delta(\mathcal{G}, R) = |R|/m$. Hence the approximation ratio is at most $\min(k, m/k) \leq \sqrt{m}$. If $\delta_{\mathcal{P}\text{-max}} \geq 1$, then the approximation ratio is at most $\min(k, |R|/k) \leq \sqrt{|R|} \leq \sqrt{n}$. ◀

Indeed, these factors for MDPCS-Greedy are tight, up to a factor two. And without the assumption $\delta_{\mathcal{P}\text{-max}} \geq 1$, there are instance collections on which MDPCS-Greedy achieves only an $\Omega(n)$ approximation. Alternatively, we could *iterate* MDPCS-Greedy, repeatedly choosing a sub-family of the form $\mathcal{P}[S]$ (for some $S \in \mathcal{G}$) that when *added* to the current solution maximizes the density of the sub-family, similar to the greedy algorithm for Set Cover. Again, there is a collection of instances in which this scheme returns only an $O(\sqrt{n})$ (or $O(\sqrt{m})$) approximation.

3.2 Forest of in-trees

If graph $G(\mathcal{F}, \prec)$, and hence graph $G(\mathcal{G}, \prec)$ has a special structure, similarly explored in the context of Partially Ordered Knapsack [15], MDPCS admits better approximation factors. We start with G a forest of in-trees: for all A in $\mathcal{F}$, at most one set immediately depends on A, that is $|\{B \in \mathcal{F} : A \prec B\}| \leq 1$. Consequently, for all $A, B \in \mathcal{F}$, *either* $\mathcal{P}[A]$ and $\mathcal{P}[B]$ are disjoint, *or* (wlog) $A \in \mathcal{P}[B]$. Therefore, a solution in such an input is a union of disjoint sub-families $\mathcal{P}[S_1], \mathcal{P}[S_2], \ldots$. In an optimal solution, each sub-family $\mathcal{P}[S_i]$ has optimum density, so MDPCS-Greedy will (in polynomial time) find an optimal solution.

3.3 Forest of out-trees

We consider the "opposite" scenario, in which the in-degree of each set is at most one: that is, for all A, $|\{B \in \mathcal{F} : B \prec A\}| \leq 1$. Focusing on the graph $G(\mathcal{F}, \prec)$, each connected component of G is a rooted out-tree. Here, we introduce another greedy algorithm, which provides an approximation factor equal to the largest tree height. It acts recursively, adopting the approach of Chekuri and Kumar [6] for the Maximum Coverage Problem with Group Budget Constraints, and so adds 1 to the approximation factor at each tree level.

Let OPT be some optimal solution to MDPCS on $\mathcal{G}$, and let δ_{OPT} be its density, $\Delta(\text{OPT}, R)$: therefore, $\text{cov}(\text{OPT}, R) - \delta_{\text{OPT}}|\text{OPT}| = 0$. Our recursive algorithm $D^{\mathcal{T}}$, shown in Algorithm 3, has as input $(\sigma + 1, T, \delta_{\text{OPT}}, R')$, where $\sigma + 1$ is an "approximation factor", T a tree, and $R' \subseteq R$ a subset of items to be covered. Let $t \sqsubset T$ denote that t is a subtree of T

Algorithm 3 Algorithm $D^{\mathcal{T}}$ called with $(\sigma+1, T, \delta_{\text{OPT}}, R')$.

```
1:  function $D^{\mathcal{T}}(\sigma+1, T, \delta_{\text{OPT}}, R')$
2:      $r \leftarrow$ root of $T$
3:      $R'_0 \leftarrow R'$, $\mathcal{A}_T^0 \leftarrow \{r\}$
4:      Solutions $\leftarrow \{\varnothing, \mathcal{A}_T^0\}$
5:      $R' \leftarrow R' \setminus \text{cov}(\mathcal{A}_T^0)$
6:      $\kappa \leftarrow$ number of children of $r$
7:      for $j = 1$ to $\kappa$ do
8:          Let $T_j$ be the subtree rooted at the $j^{\text{th}}$ child of $r$
9:      for $i = 1$ to $\kappa$ do
10:         Candidates $\leftarrow \varnothing$
11:         for each child $j$ of $r$ s.t. no sub-family of $T_j$ has yet been added to $\mathcal{A}_T^{i-1}$ do
12:             Add the output $\mathcal{A}_T^{i,j}$ of $D^{\mathcal{T}}(\sigma, T_j, \delta_{\text{OPT}}, R')$ to Candidates
13:         Let $\mathcal{A}_T^{i,j*}$ be the tree $t$ in Candidates that maximizes $\sigma|\text{cov}(t, R')| - \delta_{\text{OPT}}|t|$
14:         $\mathcal{A}_T^i \leftarrow \mathcal{A}_T^{i-1} \cup \{\mathcal{A}_T^{i,j*}\}$            ▷ By construction, $\mathcal{A}_T^i$ is a tree.
15:         Add $\mathcal{A}_T^i$ to Solutions
16:         $R' \leftarrow R' \setminus \text{cov}(\mathcal{A}_T^i)$
17:     return $\mathcal{A}_T$, the tree $t$ in Solutions that maximizes $\sigma|\text{cov}(t, R'_0)| - \delta_{\text{OPT}}|t|$
```

sharing T's root. Let $M(T, R')$ be the $t \sqsubset T$ that maximizes $|\text{cov}(t, R')| - \delta_{\text{OPT}}|t|$. Given tree T of height at most $\sigma+1$, we show inductively that $D^{\mathcal{T}}$ returns some $\mathcal{A}_T \sqsubset T$ with

$$(\sigma+1)|\text{cov}(\mathcal{A}_T, R')| - \delta_{\text{OPT}}|\mathcal{A}_T| \geq |\text{cov}(M(T, R'), R')| - \delta_{\text{OPT}}|M(T, R')|\,. \tag{2}$$

Broadly, algorithm $D^{\mathcal{T}}$behaves as follows. If the root r of tree T has κ children, $D^{\mathcal{T}}$makes a sequence of κ recursive calls to itself. After the $i-1^{\text{th}}$ call, it has a putative solution $\mathcal{A}_T^{i-1}$ comprising r itself and $i-1$ subtrees, each hanging from a different child of r. In the i^{th} iteration, $D^{\mathcal{T}}$adds a subtree from a "new" child to $\mathcal{A}_T^{i-1}$. This new subtree has the maximum value of $\sigma\text{cov}(t, R'_i) - \delta_{\text{OPT}}|t|$, where R'_i is set R' during the i^{th} iteration (before step 16). With first parameter $\sigma+1$, $D^{\mathcal{T}}$returns the best of the $\kappa+2$ putative solutions (including $\varnothing$ and $\{r\}$), the subtree $\mathcal{A}_T$ maximizing $\sigma|\text{cov}(\mathcal{A}_T, R')| - \delta_{\text{OPT}}|\mathcal{A}_T|$.

▶ **Lemma 4.** *Given tree T of height $\leq \sigma+1$, $D^{\mathcal{T}}(\sigma+1, T, \delta_{OPT}, R')$, returns $\mathcal{A}_T \sqsubset T$ satisfying inequality* (2).

Proof. First, the base case. If $\sigma = 0$, and the tree has height 1, the only options are $\varnothing$ and $\{r\}$. These are easy to evaluate and inequality (2) is easily satisfied.

If $\sigma > 0$, consider tree $M(T, R'_0)$, where R'_0 is the initial *value* of R' in $D^{\mathcal{T}}$. Again, if $M(T, R'_0)$ is empty, or if it is $\{r\}$, $D^{\mathcal{T}}$ will consider those two solutions, so will return some solution satisfying (2). Therefore, assume tree $M(T, R'_0)$ comprises root r and subtrees hanging from $\kappa^* \leq \kappa$ children. We focus analysis on $\mathcal{A}_T^{\kappa^*}$. Although $D^{\mathcal{T}}$does not know κ^*, it generates $\mathcal{A}_T^i$ for all $i \leq \kappa$, returning the *best* of these, at least *as good* as $\mathcal{A}_T^{\kappa^*}$.

We renumber root r's children to match the order in which they contribute to $\mathcal{A}_T^{\kappa^*}$, so that $j^* = i$ on each iteration. The construction of $M(T, R'_0)$ can also be interpreted iteratively, so that at iteration i it adds a subtree hanging from child number i_M; let that subtree be called $M_{i_M}(T, R'_0)$. However, we insist that if $M(T, R'_0)$ contains a subtree hanging from child $i \leq \kappa^*$, it is *chosen* at iteration i.

Consider the subtree added to $\mathcal{A}_T^{\kappa^*}$ in iteration i, $\mathcal{A}_T^{i,i}$. If $\mathcal{A}_T^{i,i}$ is different from $M_{i_M}(T, R'_0)$, it must be because $D^{\mathcal{T}}(\sigma, T_i, \delta_{\text{OPT}}, R'_i)$ returned $\mathcal{A}_T^{i,i}$, which had the largest value for $t \in$

Candidates (children numbered i and above) of $\sigma|\mathrm{cov}(t, R'_i)| - \delta_{\mathrm{OPT}}|t|$, while tree $M(T, R'_0)$ added a subtree from child $i_M \geq i$, giving inequality (3), as follows

$$\begin{aligned}\sigma|\mathrm{cov}(\mathcal{A}_T^{i,i}, R'_i)| - \delta_{\mathrm{OPT}}|\mathcal{A}_T^{i,i}| &\geq \sigma|\mathrm{cov}(\mathcal{A}_T^{i,i_M}, R'_i)| - \delta_{\mathrm{OPT}}|\mathcal{A}_T^{i,i_M}| &(3)\\ &\geq |\mathrm{cov}(M(T_{i_M}, R'_i), R'_i)| - \delta_{\mathrm{OPT}}|M(T_{i_M}, R'_i)| &(4)\\ &\geq |\mathrm{cov}(M_{i_M}(T, R'_0), R'_i)| - \delta_{\mathrm{OPT}}|M_{i_M}(T, R'_0)|\,, &(5)\end{aligned}$$

while inequality (4) arises from the inductive argument about $D^{\mathcal{T}}(\sigma, \ldots)$, while inequality (5) flows from the optimality of $M(T_{i_M}, R'_i)$ on (T_{i_M}, R'_i). If in fact $\mathcal{A}_T^{i,i}$ is the same as $M_{i_M}(T, R'_0)$, then the *overall* inequality (3) – (5) holds because $\sigma \geq 1$.

The coverage of $\mathcal{A}_T^{\kappa^*}$ on R'_0 is the union of $\mathrm{cov}(\{r\}, R'_0)$ and $\cup_{i=1}^{\kappa^*}\mathrm{cov}(\mathcal{A}_T^{i,i}, R'_i)$. From the definition of R'_i (step 16 of $D^{\mathcal{T}}$), the $\mathrm{cov}(\cdot)$ sets are disjoint. Since also $\sigma \geq 1$,

$$\begin{aligned}&\sigma|\mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)| - \delta_{\mathrm{OPT}}|\mathcal{A}_T^{\kappa^*}|\\ &\geq [|\mathrm{cov}(\{r\}, R'_0)| - \delta_{\mathrm{OPT}}] + \sum_{i=1}^{\kappa^*}\left[\sigma|\mathrm{cov}(\mathcal{A}_T^{i,i}, R'_0)| - \delta_{\mathrm{OPT}}|\mathcal{A}_T^{i,i}|\right]\,,\end{aligned}$$

and, by applying *overall* inequality (3) – (5), this is

$$\begin{aligned}&\geq [|\mathrm{cov}(\{r\}, R'_0)| - \delta_{\mathrm{OPT}}] + \sum_{i=1}^{\kappa^*}[|\mathrm{cov}(M_{i_M}(T, R'_0), R'_i)| - \delta_{\mathrm{OPT}}|M_{i_M}(T, R'_0)|]\\ &= |\mathrm{cov}(\{r\}, R'_0)| + \left[\sum_{i=1}^{\kappa^*}|\mathrm{cov}(M_{i_M}(T, R'_0), R'_i)|\right] - \delta_{\mathrm{OPT}}|M(T, R'_0)|\,, \qquad (6)\end{aligned}$$

For each iteration i, inspired by Chekuri and Kumar [6], we have

$$\mathrm{cov}(M_{i_M}(T, R'_0), R'_i) \supseteq \mathrm{cov}(M_{i_M}(T, R'_0), R'_0 \setminus \mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)\,)\,,$$

$$\text{so,}\quad \mathrm{cov}(\{r\}, R'_0) \cup \bigcup_{i=1}^{\kappa^*}\mathrm{cov}(M_{i_M}(T, R'_0), R'_i) \supseteq \left(\mathrm{cov}(\{r\}, R'_0) \cup \bigcup_{i=1}^{\kappa^*}\mathrm{cov}(M_{i_M}(T, R'_0), R'_0)\right) \setminus \mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)\,.$$

Applying the union bound to the LHS and the composition of $M(T, R'_0)$ to the RHS,

$$\begin{aligned}|\mathrm{cov}(\{r\}, R'_0)| + \sum_{i=1}^{\kappa^*}|\mathrm{cov}(M_{i_M}(T, R'_0), R'_i)| &\geq |\mathrm{cov}(M(T, R'_0), R'_0) \setminus \mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)|\\ &\geq |\mathrm{cov}(M(T, R'_0), R'_0)| - |\mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)|\,.\end{aligned}$$

Combining this with the inequality ending at (6), we see that

$$\begin{aligned}\sigma|\mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)| - \delta_{\mathrm{OPT}}|\mathcal{A}_T^{\kappa^*}| &\geq \Big(|\mathrm{cov}(M(T, R'_0), R'_0)| - |\mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)|\Big) -\\ &\qquad \delta_{\mathrm{OPT}}|M(T, R'_0)|\,,\\ \therefore\quad (\sigma+1)|\mathrm{cov}(\mathcal{A}_T^{\kappa^*}, R'_0)| - \delta_{\mathrm{OPT}}|\mathcal{A}_T^{\kappa^*}| &\geq |\mathrm{cov}(M(T, R'_0), R'_0)| - \delta_{\mathrm{OPT}}|M(T, R'_0)|\,. \qquad \blacktriangleleft\end{aligned}$$

For a forest of out-trees, we return the best individual tree solution generated by $D^{\mathcal{T}}$. Without knowing the value of δ_{OPT}, there are only mn possible values, so we can in polynomial time try them all and return the densest sub-forest. Finally, for OPT, the right-hand side of inequality (2) is zero, so Lemma 4 shows $D^{\mathcal{T}}$returns a tree with density at least δ_{OPT} divided by maximum tree height.

4 Hardness of approximation

Apart from the in-tree case, which is in P, the approximation *factors* for PCMSSC are polynomial. In this section, we show that such factors are to be "expected". We show a hardness *reduction* to PCMSSC from the Planted Dense Subgraph Conjecture, which is a statement about the difficulty of finding a dense component in an Erdős-Rényi graph [5]. Our inspiration here is Burge, Munagala and Srivastava's hardness-of-approximation reduction from Densest k-Subgraph for pipelined query operators [4].

First, we define the Planted Dense Subgraph Conjecture, where N is the input graph's order, α its log density, k the order of the planted component, and β its log density.

▶ **Definition 5.** The problem $\mathrm{PDS}(N, k, \alpha, \beta)$ has as input a graph with probability $1/2$ drawn from $G(N, N^{\alpha-1})$ and with probability $1/2$ drawn from $G(N, N^{\alpha-1})$, in which some k vertices are chosen uniformly from N and on them is added a subgraph drawn from $G(k, k^{\beta-1})$. The task is to correctly report from which of the two distributions the graph was drawn.

▶ **Conjecture 6** (Planted Dense Subgraph Conjecture). *For all $\varepsilon > 0$, $k \geq \sqrt{N}$, and $\beta < \alpha$, no probabilistic polytime algorithm can, with advantage $> \varepsilon$, solve* $\mathrm{PDS}(N, k, \alpha, \beta)$.

We show that identifying an ordering π with low PCMSSC score solves PDS almost surely.

▶ **Lemma 7.** *Assuming the* Planted Dense Subgraph Conjecture*, there is no poly-time algorithm that, for $\varepsilon > 0$, approximates* PCMSSC *within factor $O(n^{1/6-\varepsilon})$ nor $O(m^{1/12-\varepsilon})$.*

Proof. First, let $k = \sqrt{N}$, $\alpha = 1/2$ and $\beta = 1/2 - \gamma$, for some $\gamma > 0$. Given graph $H([N], \mathcal{E})$, the input to PDS, define family $\mathcal{F}$ to be $N\lambda$ *vertex sets*, together with an *edge set* for each edge in $\mathcal{E}$. More specifically, the vertex sets are $V_{u,i}$ for each $u \in \{1, \ldots, N\}$ and $i \in \{1, \ldots, \lambda\}$, while the edge sets are $E_{u,v}$ for each $(u, v) \in \mathcal{E}$. For convenience, let $U = \{0, 1, \ldots, n\}$, so that $|U| = n + 1$, with $U^+ = \{1, \ldots, n\}$.

The $\prec$ relation acts as follows: every edge set must be preceded by all "copies" of each of its endpoints' vertex sets. That is, for all $(u, v) \in \mathcal{E}$ and for all i, $V_{u,i} \prec E_{u,v}$ and $V_{v,i} \prec E_{u,v}$.

We now define the composition of the sets in $\mathcal{F}$. Every vertex set comprises the same item: $V_{u,i} = \{0\}$, for all u, i. To define the edge sets, we *associate* items with vertices: let the set of items associated with vertex v be U_v. (This association is merely a vehicle to define edge sets, and is distinct from vertex-set composition.) The construction is randomized, based on parameter $p \in (0, 1)$: each item in U^+ is associated with each vertex in H independently, with probability p. Hence $\mathbf{E}[|U_v|] = np$, while for all $j \in U^+$, $\mathbf{E}[|\{v : j \in U_v\}|] = Np$. For each edge (u, v), we define $E_{u,v}$ to be $U_u \cap U_v$, with expected size np^2.

Choosing p. If there is a planted component, $\mathcal{P}$, it (just) covers all of U: this drives our selection of p. There are $\sqrt{N}$ vertices in $\mathcal{P}$, and each item j is in $\sqrt{N}p$ of the U_v sets, on average. We expect (ignoring constants) around Np^2 vertex pairs in $\mathcal{P}$ where j is associated with both. The probability of each vertex pair having an edge is, independently, $k^{\beta-1} = (N^{1/2})^{(-1/2-\gamma)} = N^{-(1/4+\gamma/2)}$. Therefore, $\mathbf{E}[|E_{u,v} \in \mathcal{P} : j \in E_{u,v}|] \approx Np^2 N^{-(1/4+\gamma/2)} = N^{3/4-\gamma/2}p^2$. So this is close to 1, we set $p = 32 \cdot N^{(-3+\gamma')/8} \log N$, with $\gamma' = 2\gamma$.

Planted component. If there is a planted component $\mathcal{P}$, we show a "good" PCMSSC solution. Suppose that π starts with all $V_{u,i}$ for all $u \in \mathcal{P}$ followed by $E_{u,v}$ for all $u, v \in \mathcal{P}$ (edges $\mathcal{E}$ in $\mathcal{P}$). The number of sets so far is $\lambda\sqrt{N}$ plus a random value highly concentrated around $\sqrt{N}(\sqrt{N} - 1)N^{-(1/4+\gamma/2)}/2$, hence $\leq N^{3/4-\gamma/2}$ with high probability (whp). The claim below proves U is covered whp by these $\mathcal{P}$-based sets. If so, the PCMSSC cost is whp bounded by $n(\lambda\sqrt{N} + N^{3/4-\gamma/2})$.

▶ **Claim 8.** *If H was generated with a planted component, $\mathcal{P}$, then with high probability all items in U^+ are covered by the planted component's edge sets.*

Proof of Claim 8. The expected number of U_v in $\mathcal{P}$ containing j is $\sqrt{N}p$, so

$$\Pr[|v \in \mathcal{P} : j \in U_v| \leq \sqrt{N}p/2] \leq \exp(-\sqrt{N}p/8) = \exp(-4 \cdot N^{(1+\gamma')/8} \log N)\,,$$

via Chernoff bounds, which is tiny. Therefore, and since $\binom{x}{2} \geq x^2/4$ for large enough x, the probability that the number of pairs of vertices in $\mathcal{P}$ where j is associated with both is at most $Np^2/16$ is (also) at most $\exp(-4 \cdot N^{(1+\gamma')/8} \log N)$.

For each such pair, there is an edge actually present with probability $N^{-(1/4+\gamma/2)}$. Item-association and edge-presence events are independent, so in expectation j is in at least

$$\mu = (Np^2/16)N^{-(1/4+\gamma/2)} = 32^2(\log^2 N)/16$$

edge sets in $\mathcal{P}$. For all $j \in U^+$, again via Chernoff bounds,

$$\Pr[|\{(u,v) \in \mathcal{E}_{\mathcal{P}} : j \in E_{u,v}\}| \leq \mu/2] \leq \exp(-(64 \log N)/8) = N^{-8}\,.$$

Taking the union over all n items in U^+, the probability that some item is uncovered by the edge sets in $\mathcal{P}$ is at most n/N^8, which is very small (below, we choose n to be $o(N)$). ◀

No planted component. We show that if there is no planted component, even after several vertex and edge sets have appeared in π, there is a significant portion of items not yet covered, pushing the PCMSSC score very high.

Consider a solution to PCMSSC derived from x vertices and their induced edges. The number of vertex sets is λx and the number of vertex pairs is $\approx x^2/2$. The expected number of edges is $\leq x^2/\sqrt{N}$, but there are in fact $\binom{N}{x} \leq N^x$ such sets of N vertices. Via the Chernoff bound, the probability that all of them have at most 3/2 their average number of edges, $\mu_{\mathcal{E}}$, is at most $N^x \exp(-\mu_{\mathcal{E}}/12)$. If we would like this probability to be at most $1/N^8$, say, then let $\mu_{\mathcal{E}} \geq 12(x+8)\log N$. Since the number of vertex pairs is at least $x^2/4$, and hence $\mu_{\mathcal{E}} \geq x^2/(4\sqrt{N})$, there is a very low probability of exceeding $x^2/\sqrt{N}$ edges across all sets of x vertices if $x \geq 500\sqrt{N}\log N$.

By construction, each edge set has on average np^2 items. The probability that some edge set related to at least one of the at most $x^2/\sqrt{N}$ edges has more than $2np^2$ items is at most $(x^2/\sqrt{N})\exp(-np^2/12)$. If $n \geq 200 \log N/p^2$, which is satisfied by $n \in \Omega(N^{3/4})$, because $x \leq N$, this probability is tiny.

Even ignoring the possibility of edge sets covering common items, we conclude that whp, after λx vertex sets and $x^2/\sqrt{N}$ edge sets appear in π, at most $2np^2x^2/\sqrt{N} \leq 2048 \cdot nN^{(-5+\gamma')/4}x^2(\log^2 N)$ items have been covered. Hence if $x \leq N^{(5-2\gamma')/8}$, then there are $o(n)$ items covered. With $\Omega(n)$ items uncovered, the PCMSSC score is at least a constant multiplied by $n(\lambda N^{(5-2\gamma')/8} + N^{(3-2\gamma')/4})$.

Letting λ grow to at least $N^{1/4}$ and letting γ' shrink, whp the asymptotic ratio between this cost and the cost in the planted case (just before Claim 8) tends arbitrarily close to $N^{1/8}$. Since $n \in \Theta(N^{3/4})$ permits our probability bounds, whp there is a gap of $\Theta(n^{1/6-\varepsilon})$, for all $\varepsilon > 0$, in the PCMSSC costs between the two cases. Likewise, the number of sets m in the input is highly concentrated around $\lambda N + N^{3/2}$, which is $\Theta(N^{3/2})$. An algorithm for PCMSSC with an approximation factor better than $\Theta(n^{1/6-\varepsilon})$ or $\Theta(m^{1/12-\varepsilon})$, for all $\varepsilon > 0$, could solve PDS with very significant advantage. From this, we conclude Lemma 7. ◀

5 Related work

The Min-Sum Vertex Cover problem – the special case of MSSC where each set has two items – admits a 2-approximation algorithm [10]. In database theory, MSSC has been referred to as the Pipelined Set Cover problem. This model typically allows for different processing times (or costs) on the sets, and has been given an alternative (but still factor-4) approximation algorithm [19].

In web search theory, a generalization of MSSC relating the min-sum objective to Minimum Latency Set Cover is called Multiple Intents Reranking. This extends the objective function to allow for different *user profiles* – relating to how many times each item needs to be covered [1]. It has also been referred to as the Generalized Min-Sum Set Cover problem, which has constant-factor approximations [2].

In Scheduling with AND/OR Precedence Constraints (SAOPC), there are two types of jobs: AND-jobs are available only when all precedences are met, while OR-jobs only require that at least one of the precedences are met [11]. In fact, PCMSSC is a special case of SAOPC in which the precedence-constrained sets are AND-jobs, and the item are OR-jobs. Scheduling to minimize the makespan is Label Cover-hard for general AND/OR precedences [13], but the reduction has an OR-AND-OR-AND structure: it is unclear whether PCMSSC is Label Cover-hard, that is [8], hard to approximate within $2^{\log^{1-1/(\log\log^c n)} n}$.

Erlebach, Kääb, and Möhring prove that minimizing the total completion time in SAOPC is Label Cover-hard, but that scheduling available jobs with Smith's shortest processing time (SPT) rule gives an $O(n)$-approximation [9]. They further prove that this algorithm gives an $O(\sqrt{n})$-approximation for the special case with a single processor where all jobs have equal weights. However, in PCMSSC, AND jobs have zero weight and unit processing time, but OR jobs have unit weight and zero processing time.

Finally, in the Partially Ordered Knapsack (aka POK), there is a predecessor $\prec$ relation, inducing a directed graph, and each vertex has a profit and a weight [15]. The aim is to find a *closed under predecessor* set of vertices that maximizes total profit subject to a total weight constraint. For in- and out-trees, Johnson and Niemi consider pseudo-polynomial algorithms that are nonetheless efficient in practice, as well as an FPTAS (with running time proportional to $1/\varepsilon$) derived from a standard PTAS approach for knapsack problems [15].

Kolliopoulos and Steiner [16] provide an FPTAS for POK when the underlying order is two dimensional. They note that Hajiaghayi et al. [14] show that POK is hard to approximate within $2^{\log^{\delta} n}$, that POK generalizes Densest k-Subgraph, and that a constant-factor algorithm for Partially Ordered Knapsack would provide a sub-factor-2 approximation for PCSTW. Indeed, the Minimum-Rank Precedence-Closed Subgraph problem that Chekuri and Motwani solve optimally is a minimize-ratio version of POK; again, there is some connection to the difference in approximability between Densest Subgraph and Densest k-Subgraph. Borradaile, Heeringa, and Wilfong [3] examine variants of POK with undirected graphs, and 1-neighbor as well as all-neighbor precedence requirements.

Acknowledgements. We thank Tim Miller for the foundational software-testing problem.

References

1 Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *STOC: 41st ACM Symposium on Theory of Computing*, pages 669–678, 2009.

2 Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA: 21st ACM-SIAM Symposium on Discrete Algorithms*, pages 1539–1545, 2010.

3 Glencora Borradaile, Brent Heeringa, and Gordon Wilfong. The knapsack problem with neighbour constraints. *Journal of Discrete Algorithms*, 16:224–235, 2012.

4 Jen Burge, Kamesh Munagala, and Utkarsh Srivastava. Ordering pipelined query operators with precedence constraints. Technical Report 2005-40, Stanford InfoLab, 2005.

5 Moses Charikar, Yonatan Naamad, and Anthony Wirth. On approximating target set selection. In *APPROX: 19th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 4:1–4:16, 2016.

6 Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX: 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 72–83, 2004.

7 Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1):29–38, 1999.

8 Irit Dinur and Shmuel Safra. On the hardness of approximating label-cover. *Information Processing Letters*, 89(5):247–254, 2004.

9 Thomas Erlebach, Vanessa Kääb, and Rolf H Möhring. Scheduling AND/OR-networks on identical parallel machines. In *WOAO: International Workshop on Approximation and Online Algorithms*, pages 123–136, 2003.

10 Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

11 Donald W. Gillies and Jane W.-S. Liu. Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing*, 24(4):797–810, 1995.

12 A. V. Goldberg. Finding a maximum density subgraph. Technical Report CSD-84-171, UC Berkeley Computer Science Division, 1984.

13 Michael Goldwasser and Rajeev Motwani. Intractability of assembly sequencing: Unit disks in the plane. In *WADS: Workshop on Algorithms and Data Structures*, pages 307–320, 1997.

14 M Hajiaghayi, Kamal Jain, L Lau, I Măndoiu, Alexander Russell, and V Vazirani. Minimum multicolored subgraph problem in multiplex pcr primer set selection and population haplotyping. *Computational Science–ICCS 2006*, pages 758–766, 2006.

15 David S. Johnson and K. A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8(1):1–14, 1983.

16 Stavros G. Kolliopoulos and George Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.

17 Jesssica McClintock, Tim Miller, and Anthony Wirth. Prioritisation of test suites containing precedence constraints, 2017. submitted.

18 Tim Miller and Shifa-e-Zehra Haidry. Using dependency structures for prioritization of functional test suites. *IEEE Transactions on Software Engineering*, 39(2):258–275, 2013.

19 Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In *ICDT: 10th International Conference on Database Theory*, pages 83–98, 2005.

Jointly Stable Matchings[*]

Shuichi Miyazaki[1] and Kazuya Okamoto[2]

1 **Academic Center for Computing and Media Studies, Kyoto University, Kyoto, Japan**
shuichi@media.kyoto-u.ac.jp

2 **Division of Medical Information Technology and Administration Planning, Kyoto University Hospital, Kyoto, Japan**
kazuya@kuhp.kyoto-u.ac.jp

Abstract

In the stable marriage problem, we are given a set of men, a set of women, and each person's preference list. Our task is to find a stable matching, that is, a matching admitting no unmatched (man, woman)-pair each of which improves the situation by being matched together. It is known that any instance admits at least one stable matching. In this paper, we consider a natural extension where $k(\geq 2)$ sets of preference lists L_i $(1 \leq i \leq k)$ over the same set of people are given, and the aim is to find a *jointly stable matching*, a matching that is stable with respect to all L_i. We show that the decision problem is NP-complete already for $k = 2$, even if each person's preference list is of length at most four, while it is solvable in linear time for any k if each man's preference list is of length at most two (women's lists can be of unbounded length). We also show that if each woman's preference lists are same in all L_i, then the problem can be solved in linear time.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases stable marriage problem, stable matching, NP-completeness, linear time algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.56

1 Introduction

In this paper, we focus on the *stable marriage problem* [3] with incomplete preference lists (*SMI*). An instance I of SMI is a triple $I = (U, W, L)$, where U and W are the sets of men and women, respectively, such that $|U| = |W|(= n)$, and L is the set of $2n$ *preference lists*, one for each person. A person p's preference list in L is denoted by $L(p)$. Each person's preference list strictly orders a subset of the members of the opposite gender. If a person p is included in $L(q)$, we say that p is *acceptable* to q. If p is acceptable to q and vice versa, (p, q) is called an *acceptable pair*.

A *matching* is a set of acceptable (man, woman)-pairs in which no person appears more than once. For a matching M, a man m, and a woman w, if $(m, w) \in M$ then we write $M(m) = w$ and $M(w) = m$. If there is no w (respectively, m) such that $(m, w) \in M$, we say that m (respectively, w) is *single* or *unmatched* in M. For a matching M, if (i) (m, w) is an acceptable pair, (ii) m is single in M or prefers w to $M(m)$, and (iii) w is single in M or prefers m to $M(w)$, then we say that (m, w) is a *blocking pair* for M in L, or (m, w) *blocks* M in L. If there is no blocking pair for M in L, then we say that M is *stable* in L. It is well-known that any SMI instance admits at least one stable matching [3].

[*] This work was partially supported by JSPS KAKENHI Grant Numbers 16K00017 and 15K00466.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 56; pp. 56:1–56:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider an extension of SMI where two or more sets of preference lists are given. An instance I of the *Stable Marriage problem with k Incomplete lists* (*SMkI*) is a $(k+2)$-tuple $I = (U, W, L_1, L_2, \ldots, L_k)$, where U and W are the same as above, and each L_i is a set of preference lists. It asks if there exists a matching M that is stable in every L_i. Such a matching M is called *jointly stable*. Let a and b be positive integers. The restriction of SMkI where the lengths of preference lists of men are at most a and those of women are at most b is denoted by (a, b)-SMkI. If a (respectively, b) is ∞, it means that the lengths of men's (respectively, women's) preference lists are unbounded. Surprisingly, although this problem is a natural extension of the classical stable marriage problem, to the best of the authors' knowledge it has not been considered before in the literature. Note that since the number of stable matchings grows exponentially in the size of the input [7, 5, 13], an algorithm of enumerating all the stable matchings for each L_i and computing their intersection is not polynomial-time bounded.

Besides its theoretical interest, the problem has several applications: Consider a scenario of assigning medical residents to hospitals, where each resident needs to take training in three fixed clinical departments, e.g., surgery, pediatrics, and internal medicine, at an assigned hospital. A resident r ranks hospitals according to her preference, but her ranking of hospitals may differ depending on clinical departments. As a result, she has three (possibly different) preference lists over hospitals, $L_1(r)$ for surgery, $L_2(r)$ for pediatrics, and $L_3(r)$ for internal medicine. On the other hand, each clinical department may have its own criteria for ranking residents, so each hospital h has three independent preference lists over residents, $L_1(h)$ from surgery, $L_2(h)$ from pediatrics, and $L_3(h)$ from internal medicine. Clearly a blocking pair in some L_i may cause dissatisfaction to the corresponding resident and department, so we want to avoid such an assignment. Another example is a match making of Judo team competition. Suppose that there are five different weight classes, and one team consists of five players, each from each class. As a personal preference, a player p of team T who belongs to the weight class C is interested in only the players of the same class C, who are potential candidates for p's opponent. Therefore, each team has five preference lists corresponding to weight classes, and a matching avoiding blocking pairs in any class is desirable. Precisely speaking, the first and the second examples may be suitable to the Hospitals/Residents and the stable roommates, respectively, but we consider in this paper the stable marriage model as a first step.

1.1 Our Results

We show that $(4, 4)$-SMkI is NP-complete for any $k \geq 2$, while $(2, \infty)$-SMkI is solvable in time $O(kn)$ for any k. Therefore the complexity of $(3, \ell)$-SMkI for $\ell \geq 3$ is left open.

We also show that SMkI (with unbounded-length preference lists) is solvable in polynomial time if $L_1(w) = L_2(w) = \cdots = L_k(w)$ holds for every woman w. This can be thought of as a case where each woman has only one preference list, and one of its interpretations is a modification of the previous example of assigning residents to hospitals, where each resident has three preference lists as above, but each hospital has one preference list determined by e.g., a personnel director of the hospital, rather than three independent lists coming from each clinical department.

1.2 Related Work

As noted above, there seems to be no research on stable matching problems considering multiple preference lists over the same set of people. Only the related work we have found is the *bistable matching problem* introduced by Weems [14]; given an instance I of the stable marriage problem (where preference lists are complete), let $\hat{I}$ be the instance obtained by reversing the ordering of each preference list of I. A matching is *bistable* if it is stable in both I and $\hat{I}$. This is a special case of SM2I where all the preference lists are complete and $L_1(p)$ is a reversed order of $L_2(p)$ for every person p. Weems showed an $O(n^2)$-time algorithm to find a bistable matching or to report that none exists. Sethuraman and Teo [11] showed that the bistable roommates problem can also be solved in polynomial time. See pages 293–296 of [10] for a brief survey.

2 NP-completeness

In this section, we show the hardness result.

▶ **Theorem 1.** *For $k \geq 2$, $(4,4)$-SMkI is NP-complete.*

Proof. It is easy to see that $(4,4)$-SMkI is in NP. In the following, we show that $(4,4)$-SM2I is NP-hard. To show the NP-hardness for general k, one may simply set $L_2 = L_3 = \cdots = L_k$ in the reduction.

We give a polynomial-time reduction from the well-known NP-complete problem *3CNF SAT*. The definition of 3CNF SAT is as follows. Let x be a binary variable that takes 1(true) or 0(false). A *literal* is a variable x or its negation $\overline{x}$. A *clause* is a disjunction of literals, and a *Conjunctive Normal From (CNF) formula* is a conjunction of clauses. A *3CNF formula* is a CNF formula in which each clause contains at most three literals. An instance of 3CNF SAT is a 3CNF formula f and it asks if there exists an assignment to variables that makes f true. We may assume without loss of generality that each clause contains *exactly* three literals. (If a clause contains less than three literals, then repeat the same literal.)

Let f be an instance of 3CNF SAT, with variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$. We construct an instance I of $(4,4)$-SM2I. For each i $(1 \leq i \leq n)$, let s_i be the number of occurrences of the variable x_i. For the jth literal of the variable x_i $(1 \leq j \leq s_i)$, we introduce two men $a_{i,j}$ and $b_{i,j}$ and two women $c_{i,j}$ and $d_{i,j}$. We call them *literal men* and *literal women*. For each clause C_ℓ, we introduce nine men u^i_ℓ $(1 \leq i \leq 9)$ and nine women v^i_ℓ $(1 \leq i \leq 9)$. We call them *clause men* and *clause women*. Note that there are $15m$ men and $15m$ women in total.

The preference lists of literal people and clause people are given in Figures 1 and 2, respectively. In $a_{i,1}$ and $d_{i,1}$'s preference lists of L_2 in Fig. 1, $c_{i,j-1}$ and $b_{i,j-1}$ are null; hence their preference lists are of length two. Similarly, in b_{i,s_i} and c_{i,s_i}'s preference lists of L_2, $d_{i,j+1}$ and $a_{i,j+1}$ are null. We then explain $U_{i,j}$ and $V_{i,j}$ in Fig. 1. Suppose that the jth occurrence of x_i is the tth literal of the clause C_ℓ. If this literal is positive, then $U_{i,j}$ is null and $V_{i,j} = v^4_\ell$ if $t = 1$, $V_{i,j} = v^7_\ell$ if $t = 2$, and $V_{i,j} = v^1_\ell$ if $t = 3$. If it is negative, then $V_{i,j}$ is null and $U_{i,j} = u^1_\ell$ if $t = 1$, $U_{i,j} = u^4_\ell$ if $t = 2$, and $U_{i,j} = u^7_\ell$ if $t = 3$. Finally, we explain $B_{\ell,1}, B_{\ell,2}, B_{\ell,3}, D_{\ell,1}, D_{\ell,2}$, and $D_{\ell,3}$ in Fig. 2. Suppose that, for $t = 1, 2, 3$, the tth literal of the clause C_ℓ is the jth occurrence of x_i. If this literal is positive, then $D_{\ell,t}$ is null and $B_{\ell,t} = b_{i,j}$; otherwise, $B_{\ell,t}$ is null and $D_{\ell,t} = d_{i,j}$. Now the reduction is completed. It is not hard to see that the reduction can be performed in polynomial time and each person's preference list is of length at most four.

L_1	$a_{i,j}$:	$c_{i,j}$		$d_{i,j}$	$c_{i,j}$:	$b_{i,j}$		$a_{i,j}$
	$b_{i,j}$:	$d_{i,j}$	$V_{i,j}$	$c_{i,j}$	$d_{i,j}$:	$a_{i,j}$	$U_{i,j}$	$b_{i,j}$
L_2	$a_{i,j}$:	$c_{i,j}$	$c_{i,j-1}$	$d_{i,j}$	$c_{i,j}$:	$b_{i,j}$	$a_{i,j+1}$	$a_{i,j}$
	$b_{i,j}$:	$d_{i,j}$	$d_{i,j+1}$	$c_{i,j}$	$d_{i,j}$:	$a_{i,j}$	$b_{i,j-1}$	$b_{i,j}$

Figure 1 Preference lists of literal people corresponding to the jth occurrence of the variable x_i $(1 \leq j \leq s_i)$.

L_1	u_ℓ^1:	v_ℓ^1	v_ℓ^2	$D_{\ell,1}$	v_ℓ^3	v_ℓ^1:	u_ℓ^2	u_ℓ^3	$B_{\ell,3}$	u_ℓ^1
	u_ℓ^2:	v_ℓ^2	v_ℓ^3		v_ℓ^1	v_ℓ^2:	u_ℓ^3	u_ℓ^1		u_ℓ^2
	u_ℓ^3:	v_ℓ^3	v_ℓ^1		v_ℓ^2	v_ℓ^3:	u_ℓ^1	u_ℓ^2		u_ℓ^3
	u_ℓ^4:	v_ℓ^4	v_ℓ^5	$D_{\ell,2}$	v_ℓ^6	v_ℓ^4:	u_ℓ^5	u_ℓ^6	$B_{\ell,1}$	u_ℓ^4
	u_ℓ^5:	v_ℓ^5	v_ℓ^6		v_ℓ^4	v_ℓ^5:	u_ℓ^6	u_ℓ^4		u_ℓ^5
	u_ℓ^6:	v_ℓ^6	v_ℓ^4		v_ℓ^5	v_ℓ^6:	u_ℓ^4	u_ℓ^5		u_ℓ^6
	u_ℓ^7:	v_ℓ^7	v_ℓ^8	$D_{\ell,3}$	v_ℓ^9	v_ℓ^7:	u_ℓ^8	u_ℓ^9	$B_{\ell,2}$	u_ℓ^7
	u_ℓ^8:	v_ℓ^8	v_ℓ^9		v_ℓ^7	v_ℓ^8:	u_ℓ^9	u_ℓ^7		u_ℓ^8
	u_ℓ^9:	v_ℓ^9	v_ℓ^7		v_ℓ^8	v_ℓ^9:	u_ℓ^7	u_ℓ^8		u_ℓ^9

L_2	u_ℓ^1:	v_ℓ^1	v_ℓ^4	v_ℓ^2		v_ℓ^3	v_ℓ^1:	u_ℓ^2		u_ℓ^3	u_ℓ^7	u_ℓ^1
	u_ℓ^2:	v_ℓ^2		v_ℓ^3	v_ℓ^5	v_ℓ^1	v_ℓ^2:	u_ℓ^3	u_ℓ^8	u_ℓ^1		u_ℓ^2
	u_ℓ^3:	v_ℓ^3		v_ℓ^1		v_ℓ^2	v_ℓ^3:	u_ℓ^1		u_ℓ^2		u_ℓ^3
	u_ℓ^4:	v_ℓ^5	v_ℓ^7	v_ℓ^6		v_ℓ^4	v_ℓ^4:	u_ℓ^4		u_ℓ^5	u_ℓ^1	u_ℓ^6
	u_ℓ^5:	v_ℓ^6		v_ℓ^4	v_ℓ^8	v_ℓ^5	v_ℓ^5:	u_ℓ^5	u_ℓ^2	u_ℓ^6		u_ℓ^4
	u_ℓ^6:	v_ℓ^4		v_ℓ^5		v_ℓ^6	v_ℓ^6:	u_ℓ^6		u_ℓ^4		u_ℓ^5
	u_ℓ^7:	v_ℓ^9	v_ℓ^1	v_ℓ^7		v_ℓ^8	v_ℓ^7:	u_ℓ^9		u_ℓ^7	u_ℓ^4	u_ℓ^8
	u_ℓ^8:	v_ℓ^7		v_ℓ^8	v_ℓ^2	v_ℓ^9	v_ℓ^8:	u_ℓ^7	u_ℓ^5	u_ℓ^8		u_ℓ^9
	u_ℓ^9:	v_ℓ^8		v_ℓ^9		v_ℓ^7	v_ℓ^9:	u_ℓ^8		u_ℓ^9		u_ℓ^7

Figure 2 Preference lists of clause people corresponding to the ℓth clause.

We then proceed to the correctness proof. We first define partial matchings. For each i and j, we define $M_{i,j}^1 = \{(a_{i,j}, c_{i,j}), (b_{i,j}, d_{i,j})\}$ and $M_{i,j}^0 = \{(a_{i,j}, d_{i,j}), (b_{i,j}, c_{i,j})\}$. For each ℓ, we define

$$M_\ell^1 = \{(u_\ell^1, v_\ell^3), (u_\ell^2, v_\ell^1), (u_\ell^3, v_\ell^2), (u_\ell^4, v_\ell^4), (u_\ell^5, v_\ell^5), (u_\ell^6, v_\ell^6), (u_\ell^7, v_\ell^8), (u_\ell^8, v_\ell^9), (u_\ell^9, v_\ell^7)\},$$
$$M_\ell^2 = \{(u_\ell^1, v_\ell^2), (u_\ell^2, v_\ell^3), (u_\ell^3, v_\ell^1), (u_\ell^4, v_\ell^6), (u_\ell^5, v_\ell^4), (u_\ell^6, v_\ell^5), (u_\ell^7, v_\ell^7), (u_\ell^8, v_\ell^8), (u_\ell^9, v_\ell^9)\}, \quad \text{and}$$
$$M_\ell^3 = \{(u_\ell^1, v_\ell^1), (u_\ell^2, v_\ell^2), (u_\ell^3, v_\ell^3), (u_\ell^4, v_\ell^5), (u_\ell^5, v_\ell^6), (u_\ell^6, v_\ell^4), (u_\ell^7, v_\ell^9), (u_\ell^8, v_\ell^7), (u_\ell^9, v_\ell^8)\}.$$

Suppose that f is satisfiable and let T be a satisfying assignment. We will construct a jointly stable matching M for I. If $T(x_i) = 1$, then we let $M^1_{i,j} \subseteq M$ for all j. If $T(x_i) = 0$, then we let $M^0_{i,j} \subseteq M$ for all j. Suppose that the clause C_ℓ is satisfied by its tth literal (if there are more than one true literal, choose one arbitrarily). Then we let $M^t_\ell \subseteq M$. We show that M is jointly stable.

▶ **Lemma 2.** *The matching M constructed as above is jointly stable.*

Proof. Consider literal people corresponding to x_i, namely $a_{i,j}$, $b_{i,j}$, $c_{i,j}$, and $d_{i,j}$ $(1 \leq j \leq s_i)$. If $T(x_i) = 1$, then all the men are matched with their first choices in both L_1 and L_2. Similarly, if $T(x_i) = 0$, then all the women are matched with their first choices. Therefore, no blocking pair arises within literal people corresponding to the same variable. Since literal people corresponding to different variables are unacceptable to each other, no blocking pair occurs between them.

As for the 18 people corresponding to the clause C_ℓ, we can easily verify that, in any of M^1_ℓ, M^2_ℓ, and M^3_ℓ, no blocking pair arises among them. Also, since clause people corresponding to different clauses are unacceptable to each other, no blocking pair occurs between them.

Finally, we consider a possibility of a blocking pair between a literal person and a clause person. Consider the clause C_ℓ. First, suppose that M^1_ℓ is chosen as a part of M. By construction of M, this means that the clause C_ℓ is satisfied by its first literal. Suppose that this literal is the jth occurrence of x_i, and that it is a positive literal. Then by construction of preference lists, $D_{\ell,1}$ is null and $B_{\ell,1} = b_{i,j}$, so only the possible blocking pair is $(b_{i,j}, v_{\ell,4})$ in L_1. However, since C_ℓ is satisfied by the first literal, it must be the case that $T(x_i) = 1$. By construction of M, $M^1_{i,j} \subseteq M$ and hence $b_{i,j}$ is matched with his first choice woman in L_1, so he cannot form a blocking pair. Now suppose that the first literal of C_ℓ is the jth occurrence of x_i and it is a negative literal. Then $B_{\ell,1}$ is null and $D_{\ell,1} = d_{i,j}$, so, only the possible blocking pair is $(u_{\ell,1}, d_{i,j})$ in L_1. However, since C_ℓ is satisfied by the first literal, we have that $T(x_i) = 0$ and hence $d_{i,j}$ is matched with her first choice man in L_1, so $d_{i,j}$ cannot form a blocking pair. For the other two cases, that is, the case that M^2_ℓ is chosen and M^3_ℓ is chosen, we can show that there is no blocking pair by a similar argument. ◀

Conversely, suppose that I admits a jointly stable matching M. We construct a satisfying assignment T of f. First, we see basic properties of M.

▶ **Lemma 3.** *For each i, either $M^1_{i,j} \subseteq M$ for all j, or $M^0_{i,j} \subseteq M$ for all j.*

Proof. We first show that, for each i and j, either $M^1_{i,j} \subseteq M$ or $M^0_{i,j} \subseteq M$. Suppose not. Since $c_{i,j}$ and $d_{i,j}$ are the only acceptable men to $a_{i,j}$ and $b_{i,j}$ in L_1 and L_2 in common, at least one of $a_{i,j}$ and $b_{i,j}$, say $m_{i,j}$, is single in M. For the same reason, at least one of $c_{i,j}$ and $d_{i,j}$, say $w_{i,j}$, is single in M. Then $(m_{i,j}, w_{i,j})$ blocks M (in both L_1 and L_2), a contradiction.

Now suppose that the statement of the lemma is false. Then there are i and j $(1 \leq j \leq s_i - 1)$ such that (i) $M^1_{i,j} \subseteq M$ and $M^0_{i,j+1} \subseteq M$ or (ii) $M^0_{i,j} \subseteq M$ and $M^1_{i,j+1} \subseteq M$. In case of (i), $(a_{i,j+1}, c_{i,j})$ blocks M in L_2, while in case of (ii), $(b_{i,j}, d_{i,j+1})$ blocks M in L_2, a contradiction. ◀

▶ **Lemma 4.** *For each ℓ, either $M^1_\ell \subseteq M$, $M^2_\ell \subseteq M$, or $M^3_\ell \subseteq M$.*

Proof. Suppose that there is a man $m_\ell \in \{u^1_\ell, u^2_\ell, u^3_\ell\}$ who is not matched with any of v^1_ℓ, v^2_ℓ, and v^3_ℓ in M. Note that $D_{\ell,1}$ is a literal woman (if not null), who is not acceptable to u^1_ℓ in L_2. Hence it must be the case that m_ℓ is single in M. By a similar argument, there

Table 1 27 matchings and corresponding blocking pairs in L_2.

Matching	BP		Matching	BP		Matching	BP
$X_\ell^1 \cup Y_\ell^1 \cup Z_\ell^1$	(u_ℓ^7, v_ℓ^1)		$X_\ell^2 \cup Y_\ell^1 \cup Z_\ell^1$	(u_ℓ^5, v_ℓ^8)		$X_\ell^3 \cup Y_\ell^1 \cup Z_\ell^1$	(u_ℓ^5, v_ℓ^8)
$X_\ell^1 \cup Y_\ell^1 \cup Z_\ell^2$	(u_ℓ^7, v_ℓ^1)		$X_\ell^2 \cup Y_\ell^1 \cup Z_\ell^2$	(u_ℓ^8, v_ℓ^2)		$X_\ell^3 \cup Y_\ell^1 \cup Z_\ell^2$	–
$X_\ell^1 \cup Y_\ell^1 \cup Z_\ell^3$	(u_ℓ^4, v_ℓ^7)		$X_\ell^2 \cup Y_\ell^1 \cup Z_\ell^3$	(u_ℓ^5, v_ℓ^8)		$X_\ell^3 \cup Y_\ell^1 \cup Z_\ell^3$	(u_ℓ^5, v_ℓ^8)
$X_\ell^1 \cup Y_\ell^2 \cup Z_\ell^1$	(u_ℓ^7, v_ℓ^1)		$X_\ell^2 \cup Y_\ell^2 \cup Z_\ell^1$	(u_ℓ^1, v_ℓ^4)		$X_\ell^3 \cup Y_\ell^2 \cup Z_\ell^1$	(u_ℓ^1, v_ℓ^4)
$X_\ell^1 \cup Y_\ell^2 \cup Z_\ell^2$	(u_ℓ^7, v_ℓ^1)		$X_\ell^2 \cup Y_\ell^2 \cup Z_\ell^2$	(u_ℓ^1, v_ℓ^4)		$X_\ell^3 \cup Y_\ell^2 \cup Z_\ell^2$	(u_ℓ^1, v_ℓ^4)
$X_\ell^1 \cup Y_\ell^2 \cup Z_\ell^3$	–		$X_\ell^2 \cup Y_\ell^2 \cup Z_\ell^3$	(u_ℓ^1, v_ℓ^4)		$X_\ell^3 \cup Y_\ell^2 \cup Z_\ell^3$	(u_ℓ^1, v_ℓ^4)
$X_\ell^1 \cup Y_\ell^3 \cup Z_\ell^1$	(u_ℓ^7, v_ℓ^1)		$X_\ell^2 \cup Y_\ell^3 \cup Z_\ell^1$	–		$X_\ell^3 \cup Y_\ell^3 \cup Z_\ell^1$	(u_ℓ^2, v_ℓ^5)
$X_\ell^1 \cup Y_\ell^3 \cup Z_\ell^2$	(u_ℓ^7, v_ℓ^1)		$X_\ell^2 \cup Y_\ell^3 \cup Z_\ell^2$	(u_ℓ^8, v_ℓ^2)		$X_\ell^3 \cup Y_\ell^3 \cup Z_\ell^2$	(u_ℓ^2, v_ℓ^5)
$X_\ell^1 \cup Y_\ell^3 \cup Z_\ell^3$	(u_ℓ^4, v_ℓ^7)		$X_\ell^2 \cup Y_\ell^3 \cup Z_\ell^3$	(u_ℓ^4, v_ℓ^7)		$X_\ell^3 \cup Y_\ell^3 \cup Z_\ell^3$	(u_ℓ^2, v_ℓ^5)

is a woman $w_\ell \in \{v_\ell^1, v_\ell^2, v_\ell^3\}$ who is single in M. Then (m_ℓ, w_ℓ) blocks M in L_1 and L_2, a contradiction. Therefore, u_ℓ^1, u_ℓ^2, and u_ℓ^3 are matched with v_ℓ^1, v_ℓ^2, and v_ℓ^3 in M. There are six possible ways, namely,

$$X_\ell^1 = \{(u_\ell^1, v_\ell^1), (u_\ell^2, v_\ell^2), (u_\ell^3, v_\ell^3)\},\ X_\ell^2 = \{(u_\ell^1, v_\ell^2), (u_\ell^2, v_\ell^3), (u_\ell^3, v_\ell^1)\},$$
$$X_\ell^3 = \{(u_\ell^1, v_\ell^3), (u_\ell^2, v_\ell^1), (u_\ell^3, v_\ell^2)\},\ X_\ell^4 = \{(u_\ell^1, v_\ell^1), (u_\ell^2, v_\ell^3), (u_\ell^3, v_\ell^2)\},$$
$$X_\ell^5 = \{(u_\ell^1, v_\ell^2), (u_\ell^2, v_\ell^1), (u_\ell^3, v_\ell^3)\}, \text{ and } X_\ell^6 = \{(u_\ell^1, v_\ell^3), (u_\ell^2, v_\ell^2), (u_\ell^3, v_\ell^1)\}.$$

It is easy to see that X_ℓ^4 is blocked by (u_ℓ^3, v_ℓ^1), X_ℓ^5 is blocked by (u_ℓ^2, v_ℓ^3), and X_ℓ^6 is blocked by (u_ℓ^1, v_ℓ^2) in L_1. Therefore, only X_ℓ^1, X_ℓ^2, and X_ℓ^3 can be a part of M. The same argument applies to u_ℓ^4, u_ℓ^5, u_ℓ^6, v_ℓ^4, v_ℓ^5, v_ℓ^6 and u_ℓ^7, u_ℓ^8, u_ℓ^9, v_ℓ^7, v_ℓ^8, v_ℓ^9, implying that only

$$Y_\ell^1 = \{(u_\ell^4, v_\ell^4), (u_\ell^5, v_\ell^5), (u_\ell^6, v_\ell^6)\},\ Y_\ell^2 = \{(u_\ell^4, v_\ell^5), (u_\ell^5, v_\ell^6), (u_\ell^6, v_\ell^4)\},$$

$$Y_\ell^3 = \{(u_\ell^4, v_\ell^6), (u_\ell^5, v_\ell^4), (u_\ell^6, v_\ell^5)\},$$

and

$$Z_\ell^1 = \{(u_\ell^7, v_\ell^7), (u_\ell^8, v_\ell^8), (u_\ell^9, v_\ell^9)\},\ Z_\ell^2 = \{(u_\ell^7, v_\ell^8), (u_\ell^8, v_\ell^9), (u_\ell^9, v_\ell^7)\},$$

$$Z_\ell^3 = \{(u_\ell^7, v_\ell^9), (u_\ell^8, v_\ell^7), (u_\ell^9, v_\ell^8)\}$$

are valid.

Therefore, there are 27 possible combinations. Note that $M_\ell^1 = X_\ell^3 \cup Y_\ell^1 \cup Z_\ell^2$, $M_\ell^2 = X_\ell^2 \cup Y_\ell^3 \cup Z_\ell^1$, and $M_\ell^3 = X_\ell^1 \cup Y_\ell^2 \cup Z_\ell^3$. We show that the remaining 24 matchings are unstable in L_2. Table 1 shows 27 matchings in "Matching" columns and corresponding blocking pairs of 24 matchings in "BP" columns. This completes the proof. ◀

By Lemma 3, either $M_{i,j}^1 \subseteq M$ for all j or $M_{i,j}^0 \subseteq M$ for all j holds. In the former case, we set $T(x_i) = 1$, otherwise, we set $T(x_i) = 0$. We show that T satisfies f. Suppose not, and let C_ℓ be an unsatisfied clause. For $t = 1, 2, 3$, let the tth literal of C_ℓ be the j_tth occurrence of the variable x_{i_t}. We will show three claims:

Claim 1. $\boldsymbol{M_\ell^1 \not\subseteq M}$**.** Consider the first literal of C_ℓ. Suppose that it appears positively in C_ℓ. Then by construction of the preference lists, the lists of b_{i_1,j_1} and v_ℓ^4 in L_1 are as follows:

$$b_{i_1,j_1} : \ d_{i_1,j_1} \ \ v_\ell^4 \ \ c_{i_1,j_1} \qquad\qquad v_\ell^4 : \ u_\ell^5 \ \ u_\ell^6 \ \ b_{i_1,j_1} \ \ u_\ell^4$$

Since C_ℓ is unsatisfied, $T(x_{i_1}) = 0$ and so by construction of T, $M^0_{i_1,j_1} \subseteq M$, i.e., $M(b_{i_1,j_1}) = c_{i_1,j_1}$. If $M_\ell^1 \subseteq M$, then $M(v_\ell^4) = u_\ell^4$ and hence (b_{i_1,j_1}, v_ℓ^4) blocks M in L_1, a contradiction.

Next, suppose that the first literal of C_ℓ is negative, i.e., $\overline{x_{i_1}}$. Then by construction, the preference lists of d_{i_1,j_1} and u_ℓ^1 in L_1 are as follows:

$$u_\ell^1 : \ v_\ell^1 \ \ v_\ell^2 \ \ d_{i_1,j_1} \ \ v_\ell^3 \qquad\qquad d_{i_1,j_1} : \ a_{i_1,j_1} \ \ u_\ell^1 \ \ b_{i_1,j_1}$$

Since C_ℓ is unsatisfied, $T(x_{i_1}) = 1$ and so by construction of T, $M^1_{i_1,j_1} \subseteq M$, i.e., $M(d_{i_1,j_1}) = b_{i_1,j_1}$. If $M_\ell^1 \subseteq M$, then $M(u_\ell^1) = v_\ell^3$ and hence (u_ℓ^1, d_{i_1,j_1}) blocks M in L_1, a contradiction. Therefore, we can conclude that $M_\ell^1 \not\subseteq M$.

Claim 2. $\boldsymbol{M_\ell^2 \not\subseteq M}$**.** Consider the second literal of C_ℓ, and first suppose that it is a positive literal, i.e., x_{i_2}. Then by construction, the preference lists of b_{i_2,j_2} and v_ℓ^7 in L_1 are as follows:

$$b_{i_2,j_2} : \ d_{i_2,j_2} \ \ v_\ell^7 \ \ c_{i_2,j_2} \qquad\qquad v_\ell^7 : \ u_\ell^8 \ \ u_\ell^9 \ \ b_{i_2,j_2} \ \ u_\ell^7$$

Since C_ℓ is unsatisfied, $T(x_{i_2}) = 0$ and hence by construction of T, $M^0_{i_2,j_2} \subseteq M$, i.e., $M(b_{i_2,j_2}) = c_{i_2,j_2}$. If $M_\ell^2 \subseteq M$, then $M(v_\ell^7) = u_\ell^7$ and hence (b_{i_2,j_2}, v_ℓ^7) blocks M in L_1, a contradiction.

Next, suppose that the second literal of C_ℓ is $\overline{x_{i_2}}$. Then by construction, the preference lists of d_{i_2,j_2} and u_ℓ^4 in L_1 are as follows:

$$u_\ell^4 : \ v_\ell^4 \ \ v_\ell^5 \ \ d_{i_2,j_2} \ \ v_\ell^6 \qquad\qquad d_{i_2,j_2} : \ a_{i_2,j_2} \ \ u_\ell^4 \ \ b_{i_2,j_2}$$

Since C_ℓ is unsatisfied, $T(x_{i_2}) = 1$ and by construction of T, $M^1_{i_2,j_2} \subseteq M$, i.e., $M(d_{i_2,j_2}) = b_{i_2,j_2}$. If $M_\ell^2 \subseteq M$, then $M(u_\ell^4) = v_\ell^6$ and hence (u_ℓ^4, d_{i_2,j_2}) blocks M in L_1, a contradiction. Therefore, we can conclude that $M_\ell^2 \not\subseteq M$.

Claim 3. $\boldsymbol{M_\ell^3 \not\subseteq M}$**.** Consider the third literal of C_ℓ. First, suppose that it is a positive literal x_{i_3}. Then by construction, the preference lists of b_{i_3,j_3} and v_ℓ^1 in L_1 are as follows:

$$b_{i_3,j_3} : \ d_{i_3,j_3} \ \ v_\ell^1 \ \ c_{i_3,j_3} \qquad\qquad v_\ell^1 : \ u_\ell^2 \ \ u_\ell^3 \ \ b_{i_3,j_3} \ \ u_\ell^1$$

Since C_ℓ is unsatisfied, $T(x_{i_3}) = 0$ and thus by construction of T, $M^0_{i_3,j_3} \subseteq M$, i.e., $M(b_{i_3,j_3}) = c_{i_3,j_3}$. If $M_\ell^3 \subseteq M$, then $M(v_\ell^1) = u_\ell^1$ and hence (b_{i_3,j_3}, v_ℓ^1) blocks M in L_1, a contradiction.

Next, suppose that the third literal of C_ℓ is negative, i.e., $\overline{x_{i_3}}$. Then by construction, the preference lists of d_{i_3,j_3} and u_ℓ^7 in L_1 are as follows:

$$u_\ell^7 : \ v_\ell^7 \ \ v_\ell^8 \ \ d_{i_3,j_3} \ \ v_\ell^9 \qquad\qquad d_{i_3,j_3} : \ a_{i_3,j_3} \ \ u_\ell^7 \ \ b_{i_3,j_3}$$

Since C_ℓ is unsatisfied, $T(x_{i_3}) = 1$ and by construction of T, $M^1_{i_3,j_3} \subseteq M$, i.e., $M(d_{i_3,j_3}) = b_{i_3,j_3}$. If $M_\ell^3 \subseteq M$, then $M(u_\ell^7) = v_\ell^9$ and hence (u_ℓ^7, d_{i_3,j_3}) blocks M in L_1, a contradiction. Therefore, we can conclude that $M_\ell^3 \not\subseteq M$.

From Claims 1, 2, and 3, none of M_ℓ^1, M_ℓ^2, and M_ℓ^3 can be a part of M, but this contradicts Lemma 4. Hence we conclude that T satisfies f, which completes the proof of Theorem 1. ◀

In the above reduction, we have exploited existence of pairs that are acceptable in L_1 but not in L_2, or vice versa. Then one may be curious about whether SMkI is solvable in polynomial time if the set of acceptable pairs is the same in all L_i. However, this is unlikely, as shown in the following corollary. Let SMk denote the special case of SMkI where all the preference lists are complete. Clearly SMk satisfies the above mentioned condition.

▶ **Corollary 5.** *For $k \geq 2$, SMk is NP-complete.*

Proof. Apparently SM$k \in$ NP. For the NP-hardness, in the reduction given in the proof of Theorem 1, make every preference list complete by appending missing persons to the tail of the list in an arbitrary order. It is not hard to see that the same correctness proof (with slight modifications) applies. ◀

3 Tractable Cases

In this section, we assume without loss of generality that acceptability is mutual, i.e., m is acceptable to w in L_i if and only if w is acceptable to m in L_i. This is because, if for example m is acceptable to w while w is not acceptable to m, then (m, w) can neither be a part of a matching nor a blocking pair. Hence we may remove m from w's list safely, without changing the set of jointly stable matchings. This preprocessing can be done in time linear in the total length of the input preference lists.

However, even if (m, w) is an acceptable pair in L_i but is an unacceptable pair in L_j $(j \neq i)$, we must not remove m and w from each other's list in L_i. This is because, although (m, w) cannot be a pair in a jointly stable matching, it may block some matching in L_i and removing it may change the set of jointly stable matchings.

3.1 Length–Two Preferences Lists of One Side

Our first positive result is for instances in which the length of preference lists of one side, say men's side, is bounded by two. The proof of Theorem 6 exploits a partially-ordered set (poset) of rotations and its relation to the whole set of stable matchings. These structural properties were originally studied for complete preference lists, but they can be extended easily and naturally to incomplete preference lists. Here we give brief explanations about them. See [5] for more detail. Readers who are familiar with these notions may skip the following two paragraphs.

Let I be an instance of SMI and M be a stable matching for I. For a man m matched in M, $s_M(m)$ denotes the first woman w in m's list such that w is matched in M and w prefers m to $M(w)$. Note that m prefers $M(m)$ to $s_M(m)$; otherwise, $(m, s_M(m))$ blocks M. Also, $next_M(m)$ denotes the partner of $s_M(m)$ in M, that is, $next_M(m) = M(s_M(m))$. Let $\rho = (m_0, w_0), (m_1, w_1), \ldots, (m_{r-1}, w_{r-1})$ $(r \geq 2)$ be a sequence of pairs such that each pair in ρ is contained in M and $m_{i+1} = next_M(m_i)$ for each i, where $i + 1$ is taken modulo r. Then we call ρ a *rotation exposed in* M. By *eliminating* a rotation ρ from M, we mean to replace pairs $(m_0, w_0), (m_1, w_1), \ldots, (m_{r-1}, w_{r-1})$ by $(m_0, w_1), (m_1, w_2), \ldots, (m_{r-1}, w_0)$ in M. The resulting matching, denoted by M/ρ, is also stable in I. Note that each man included in ρ has a worse partner in M/ρ than in M.

Let Π be the set of rotations that are exposed in one or more stable matchings for I. We can define a partial order $\preceq$ on Π, and $(\Pi, \preceq)$ is called the *rotation poset* of I. A subset $P \subseteq \Pi$ is called a *closed subset* of Π if $\rho \in P$ and $\rho' \preceq \rho$ then $\rho' \in P$. There is a one-to-one correspondence between the stable matchings for I and the closed subsets of Π by the mapping defined as follows. Let M_0 be the *man-optimal stable matching* of I (which is guaranteed to exist and can be found by the *men-oriented Gale-Shapley algorithm* in time linear in the total length of preference lists). Let P be a closed subset of Π. If we eliminate rotations in P one by one according to the order $\preceq$, we obtain a stable matching for I. Conversely, any stable matching for I is obtained by this procedure for some closed subset of Π. In particular, the empty set corresponds to the man-optimal stable matching and the whole set Π corresponds to the *woman-optimal stable matching* (which is the opposite extreme to the man-optimal stable matching). The rotation poset can be constructed in time linear in the total length of preference lists (Sec. 3.3 of [5]).

▶ **Theorem 6.** *$(2, \infty)$-SMkI is solvable in time $O(kn)$.*

Proof. We first compute the man-optimal stable matchings M_i for L_i $(i = 1, 2, \ldots, k)$ using the men-oriented version of the Gale-Shapley algorithm. For each L_i, any stable matching leaves the same set of men and women unmatched [4]. Thus if there are i and j $(i \neq j)$ such that the set of matched people in M_i and that in M_j are different, then we can immediately answer "no". In the following, we assume that the sets of matched people are the same in all M_i.

For each i, we compute all the rotations $\rho^i_1, \rho^i_2, \ldots, \rho^i_{n_i}$ with respect to L_i. Since the length of each man's preference list is at most two, each man is contained in at most one rotation. This means that all the rotations are mutually incomparable in the rotation poset. Hence there is a one-to-one correspondence between the set of stable matchings for L_i and the power set of $\{\rho^i_1, \rho^i_2, \ldots, \rho^i_{n_i}\}$: the subset $S \subseteq \{\rho^i_1, \rho^i_2, \ldots, \rho^i_{n_i}\}$ corresponds to the stable matching $M_{i,S}$ obtained by eliminating all the rotations in S from M_i. Consider a man m who is matched in M_i. If m is not included in a rotation, his partner is the same in all the stable matchings of L_i. If he is included in a rotation ρ^i_j, then he is matched in $M_{i,S}$ with his first choice if $\rho^i_j \notin S$ and with his second choice if $\rho^i_j \in S$.

The remaining task is to check if there are k subsets $S_i \subseteq \{\rho^i_1, \rho^i_2, \ldots, \rho^i_{n_i}\}$ $(1 \leq i \leq k)$ such that $M_{1,S_1} = M_{2,S_2} = \cdots = M_{k,S_k}$. For this purpose, we introduce a binary variable x^i_j for ρ^i_j $(1 \leq i \leq k,\ 1 \leq j \leq n_i)$, where $x^i_j = 1$ means to put ρ^i_j in S_i. We then construct a 2CNF SAT instance as follows.

For each man m who is matched in M_1 (and equivalently in all M_i), we fix the value of variables or construct 2CNF clauses to ensure that m's partners coincide in all $M_{1,S_1}, M_{2,S_2}, \ldots, M_{k,S_k}$. If (m, w) is a pair in some stable matching of L, w is called m's *stable partner in L*. Also, if w is m's stable partner in all L_i, w is called m's *jointly stable partner*. If m has no jointly stable partner, we immediately output "no". If m has one jointly stable partner w, then for each i, we enforce the variable (if any) to match m with w in M_{i,S_i}. Namely, if m is not included in a rotation, then there is no variable and we do nothing. If m is included in a rotation ρ^i_j and w is his first (second) choice in L_i, then we set $x^i_j = 0$ $(x^i_j = 1)$. During this course, if some variable is fixed differently, then we immediately output "no". Finally, suppose that m has two jointly stable partners w' and w''. This means that for each i, $L_i(m)$ contains both w' and w'' and m is included in a rotation of L_i. Let $\rho^i_{j_i}$ be the rotation that includes m. For $i = 2, \ldots, k$, we construct two clauses as follows: If the order of w' and w'' is same in $L_1(m)$ and $L_i(m)$, then we construct $(x^1_{j_1} \vee \overline{x^i_{j_i}})$ and $(\overline{x^1_{j_1}} \vee x^i_{j_i})$; otherwise, we construct $(x^1_{j_1} \vee x^i_{j_i})$ and $(\overline{x^1_{j_1}} \vee \overline{x^i_{j_i}})$. The construction of 2CNF formula is completed by doing this for all the men m who are matched in M_1. It is not hard to see that a satisfying assignment corresponds to subsets S_i such that $M_{1,S_1} = M_{2,S_2} = \cdots = M_{k,S_k}$.

Recall that men's preference lists are of length at most two and acceptability is mutual by assumption, so the total lengths of L_i is $O(n)$. Therefore, for each i, finding M_i and computing the set of rotations of L_i can be done in $O(n)$ time, and hence in $O(kn)$ time in total. Constructing 2CNF clauses for each man can be done in time $O(k)$, and therefore $O(kn)$ for at most n men. The resulting 2CNF formula has size $O(kn)$. Finally, solving 2CNF satisfiability problem can be done in linear time [2, 1]. Thus overall time-complexity is $O(kn)$. ◀

3.2 Identical Preference Lists of One Side

The next polynomial-time solvable case is that each woman's preference lists are identical in all L_i. It should be noted that this condition is different from the so-called *master lists*, in which all the women have the same preference list. In our case, w and w' may have different preference lists.

▶ **Theorem 7.** *If each woman's preference lists in all L_i ($1 \leq i \leq k$) are identical, SMkI is solvable in time $O(N)$, where N is the total length of preference lists in an input.*

Proof. Let $I = (U, W, L_1, L_2, \ldots, L_k)$ be an instance of SMkI. We first note that, since $L_1(w) = L_2(w) = \cdots = L_k(w)$ for every woman w, for each man m the sets of women included in $L_i(m)$ are the same for all i, due to the mutual-acceptability assumption made at the beginning of this section. Now we construct a set L of preference lists from $L_1, L_2, \ldots, L_k$ as follows: For each woman w, let $L(w) := L_1(w)$. For each man m, the set of women included in $L(m)$ is the same as in $L_i(m)$, and their order is defined as follows. Let w' and w'' be women in $L(m)$. If m prefers w' to w'' in all $L_i(m)$, then m prefers w' to w'' in $L(m)$. If m prefers w' to w'' in some $L_i(m)$ and w'' to w' in some $L_j(m)$, then m is indifferent between w' and w'' in $L(m)$. It is not hard to see that $L(m)$ is a partially-ordered list and hence $I' = (U, W, L)$ can be regarded as an instance of the *Stable Marriage problem with Partially-ordered and Incomplete lists* (*SMPI*).

We now recall the *super-stability* [5, 6] in the case that preference lists are not necessarily in a total order. For a matching M, (m, w) is a *blocking pair in super-stability* if (1) $(m, w) \notin M$ but m and w are acceptable to each other, (2) m is single in M, or prefers w to $M(m)$, or is indifferent between w and $M(m)$, and (3) w is single in M, or prefers m to $M(w)$, or is indifferent between m and $M(w)$. We say that a matching is *super-stable* if it admits no blocking pair in super-stability. Irving [6] developed an $O(n^2)$-time algorithm to find a super-stable matching or to report that no super-stable matching exists when preference lists are complete and may include ties. Manlove [8] extended this algorithm for incomplete preference lists, and showed that it runs in time $O(N)$ where N is the total length of preference lists in an input. Also, Manlove showed that the same algorithm is applicable for partially-ordered preference lists, i.e., SMPI (page 169 of [10]). Therefore, to complete the proof, it suffices to show that a matching M is jointly stable in I if and only if M is super-stable in I'.

First suppose that M is not a jointly stable matching of I and hence has a blocking pair (m, w) in L_i for some i. Then w is single in M or prefers m to $M(w)$ in $L_i(w)$. In the latter case, w prefers m to $M(w)$ also in $L(w)$. Similarly, m is single in M or prefers w to $M(w)$ in $L_i(m)$. In the latter case, m prefers w to $M(m)$ or is indifferent between them in $L(m)$. Thus (m, w) is a blocking pair in super-stability for M and therefore M is not super-stable in I'.

Conversely, suppose that M is not super-stable in I'. Then, there is a blocking pair (m, w) in super-stability. Since $L(w)$ is a total order, w is unmatched in M or prefers m to $M(w)$ in $L(w)$. In the latter case, w prefers m to $M(w)$ in all $L_i(w)$. Note that m either (i) is unmatched in M, or (ii) prefers w to $M(m)$ in $L(m)$, or (iii) is indifferent between w and $M(m)$ in $L(m)$. In the case of (i), (m, w) is a blocking pair for M in all L_i. In the case of (ii), m prefers w to $M(m)$ in all $L_i(m)$, so again (m, w) is a blocking pair for M in all L_i. In the case of (iii), m prefers w to $M(m)$ in $L_i(m)$ for some i, so that (m, w) is a blocking pair for M in L_i. In any case, M in not jointly stable in I.

Constructing I' from I and solving I' can both be done in $O(N)$ time, hence the theorem follows. ◀

As a byproduct of the above proof, we can show the existence of the man-optimal and woman-optimal stable matchings. Let us call a jointly stable matching M *man-optimal* if for any man m and any jointly stable matching M', either $M(m) = M'(m)$ or m prefers $M(m)$ to $M'(m)$ in all L_i. The *woman-optimal* jointly stable matching is defined similarly.

Let $I = (U, W, L_1, L_2, \ldots, L_k)$ be an SMkI instance and $I' = (U, W, L)$ be the SMPI instance constructed as in the above proof. It is known that the set of super-stable matchings for an SMPI instance form a distributive lattice ([12, 9] and page 169 of [10]), so there are the man-optimal and the woman-optimal stable matchings for I', denoted M_U and M_W, respectively. Since women's preference lists are the same in L and all L_i, M_W is the woman-optimal jointly stable matching for I. Consider a man m and suppose that m is indifferent between w_1 and w_2 in $L(m)$. It is known that it cannot be the case that m is matched with w_1 in one super-stable matching and with w_2 in another super-stable matching. Thus by the man-optimality of M_U, for every man m, either $M_U(m) = M(m)$ or m prefers $M_U(m)$ to $M(m)$ in $L(m)$ for any super-stable matching M. This implies that by construction of L, either $M_U(m) = M(m)$ or m prefers $M_U(m)$ to $M(m)$ in $L_i(m)$ for all i, implying the existence of the man-optimal jointly stable matching.

4 Conclusion

In this paper, we considered a variant of the stable marriage problem in which we are given k sets of preference lists $L_1, L_2, \ldots, L_k$, and are asked to determine the existence of a matching that is stable with respect to all L_i ($1 \leq i \leq k$). We have shown that the problem is NP-complete for $k \geq 2$ even if all the preference lists are of length at most four, while it is solvable in linear time if each man's preference list is of length at most two. We also showed that the problem is solvable in linear time if every woman has an identical preference list in all L_i.

An important future work is to determine the complexity of the problem when the lengths of preference lists are bounded by three, namely, $(3, \ell)$-SMkI for $\ell \geq 3$. Another direction is approximability of SMkI; given an instance, find a matching that is stable in as many L_i as possible. Finding a stable matching in any one list is a trivial k-approximation algorithm. On the other hand, using Theorem 1 we can easily deduce an approximation hardness of $2 - \epsilon$ for even k and $2 - \frac{2}{k+1} - \epsilon$ for odd k, for any positive constant ϵ under P≠NP. Narrowing this gap is an interesting future work. Considering an alternative optimization criteria, e.g., minimizing the total number of blocking pairs over all L_i, would also be attractive.

References

1 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

2 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976. doi:10.1137/0205048.

3 David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. URL: http://www.jstor.org/stable/2312726.

4 David Gale and Marilda Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11(3):223–232, 1985. doi:10.1016/0166-218X(85)90074-5.

5 Dan Gusfield and Robert W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.

6 Robert W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48(3):261–272, 1994. doi:10.1016/0166-218X(92)00179-P.

7 Robert W. Irving and Paul Leather. The complexity of counting stable marriages. *SIAM J. Comput.*, 15(3):655–667, 1986. doi:10.1137/0215048.

8 David F. Manlove. Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, 1999.

9 David F. Manlove. The structure of stable marriage with indifference. *Discrete Applied Mathematics*, 122(1-3):167–181, 2002. doi:10.1016/S0166-218X(01)00322-5.

10 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013. doi:10.1142/8591.

11 Jay Sethuraman and Chung-Piaw Teo. A polynomial-time algorithm for the bistable roommates problem. *J. Comput. Syst. Sci.*, 63(3):486–497, 2001. doi:10.1006/jcss.2001.1791.

12 Boris Spieker. The set of super-stable marriages forms a distributive lattice. *Discrete Applied Mathematics*, 58(1):79–84, 1995. doi:10.1016/0166-218X(94)00080-W.

13 Edward G. Thurber. Concerning the maximum number of stable matchings in the stable marriage problem. *Discrete Mathematics*, 248(1-3):195–219, 2002. doi:10.1016/S0012-365X(01)00194-7.

14 Bob P. Weems. Bistable versions of the marriages and roommates problems. *J. Comput. Syst. Sci.*, 59(3):504–520, 1999. doi:10.1006/jcss.1999.1657.

Fast Compressed Self-Indexes with Deterministic Linear-Time Construction[*][†]

J. Ian Munro[1], Gonzalo Navarro[2], and Yakov Nekrich[3]

1 **Cheriton School of Computer Science, University of Waterloo, Canada**
imunro@uwaterloo.ca

2 **CeBiB — Center of Biotechnology and Bioengineering, Department of Computer Science, University of Chile, Chile**
gnavarro@dcc.uchile.cl

3 **Cheriton School of Computer Science, University of Waterloo, Canada**
yakov.nekrich@googlemail.com

Abstract

We introduce a compressed suffix array representation that, on a text T of length n over an alphabet of size σ, can be built in $O(n)$ deterministic time, within $O(n \log \sigma)$ bits of working space, and counts the number of occurrences of any pattern P in T in time $O(|P| + \log\log_w \sigma)$ on a RAM machine of $w = \Omega(\log n)$-bit words. This new index outperforms all the other compressed indexes that can be built in linear deterministic time, and some others. The only faster indexes can be built in linear time only in expectation, or require $\Theta(n \log n)$ bits.

1998 ACM Subject Classification E.1 Data Structures, E.4 Coding and Information Theory

Keywords and phrases Succinct data structures, Self-indexes, Suffix arrays, Deterministic construction

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.57

1 Introduction

The string indexing problem consists in preprocessing a string T so that, later, we can efficiently find occurrences of patterns P in T. The most popular solutions to this problem are suffix trees [29] and suffix arrays [21]. Both can be built in $O(n)$ deterministic time on a text T of length n over an alphabet of size σ, and the best variants can count the number of times a string P appears in T in time $O(|P|)$, and even in time $O(|P|/\log_\sigma n)$ in the word-RAM model if P is given packed into $|P|/\log_\sigma n$ words [26]. Once counted, each occurrence can be located in $O(1)$ time. Those optimal times, however, come with two important drawbacks:

- The variants with this counting time cannot be built in $O(n)$ worst-case time.
- The data structures use $\Theta(n \log n)$ bits of space.

The reason of the first drawback is that some form of perfect hashing is always used to ensure constant time per pattern symbol (or pack of symbols). The classical suffix trees and arrays with linear-time deterministic construction offer $O(|P| \log \sigma)$ or $O(|P| + \log n)$ counting time, respectively. More recently, those times have been reduced to $O(|P| + \log \sigma)$ [10] and even to $O(|P| + \log\log \sigma)$ [14]. Simultaneously with our work, a suffix tree variant

[*] Funded with Basal Funds FB0001, Conicyt, Chile.

[†] The full version of this article is available at [23], https://arxiv.org/abs/1707.01743.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 57; pp. 57:1–57:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

was introduced by Bille et al. [7], which can be built in linear deterministic time and counts in time $O(|P|/\log_\sigma n + \log|P| + \log\log\sigma)$. All those indexes, however, still suffer from the second drawback, that is, they use $\Theta(n\log n)$ bits of space. This makes them impractical in most applications that handle large text collections.

Research on the second drawback dates back to almost two decades [25], and has led to indexes using $nH_k(T) + o(n(H_k(T)+1))$ bits, where $H_k(T) \leq \log\sigma$ is the k-th order entropy of T [22], for any $k \leq \alpha\log_\sigma n - 1$ and any constant $0 < \alpha < 1$. That is, the indexes use asymptotically the same space of the compressed text, and can reproduce the text and search it; thus they are called self-indexes. The fastest compressed self-indexes that can be built in linear deterministic time are able to count in time $O(|P|\log\log\sigma)$ [1] or $O(|P|(1+\log_w\sigma))$ [6]. There exist other compressed self-indexes that obtain times $O(|P|)$ [5] or $O(|P|/\log_\sigma n + \log_\sigma^\epsilon n)$ for any constant $\epsilon > 0$ [18], but both rely on perfect hashing and are not built in linear deterministic time. All those compressed self-indexes use $O(n\frac{\log n}{b})$ further bits to locate the position of each occurrence found in time $O(b)$, and to extract any substring S of T in time $O(|S|+b)$.

In this paper we introduce the first compressed self-index that can be built in $O(n)$ deterministic time (moreover, using $O(n\log\sigma)$ bits of space [24]) and with counting time $O(|P| + \log\log_w\sigma)$, where $w = \Omega(\log n)$ is the size in bits of the computer word. More precisely, we prove the following result.

► **Theorem 1.** *On a RAM machine of $w = \Omega(\log n)$ bits, we can construct an index for a text T of length n over an alphabet of size $\sigma = O(n/\log n)$ in $O(n)$ deterministic time using $O(n\log\sigma)$ bits of working space. This index occupies $nH_k(T) + o(n\log\sigma) + O(n\frac{\log n}{b})$ bits of space for a parameter b and any $k \leq \alpha\log_\sigma n - 1$, for any constant $0 < \alpha < 1$. The occurrences of a pattern string P can be counted in $O(|P| + \log\log_w\sigma)$ time, and then each such occurrence can be located in $O(b)$ time. An arbitrary substring S of T can be extracted in time $O(|S|+b)$.*

We obtain our results with a combination of the compressed suffix tree $\mathcal{T}$ of T and the Burrows-Wheeler transform $\overline{B}$ of the reversed text $\overline{T}$. We manage to simulate the suffix tree traversal for P, simultaneously on $\mathcal{T}$ and on $\overline{B}$. With a combination of storing deterministic dictionaries and precomputed rank values for sampled nodes of $\mathcal{T}$, and a constant-time method to compute an extension of partial rank queries that considers small ranges in $\overline{B}$, we manage to ensure that all the suffix tree steps, except one, require constant time. The remaining one is solved with general rank queries in time $O(\log\log_w\sigma)$. As a byproduct, we show that the compressed sequence representations that obtain those rank times [6] can also be built in linear deterministic time.

Compared with previous work, other indexes may be faster at counting, but either they are not built in linear deterministic time [5, 18, 26] or they are not compressed [26, 7]. Our index outperforms all the previous compressed [13, 1, 6], as well as some uncompressed [14], indexes that can be built deterministically.

2 Related Work

Let T be a string of length n over an alphabet of size σ that is indexed to support searches for patterns P. It is generally assumed that $\sigma = o(n)$, a reasonable convention we will follow. Searches typically require to *count* the number of times P appears in T, and then *locate* the positions of T where P occurs. The vast majority of the indexes for this task are suffix tree [29] or suffix array [21] variants.

Table 1 Our results in context. The x axis refers to the space used by the indexes (compressed meaning $nH_k(T) + o(n \log \sigma)$ bits and uncompressed meaning $\Theta(n \log n)$ bits), and the y axis refers to the *linear-time* construction. In the cells we show the counting time for a pattern P. We only list the dominant alternatives, graying out those outperformed by our new results.

	Compressed	Uncompressed
Deterministic	$\lvert P\rvert \log\log\sigma$ [1] $\lvert P\rvert(1+\log_w \sigma)$ [6] $\lvert P\rvert+\log\log_w \sigma$ **(ours)**	$\lvert P\rvert + \log\log\sigma$ [14] $\lvert P\rvert/\log_\sigma n+\log\lvert P\rvert+\log\log\sigma$ [7]
Randomized	$\lvert P\rvert(1+\log\log_w \sigma)$ [6] $\lvert P\rvert$ [5]	$\lvert P\rvert/\log_\sigma n + \log_\sigma^\epsilon n$ [18, 26]

The suffix tree can be built in linear deterministic time [29], even on arbitrarily large integer alphabets [11]. The suffix array can be easily derived from the suffix tree in linear time, but it can also be built independently in linear deterministic time [20]. In their basic forms, these structures allow counting the number of occurrences of a pattern P in T in time $O(|P| \log \sigma)$ (suffix tree) or $O(|P| + \log n)$ (suffix array). Once counted, the occurrences can be located in constant time each.

Cole et al. [10] introduced the *suffix trays*, a simple twist on suffix trees that reduces their counting time to $O(|P| + \log \sigma)$. Fischer and Gawrychowski [14] introduced the *wexponential search trees*, which yield dynamic suffix trees with counting time $O(|P| + \log\log \sigma)$.

All these structures can be built in linear deterministic time, but require $\Theta(n \log n)$ bits of space, which challenges their practicality when handling large text collections.

Faster counting is possible if we resort to perfect hashing and give away the linear deterministic construction time. In the classical suffix tree, we can easily achieve $O(|P|)$ time by hashing the children of suffix tree nodes, and this is optimal in general. In the RAM model with word size $\Theta(\log n)$, and if the consecutive symbols of P come packed into $|P|/\log_\sigma n$ words, the optimal time is instead $O(|P|/\log_\sigma n)$. This optimal time was recently reached by Navarro and Nekrich [26] (note that their time is not optimal if $w = \omega(\log n)$), with a simple application of weak-prefix search, already hinted in the original article [2]. However, even the randomized construction time of the weak-prefix search structure is $O(n \log^\epsilon n)$, for any constant $\epsilon > 0$. By replacing the weak-prefix search with the solution of Grossi and Vitter [18] for the last nodes of the search, and using a randomized construction of their perfect hash functions, the index of Navarro and Nekrich [26] can be built in linear randomized time and count in time $O(|P|/\log_\sigma n + \log_\sigma^\epsilon n)$. Only recently, simultaneously with our work, a deterministic linear-time construction algorithm was finally obtained for an index obtaining $O(|P|/\log_\sigma n + \log|P| + \log\log\sigma)$ counting time [7].

Still, these structures are not compressed. Compressed suffix trees and arrays appeared in the year 2000 [25]. To date, they take the space of the compressed text and replace it, in the sense that they can extract any desired substring of T; they are thus called self-indexes. The space occupied is measured in terms of the k-th order empirical entropy of T, $H_k(T) \le \log \sigma$ [22], which is a lower bound on the space reached by any statistical compressor that encodes each symbol considering only the k previous ones. Self-indexes may occupy as little as $nH_k(T) + o(n(H_k(T) + 1))$ bits, for any $k \le \alpha \log_\sigma n - 1$, for any constant $0 < \alpha < 1$.

The fastest self-indexes with linear-time deterministic construction are those of Barbay et al. [1], which counts in time $O(|P| \log\log\sigma)$, and Belazzougui and Navarro [6, Thm. 7], which counts in time $O(|P|(1+\log_w \sigma))$. The latter requires $O(n(1+\log_w \sigma))$ construction time, but if $\log\sigma = O(\log w)$, its counting time is $O(|P|)$ and its construction time is $O(n)$.

If we admit randomized linear-time constructions, then Belazzougui and Navarro [6, Thm. 10] reach $O(|P|(1+\log\log_w \sigma))$ counting time. At the expense of $O(n)$ further bits, in another work [5] they reach $O(|P|)$ counting time. Using $O(n\log\sigma)$ bits, and if P comes in packed form, Grossi and Vitter [18] can count in time $O(|P|/\log_\sigma n + \log_\sigma^\epsilon n)$, for any constant $\epsilon > 0$, however their construction requires $O(n\log\sigma)$ time.

Table 1 puts those results and our contribution in context. Our new self-index, with $O(|P| + \log\log_w \sigma)$ counting time, linear-time deterministic construction, and $nH_k(T) + o(n\log\sigma)$ bits of space, dominates all the compressed indexes with linear-time deterministic construction [1, 6], as well as some uncompressed ones [14] (to be fair, we do not cover the case $\log\sigma = O(\log w)$, as in this case the previous work [6, Thm. 7] already obtains our result). Our self-index also dominates a previous one with linear-time randomized construction [6, Thm. 10], which we incidentally show can also be built deterministically. The only aspect in which some of the dominated indexes outperform ours is in that they may use $o(n(H_k(T)+1))$ [6, Thm. 10] or $o(n)$ [6, Thm. 7] bits of redundancy, instead of our $o(n\log\sigma)$ bits.

3 Preliminaries

We denote by $T[i..]$ the suffix of $T[0,n-1]$ starting at position i and by $T[i..j]$ the substring that begins with $T[i]$ and ends with $T[j]$, $T[i..] = T[i]T[i+1]\ldots T[n-1]$ and $T[i..j] = T[i]T[i+1]\ldots T[j-1]T[j]$. We assume that the text T ends with a special symbol \$ that lexicographically precedes all other symbols in T. The alphabet size is σ and symbols are integers in $[0..\sigma-1]$ (so \$ corresponds to 0). In this paper, as in the previous work on this topic, we use the word RAM model of computation. A machine word consists of $w = \Omega(\log n)$ bits and we can execute standard bit and arithmetic operations in constant time. We assume for simplicity that the alphabet size $\sigma = O(n/\log n)$ (otherwise the text is almost incompressible anyway [15]). We also assume $\log\sigma = \omega(\log w)$, since otherwise our goal is already reached in previous work [6, Thm. 7].

3.1 Rank and Select Queries

We define three basic queries on sequences. Let $B[0..n-1]$ be a sequence of symbols over alphabet $[0..\sigma-1]$. The rank query, $\mathrm{rank}_a(i,B)$, counts how many times a occurs among the first $i+1$ symbols in B, $\mathrm{rank}_a(i,B) = |\{\, j \le i,\ B[j] = a\}|$. The select query, $\mathrm{select}_a(i,B)$, finds the position in B where a occurs for the i-th time, $\mathrm{select}_a(i,B) = j$ iff $B[j] = a$ and $\mathrm{rank}_a(j,B) = i$. The third query is $\mathrm{access}(i,B)$, which returns simply $B[i]$.

We can answer access queries in $O(1)$ time and select queries in any $\omega(1)$ time, or vice versa, and rank queries in time $O(\log\log_w \sigma)$, which is optimal [6]. These structures use $n\log\sigma + o(n\log\sigma)$ bits, and we will use variants that require only compressed space. In this paper, we will show that those structures can be built in linear deterministic time.

An important special case of rank queries is the partial rank query, $\mathrm{rank}_{B[i]}(i,B)$, which asks how many times $B[i]$ occurs in $B[0..i]$. Unlike general rank queries, partial rank queries can be answered in $O(1)$ time [6]. Such a structure can be built in $O(n)$ deterministic time and requires $O(n\log\log\sigma)$ bits of working and final space [24, Thm. A.4.1].

For this paper, we define a generalization of partial rank queries called interval rank queries, $\mathrm{rank}_a(i,j,B) = \langle \mathrm{rank}_a(i-1,B), \mathrm{rank}_a(j,B)\rangle$, from where in particular we can deduce the number of times a occurs in $B[i..j]$. If a does not occur in $B[i..j]$, however, this query just returns *null* (this is why it can be regarded as a generalized partial rank query).

In the special case where the alphabet size is small, $\log\sigma = O(\log w)$, we can represent B so that rank, select, and access queries are answered in $O(1)$ time [6, Thm. 7], but we are not focusing on this case in this paper, as the problem has already been solved for this case.

3.2 Suffix Array and Suffix Tree

The suffix tree [29] for a string $T[0..n-1]$ is a compacted digital tree on the suffixes of T, where the leaves point to the starting positions of the suffixes. We call X_u the string leading to suffix tree node u. The suffix array [21] is an array $SA[0..n-1]$ such that $SA[i] = j$ if and only if $T[j..]$ is the $(i+1)$-th lexicographically smallest suffix of T. All the occurrences of a substring P in T correspond to suffixes of T that start with P. These suffixes descend from a single suffix tree node, called the *locus* of P, and also occupy a contiguous interval in the suffix array SA. Note that the locus of P is the node u closest to the root for which P is a prefix of X_u. If P has no locus node, then it does not occur in T.

3.3 Compressed Suffix Array and Tree

A compressed suffix array (CSA) is a compact data structure that provides the same functionality as the suffix array. The main component of a CSA is the one that allows determining, given a pattern P, the suffix array range $SA[i..j]$ of the prefixes starting with P. Counting is then solved as $j-i+1$. For locating any cell $SA[k]$, and for extracting any substring S from T, most CSAs make use of a sampled array SAM_b, which contains the values of $SA[i]$ such that $SA[i] \bmod b = 0$ or $SA[i] = n-1$. Here b is a tradeoff parameter: CSAs require $O(n\frac{\log n}{b})$ further bits and can locate in time proportional to b and extract S in time proportional to $b+|S|$. We refer to a survey [25] for a more detailed description.

A compressed suffix tree [28] is formed by a compressed suffix array and other components that add up to $O(n)$ bits. These include in particular a representation of the tree topology that supports constant-time computation of the preorder of a node, its number of children, its j-th child, its number of descendant leaves, and lowest common ancestors, among others [27]. Computing node preorders is useful to associate satellite information to the nodes.

Both the compressed suffix array and tree can be built in $O(n)$ deterministic time using $O(n\log\sigma)$ bits of space [24].

3.4 Burrows-Wheeler Transform and FM-index

The Burrows-Wheeler Transform (BWT) [8] of a string $T[0..n-1]$ is another string $B[0..n-1]$ obtained by sorting all possible rotations of T and writing the last symbol of every rotation (in sorted order). The BWT is related to the suffix array by the identity $B[i] = T[(SA[i]-1) \bmod n]$. Hence, we can build the BWT by sorting the suffixes and writing the symbols that precede the suffixes in lexicographical order.

The FM-index [12, 13] is a CSA that builds on the BWT. It consists of the following three main components: (1) the BWT B of T; (2) the array $Acc[0..\sigma-1]$ where $Acc[i]$ holds the total number of symbols $a < i$ in T (or equivalently, the total number of symbols $a < i$ in B); (3) the sampled array SAM_b.

The interval of a pattern string $P[0..m-1]$ in the suffix array SA can be computed on the BWT B. The interval is computed backwards: for $i = m-1, m-2, \ldots$, we identify the interval of $P[i..m-1]$ in B. The interval is initially the whole $B[0..n-1]$. Suppose that we know the interval $B[i_1..j_1]$ that corresponds to $P[i+1..m-1]$. Then the interval $B[i_2..j_2]$ that corresponds to $P[i..m-1]$ is computed as $i_2 = Acc[a] + \mathrm{rank}_c(i_1-1, B)$ and $j_2 = Acc[a] + \mathrm{rank}_c(j_1, B) - 1$, where $a = P[i]$. Thus the interval of P is found by answering $2m$ rank queries. Any sequence representation offering rank and access queries can then be applied on B to obtain an FM-index.

An important procedure on the FM-index is the computation of the function LF, defined as: if $SA[j] = i+1$, then $SA[LF(j)] = i$. LF can be computed with access and partial rank queries on B, $LF(j) = \mathrm{rank}_{B[j]}(i, B) + Acc[B[j]] - 1$, and thus constant-time computation of LF is possible. Using SAM_b and $O(b)$ applications of LF, we can locate any cell $SA[r]$. A similar procedure extracts any substring S of T with $O(b + |S|)$ applications of LF.

4 Small Interval Rank Queries

We start by showing how a compressed data structure that supports select queries can be extended to support a new kind of queries that we dub *small interval rank queries*. An interval query $\mathrm{rank}_a(i, j, B)$ is a small interval rank query if $j - i \le \log^2\sigma$. Our compressed index relies on the following result.

▶ **Lemma 2.** *Suppose that we are given a data structure that supports* access *queries on a sequence* $C[0..m-1]$*, on alphabet* $[0..\sigma-1]$*, in time* t*. Then, using* $O(m \log\log\sigma)$ *additional bits, we can support small interval rank queries on* C *in* $O(t)$ *time.*

Proof. We split C into groups G_i of $\log^2\sigma$ consecutive symbols, $G_i = C[i\log^2\sigma..(i+1)\log^2\sigma - 1]$. Let A_i denote the sequence of the distinct symbols that occur in G_i. Storing A_i directly would need $\log\sigma$ bits per symbol. Instead, we encode each element of A_i as its first position in G_i, which needs only $O(\log\log\sigma)$ bits. With this encoded sequence, since we have $O(t)$-time access to C, we have access to any element of A_i in time $O(t)$. In addition, we store a succinct SB-tree [17] on the elements of A_i. This structure uses $O(p \log\log u)$ bits to index p elements in $[1..u]$, and supports predecessor (and membership) queries in time $O(\log p / \log\log u)$ plus one access to A_i. Since $u = \sigma$ and $p \le \log^2\sigma$, the query time is $O(t)$ and the space usage is bounded by $O(m \log\log\sigma)$ bits.

For each $a \in A_i$ we also keep the increasing list $I_{a,i}$ of all the positions where a occurs in G_i. Positions are stored as differences with the left border of G_i: if $C[j] = a$, we store the difference $j - i\log^2\sigma$. Hence elements of $I_{a,i}$ can also be stored in $O(\log\log\sigma)$ bits per symbol, adding up to $O(m \log\log\sigma)$ bits. We also build an SB-tree on top of each $I_{a,i}$ to provide for predecessor searches.

Using the SB-trees on A_i and $I_{a,i}$, we can answer small interval rank queries $\mathrm{rank}_a(x, y, C)$. Consider a group $G_i = C[i\log^2\sigma..(i+1)\log^2\sigma - 1]$, an index k such that $i\log^2\sigma \le k \le (i+1)\log^2\sigma$, and a symbol a. We can find the largest $i\log^2\sigma \le r \le k$ such that $C[r] = a$, or determine it does not exist: First we look for the symbol a in A_i; if $a \in A_i$, we find the predecessor of $k - i\log^2\sigma$ in $I_{a,i}$.

Now consider an interval $C[x..y]$ of size at most $\log^2\sigma$. It intersects at most two groups, G_i and G_{i-1}. We find the rightmost occurrence of symbol a in $C[x..y]$ as follows. First we look for the rightmost occurrence $y' \le y$ of a in G_i; if a does not occur in $C[i\log^2\sigma..y]$, we look for the rightmost occurrence $y' \le i\log^2\sigma - 1$ of a in G_{i-1}. If this is $\ge x$, we find the leftmost occurrence x' of a in $C[x..y]$ using a symmetric procedure. When $x' \le y'$ are found, we can compute $\text{rank}_a(x', C)$ and $\text{rank}_a(y', C)$ in $O(1)$ time by answering partial rank queries (Section 3.1). These are supported in $O(1)$ time and $O(m \log\log\sigma)$ bits. The answer is then $\langle \text{rank}_a(x', C) - 1, \text{rank}_a(y', C)\rangle$, or *null* if a does not occur in $C[x..y]$. ◀

The construction of the small interval rank data structure is dominated by the time needed to build the succinct SB-trees [17]. These are simply B-trees with arity $O(\sqrt{\log u})$ and height $O(\log p / \log\log u)$, where in each node a Patricia tree for $O(\log\log u)$-bit chunks of the keys are stored. To build the structure in $O(\log p / \log\log u)$ time per key, we only need to build those Patricia trees in linear time. Given that the total number of bits of all the keys to insert in a Patricia tree is $O(\sqrt{\log u} \log\log u)$, we do not even need to build the Patricia tree. Instead, a universal precomputed table may answer any Patricia tree search for any possible set of keys and any possible pattern, in constant time. The size of the table is $O(2^{O(\sqrt{\log u} \log\log u)} \sqrt{\log u}) = o(u)$ bits (the authors [17] actually use a similar table to answer queries). For our values of p and u, the construction requires $O(mt)$ time and the universal table is of $o(\sigma)$ bits.

5 Compressed Index

We classify the nodes of the suffix tree $\mathcal{T}$ of T into heavy, light, and special, as in previous work [26, 24]. Let $d = \log\sigma$. A node u of $\mathcal{T}$ is *heavy* if it has at least d leaf descendants and *light* otherwise. We say that a heavy node u is *special* if it has at least two heavy children.

For every special node u, we construct a deterministic dictionary [19] D_u that contains the labels of all the heavy children of u: If the jth child of u, u_j, is heavy and the first symbol on the edge from to u to u_j is a_j, then we store the key a_j in D_u with j as satellite data. If a heavy node u has only one heavy child u_j and d or more light children, then we also store the data structure D_u (containing only that heavy child of u). If, instead, a heavy node has one heavy child and less than d light children, we just keep the index of the heavy child using $O(\log d) = O(\log\log\sigma)$ bits.

The second component of our index is the Burrows-Wheeler Transform $\overline{B}$ of the reverse text $\overline{T}$. We store a data structure that supports rank, partial rank, select, and access queries on $\overline{B}$. It is sufficient for us to support access and partial rank queries in $O(1)$ time and rank queries in $O(\log\log_w \sigma)$ time. We also construct the data structure described in Lemma 2, which supports small interval rank queries in $O(1)$ time. Finally, we explicitly store the answers to some rank queries. Let $\overline{B}[l_u..r_u]$ denote the range of $\overline{X_u}$, where $\overline{X_u}$ is the reverse of X_u, for a suffix tree node u. For all data structures D_u and for every symbol $a \in D_u$ we store the values of $\text{rank}_a(l_u - 1, \overline{B})$ and $\text{rank}_a(r_u, \overline{B})$.

Let us show how to store the selected precomputed answers to rank queries in $O(\log\sigma)$ bits per query. Following a known scheme [16], we divide the sequence $\overline{B}$ into chunks of size σ. For each symbol a, we encode the number d_k of times a occurs in each chunk k in a binary sequence $A_a = 01^{d_0}01^{d_1}01^{d_2}\ldots$. If a symbol $\overline{B}[i]$ belongs to chunk $k = \lfloor i/\sigma \rfloor$, then $\text{rank}_a(i, \overline{B})$ is $\text{select}_0(k+1, A_a) - k$ plus the number of times a occurs in $\overline{B}[k\sigma..i]$. The former value is computed in $O(1)$ time with a structure that uses $|A_a| + o(|A_a|)$ bits [9], whereas the latter value is in $[0, \sigma]$ and thus can be stored in D_u using just $O(\log\sigma)$ bits. The total size of all the sequences A_a is $O(n)$ bits.

Therefore, D_u needs $O(\log\sigma)$ bits per element. The total number of elements in all the structures D_u is equal to the number of special nodes plus the number of heavy nodes with one heavy child and at least d light children. Hence all D_u contain $O(n/d)$ symbols and use $O((n/d)\log\sigma) = O(n)$ bits of space. Indexes of heavy children for nodes with only one heavy child and less than d light children add up to $O(n \log\log\sigma)$ bits. The structures for partial rank and small interval rank queries on $\overline{B}$ use $O(n \log\log\sigma)$ further bits. Since we assume that σ is $\omega(1)$, we can simplify $O(n \log\log\sigma) = o(n \log\sigma)$.

The sequence representation that supports access and rank queries on $\overline{B}$ can be made to use $nH_k(T) + o(n(H_k(T) + 1))$ bits, by exploiting the fact that it is built on a BWT [6, Thm. 10].[1] We note that they use constant-time select queries on $\overline{B}$ instead of constant-time access, so they can use select queries to perform LF^{-1}-steps in constant time. Instead, with our partial rank queries, we can perform LF-steps in constant time (recall Section 3.4), and thus have constant-time access instead of constant-time select on $\overline{B}$ (we actually do not use query select at all). They avoid this solution because partial rank queries require $o(n \log \sigma)$ bits, which can be more than $o(n(H_k(T) + 1))$, but we are already paying this price.

Apart from this space, array Acc needs $O(\sigma \log n) = O(n)$ bits and SAM_b uses $O(n\frac{\log n}{b})$. The total space usage of our self-index then adds up to $nH_k(T) + o(n \log \sigma) + O(n\frac{\log n}{b})$ bits.

6 Pattern Search

Given a query string P, we will find in time $O(|P| + \log\log_w \sigma)$ the range of the reversed string $\overline{P}$ in $\overline{B}$. A backward search for P in B will be replaced by an analogous backward search for $\overline{P}$ in $\overline{B}$, that is, we will find the range of $\overline{P[0..i]}$ if the range of $\overline{P[0..i-1]}$ is known. Let $[l_i..r_i]$ be the range of $\overline{P[0..i]}$. We can compute l_i and r_i from l_{i-1} and r_{i-1} as $l_i = Acc[a] + \text{rank}_a(l_{i-1} - 1, \overline{B})$ and $r_i = Acc[a] + \text{rank}_a(r_{i-1}, \overline{B}) - 1$, for $a = P[i]$. Using our auxiliary data structures on $\overline{B}$ and the additional information stored in the nodes of the suffix tree $\mathcal{T}$, we can answer the necessary rank queries in constant time (with one exception). The idea is to traverse the suffix tree $\mathcal{T}$ in synchronization with the forward search on $\overline{B}$, until the locus of P is found or we determine that P does not occur in T.

Our procedure starts at the root node of $\mathcal{T}$, with $l_{-1} = 0$, $r_{-1} = n - 1$, and $i = 0$. We compute the ranges $\overline{B}[l_i..r_i]$ that correspond to $\overline{P[0..i]}$ for $i = 0, \ldots, |P| - 1$. Simultaneously, we move down in the suffix tree. Let u denote the last visited node of $\mathcal{T}$ and let $a = P[i]$. We denote by u_a the next node that we must visit in the suffix tree, i.e., u_a is the locus of $P[0..i]$. We can compute l_i and r_i in $O(1)$ time if $\text{rank}_a(r_{i-1}, \overline{B})$ and $\text{rank}_a(l_{i-1} - 1, \overline{B})$ are known. We will show below that these queries can be answered in constant time because either (a) the answers to rank queries are explicitly stored in D_u or (b) the rank query that must be answered is a small interval rank query. The only exception is the situation when we move from a heavy node to a light node in the suffix tree; in this case the rank query takes $O(\log\log_w \sigma)$ time. We note that, once we are in a light node, we need not descend in $\mathcal{T}$ anymore; it is sufficient to maintain the interval in $\overline{B}$.

For ease of description we distinguish between the following cases.

1. Node u is heavy and $a \in D_u$. In this case we identify the heavy child u_a of u that is labeled with a in constant time using the deterministic dictionary. We can also find l_i and r_i in time $O(1)$ because $\text{rank}_a(l_{i-1} - 1, \overline{B})$ and $\text{rank}_a(r_{i-1}, \overline{B})$ are stored in D_u.
2. Node u is heavy and $a \notin D_u$. In this case u_a, if it exists, is a light node. We then find it with two standard rank queries on $\overline{B}$, in order to compute l_i and r_i or determine that P does not occur in T.
3. Node u is heavy but we do not keep a dictionary D_u for the node u. In this case u has at most one heavy child and less than d light children. We have two subcases:
   a. If u_a is the (only) heavy node, we find this out with a single comparison, as the heavy node is identified in u. However, the values $\text{rank}_a(l_{i-1} - 1, \overline{B})$ and $\text{rank}_a(r_{i-1}, \overline{B})$ are not stored in u. To compute them, we exploit the fact that the number of non-a's in $\overline{B}[l_{i-1}..r_{i-1}]$ is less than d^2, as all the children apart from u_a are light and less than d.

[1] In fact it is $nH_k(\overline{T})$, but this is $nH_k(T) + O(\log n)$ [12, Thm. A.3].

Therefore, the first and the last occurrences of a in $\overline{B}[l_{i-1}..r_{i-1}]$ must be at distance less than d^2 from the extremes l_{i-1} and r_{i-1}, respectively. Therefore, a small interval rank query, $\text{rank}_a(l_{i-1}, l_{i-1}+d^2, \overline{B})$, gives us $\text{rank}_a(l_{i-1}-1, \overline{B})$, since there is for sure an a in the range. Analogously, $\text{rank}_a(r_{i-1}-d^2, r_{i-1}, \overline{B})$ gives us $\text{rank}_a(r_{i-1}, \overline{B})$.

b. If u_a is a light node, we compute l_i and r_i with two standard rank queries on $\overline{B}$ (or we might determine that P does not appear in T).

4. Node u is light. In this case, $P[0..i-1]$ occurs at most d times in T. Hence $\overline{P[0..i-1]}$ also occurs at most d times in $\overline{T}$ and $r_{i-1} - l_{i-1} \leq d$. Therefore we can compute r_i and l_i in $O(1)$ time by answering a small interval rank query, $\langle \text{rank}_a(l_{i-1}-1, \overline{B}), \text{rank}_a(r_{i-1}, \overline{B})\rangle$. If this returns *null*, then P does not occur in T.

5. We are on an edge of the suffix tree between a node u and some child u_j of u. In this case all the occurrences of $P[0..i-1]$ in T are followed by the same symbol, c, and all the occurrences of $\overline{P[0..i-1]}$ are preceded by c in $\overline{T}$. Therefore $\overline{B}[l_{i-1}..r_{i-1}]$ contains only the symbol c. This situation can be verified with access and partial rank queries on $\overline{B}$: $\overline{B}[r_{i-1}] = \overline{B}[l_{i-1}] = c$ and $\text{rank}_c(r_{i-1}, \overline{B}) - \text{rank}_c(l_{i-1}, \overline{B}) = r_{i-1} - l_{i-1}$. In this case, if $a \neq c$, then P does not occur in T; otherwise we obtain the new range with the partial rank query $\text{rank}_c(r_{i-1}, \overline{B})$, and $\text{rank}_c(l_{i-1}-1, \overline{B}) = \text{rank}_c(r_{i-1}, \overline{B}) - (r_{i-1} - l_{i-1} + 1)$. Note that if u is light we do not need to consider this case; we may directly apply case 4.

Except for the cases 2 and 3b, we can find l_i and r_i in $O(1)$ time. In cases 2 and 3b we need $O(\log\log_w \sigma)$ time to answer general rank queries. However, these cases only take place when the node u is heavy and its child u_a is light. Since all descendants of a light node are light, those cases occur only once along the traversal of P. Hence the total time to find the range of $\overline{P}$ in $\overline{B}$ is $O(|P| + \log\log_w \sigma)$. Once the range is known, we can count and report all occurrences of $\overline{P}$ in the standard way.

7 Linear-Time Construction

7.1 Sequences and Related Structures

Apart from constructing the BWT $\overline{B}$ of $\overline{T}$, which is a component of the final structure, the linear-time construction of the other components requires that we also build, as intermediate structures, the BWT B of T, and the compressed suffix trees $\overline{\mathcal{T}}$ and $\mathcal{T}$ of $\overline{T}$ and T, respectively. All these are built in $O(n)$ deterministic time and using $O(n \log \sigma)$ bits of space [24]. We also keep, on top of both $\overline{B}$ and B, $O(n \log\log \sigma)$-bit data structures able to report, for any interval $\overline{B}[i..j]$ or $B[i..j]$, all the distinct symbols from this interval, and their frequencies in the interval. The symbols are retrieved in arbitrary order. These auxiliary data structures can also be constructed in $O(n)$ time [24, Sec. A.5]. On top of the sequences B and $\overline{B}$, we build the representation that supports access in $O(1)$ and rank in $O(\log\log_w \sigma)$ time [6]. This was built using perfect hashing, but it can also be built deterministically [4, Lem. 11].

7.2 Structures D_u

The most complex part of the construction is to fill the data of the D_u structures. We visit all the nodes of $\mathcal{T}$ and identify those nodes u for which the data structure D_u must be constructed. This can be easily done in linear time, by using the constant-time computation of the number of descendant leaves. To determine if we must build D_u, we traverse its children $u_1, u_2, \ldots$ and count their descendant leaves to decide if they are heavy or light.

We use a bit vector D to mark the preorders of the nodes u for which D_u will be constructed: If p is the preorder of node u, then it stores a structure D_u iff $D[p] = 1$, in which case D_u is stored in an array at position $\text{rank}_1(D, p)$. If, instead, u does not store D_u

but it has one heavy child, we store its child rank in another array indexed by $\mathrm{rank}_0(D,p)$, using $\log\log\sigma$ bits per cell.

The main difficulty is how to compute the symbols a to be stored in D_u, and the ranges $\overline{B}[l_u, r_u]$, for all the selected nodes u. It is not easy to do this through a preorder traversal of $\mathcal{T}$ because we would need to traverse edges that represent many symbols. Our approach, instead, is inspired by the navigation of the suffix-link tree using two BWTs given by Belazzougui et al. [3]. Let $\mathcal{T}_w$ denote the tree whose edges correspond to Weiner links between internal nodes in $\mathcal{T}$. That is, the root of $\mathcal{T}_w$ is the same root of $\mathcal{T}$ and, if we have internal nodes $u, v \in \mathcal{T}$ where $X_v = a \cdot X_u$ for some symbol a, then v descends from u by the symbol a in $\mathcal{T}_w$. It is well known that that the nodes of $\mathcal{T}_w$ are the internal nodes of $\mathcal{T}$.

We do not build $\mathcal{T}_w$ explicitly, but just traverse its nodes conceptually in depth-first order and compute the symbols to store in the structures D_u and the intervals in $\overline{B}$. Let u be the current node of $\mathcal{T}$ in this traversal and $\overline{u}$ its corresponding locus in $\overline{\mathcal{T}}$. Assume for now that $\overline{u}$ is a node, too. Let $[l_u, r_u]$ be the interval of X_u in B and $[l_{\overline{u}}, r_{\overline{u}}]$ be the interval of the reverse string $X_{\overline{u}}$ in $\overline{B}$.[2] Our algorithm starts at the root nodes of $\mathcal{T}_w$, $\mathcal{T}$, and $\overline{\mathcal{T}}$, which correspond to the empty string, and the intervals in B and $\overline{B}$ are $[l_u, r_u] = [l_{\overline{u}}, r_{\overline{u}}] = [0, n-1]$. We will traverse only the heavy nodes, yet in some cases we will have to work on all the nodes. We ensure that on heavy nodes we work at most $O(\log\sigma)$ time, and at most $O(1)$ time on arbitrary nodes.

Upon arriving at each node u, we first compute its heavy children. From the topology of $\mathcal{T}$ we identify the interval $[l_i, r_i]$ for every child u_i of u, by counting leaves in the subtrees of the successive children of u. By reporting all the distinct symbols in $\overline{B}[l_{\overline{u}}..r_{\overline{u}}]$ with their frequencies, we identify the labels of those children. However, the labels are retrieved in arbitrary order and we cannot afford sorting them all. Yet, since the labels are associated with their frequencies in $\overline{B}[l_{\overline{u}}..r_{\overline{u}}]$, which match their number of leaves in the subtrees of u, we can discard the labels of the light children, that is, those appearing less than d times in $\overline{B}[l_{\overline{u}}..r_{\overline{u}}]$. The remaining, heavy, children are then sorted and associated with the successive heavy children u_i of u in $\mathcal{T}$.

If our preliminary pass marked that a D_u structure must be built, we construct at this moment the deterministic dictionary [19] with the labels a of the heavy children of u we have just identified, and associate them with the satellite data $\mathrm{rank}_a(l_{\overline{u}} - 1, \overline{B})$ and $\mathrm{rank}_a(r_{\overline{u}}, \overline{B})$. This construction takes $O(\log\sigma)$ time per element, but it includes only heavy nodes.

We now find all the Weiner links from u. For every (heavy or light) child u_i of u, we compute the list L_i of all the distinct symbols that occur in $B[l_i..r_i]$. We mark those symbols a in an array $V[0..\sigma-1]$ that holds three possible values: not seen, seen, and seen (at least) twice. If $V[a]$ is not seen, then we mark it as seen; if it is seen, we mark it as seen twice; otherwise we leave it as seen twice. We collect a list E_u of the symbols that are seen twice along this process, in arbitrary order. For every symbol a in E_u, there is an Weiner link from u labeled by a: Let $X = X_u$; if a occurred in L_i and L_j then both aXa_i and aXa_j occur in T and there is a suffix tree node that corresponds to the string aX. The total time to build E_u amortizes to $O(n)$: for each child v of u, we pay $O(1)$ time for each child the node $\overline{v}$ has in $\overline{\mathcal{T}}$; each node in $\overline{\mathcal{T}}$ contributes once to the cost.

The targets of the Weiner links from u in $\mathcal{T}$ correspond to the children of the node $\overline{u}$ in $\overline{\mathcal{T}}$. To find them, we collect all the distinct symbols in $B[l_u..r_u]$ and their frequencies. Again, we discard the symbols with frequency less than d, as they will lead to light nodes, which we do not have to traverse. The others are sorted and associated with the successive heavy

[2] In the rest of the paper we wrote $\overline{B}[l_u..r_u]$ instead of $\overline{B}[l_{\overline{u}}..r_{\overline{u}}]$ for simplicity, but this may cause confusion in this section.

children of $\overline{u}$. By counting leaves in the successive children, we obtain the intervals $\overline{B}[l'_i..r'_i]$ corresponding to the heavy children $\overline{u}'_i$ of $\overline{u}$.

We are now ready to continue the traversal of $\mathcal{T}_w$: for each Weiner link from u by symbol a leading to a heavy node, which turns out to be the i-th child of $\overline{u}$, we know that its node in $\overline{\mathcal{T}}$ is $\overline{u}'_i$ (computed from $\overline{u}$ using the tree topology) and its interval is $\overline{B}[l'_i..r'_i]$. To compute the corresponding interval on B, we use the backward step operation, $B[x, y] = B[Acc[a] + \mathrm{rank}_a(l_u - 1, B), Acc[a] + \mathrm{rank}_a(r_u, B) - 1]$. This requires $O(\log\log_w \sigma)$ time, but applies only to heavy nodes. Finally, the corresponding node in $\mathcal{T}$ is obtained in constant time as the lowest common ancestor of the x-th and the y-th leaves of $\mathcal{T}$.

In the description above we assumed for simplicity that $\overline{u}$ is a node in $\overline{\mathcal{T}}$. In the general case $\overline{u}$ can be located on an edge of $\overline{\mathcal{T}}$. This situation arises when all occurrences of $\overline{X_u}$ in the reverse text $\overline{T}$ are followed by the same symbol a. In this case there is at most one Weiner link from u; the interval in $\overline{B}$ does not change as we follow that link.

A recursive traversal of $\mathcal{T}_w$ might require $O(n\sigma \log n)$ bits for the stack, because we store several integers associated to heavy children during the computation of each node u. We can reduce this to $O(\sigma \log^2 n) = O(n \log \sigma)$ by standard means [3, Lem. 1].

We have spent at most $O(\log \sigma)$ time on heavy nodes, which are $O(n/d) = O(n/\log \sigma)$ in total, thus these costs add up to $O(n)$. All other costs that apply to arbitrary nodes are $O(1)$. The structures for partial rank queries (and the succinct SB-trees) can also be built in linear deterministic time, as seen in Section 4. Then our index is constructed in $O(n)$ time.

8 Conclusions

We have shown how to build, in $O(n)$ deterministic time and using $O(n \log \sigma)$ bits of working space, a compressed self-index for a text T of length n over an alphabet of size σ that searches for patterns P in time $O(|P| + \log\log_w \sigma)$, on a w-bit word RAM machine. This improves upon previous compressed self-indexes requiring $O(|P| \log\log \sigma)$ [1] or $O(|P|(1 + \log_w \sigma))$ [6] time, on previous uncompressed indexes requiring $O(|P| + \log\log \sigma)$ time [14] (but that supports dynamism), and on previous compressed self-indexes requiring $O(|P|(1+\log\log_w \sigma))$ time and randomized construction (which we now showed how to build in linear deterministic time) [6]. The only indexes offering better search time require randomized construction [5, 18, 26] or $\Theta(n \log n)$ bits of space [26, 7].

In our extended paper [23], we show that using $O(n \log \sigma)$ bits of space, we can build in $O(n)$ deterministic time an index that searches in time $O(|P|/\log_\sigma n + \log n(\log\log n)^2)$. Current indexes obtaining similar counting time require $O(n \log \sigma)$ construction time [18] or higher [26], or $O(n \log n)$ bits of space [26, 7].

It is not clear if $O(|P|)$ time, or even $O(|P|/\log_\sigma n)$, query time can be achieved with a linear deterministic construction time, even if we allow $O(n \log n)$ bits of space for the index (this was recently approached, but some additive polylog factors remain [7]). This is the most interesting open problem for future research.

References

1 J. Barbay, F. Claude, T. Gagie, G. Navarro, and Y. Nekrich. Efficient fully-compressed sequence representations. *Algorithmica*, 69(1):232–268, 2014.

2 D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Fast prefix search in little space, with applications. In *Proc. 18th ESA*, LNCS 6346, pages 427–438, 2010.

3 D. Belazzougui, F. Cunial, J. Kärkkäinen, and V. Mäkinen. Versatile succinct representations of the bidirectional Burrows-Wheeler transform. In *Proc. 21st ESA*, pages 133–144, 2013.

4 D. Belazzougui, F. Cunial, J. Kärkkäinen, and V. Mäkinen. Linear-time string indexing and analysis in small space. *CoRR*, abs/1609.06378, 2016.

5 D. Belazzougui and G. Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Alg.*, 10(4):article 23, 2014.

6 D. Belazzougui and G. Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Alg.*, 11(4):article 31, 2015.

7 P. Bille, I. L. Gørtz, and F. R. Skjoldjensen. Deterministic indexing for packed strings. In *Proc. 28th CPM*, LIPIcs 78, page article 6, 2017.

8 M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

9 D. R. Clark. *Compact PAT Trees.* PhD thesis, University of Waterloo, Canada, 1996.

10 R. Cole, T. Kopelowitz, and M. Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015.

11 M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th FOCS*, pages 137–143, 1997.

12 P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.

13 P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Alg.*, 3(2):article 20, 2007.

14 J. Fischer and P. Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Proc. 26th CPM*, LNCS 9133, pages 160–171, 2015.

15 T. Gagie. Large alphabets and incompressibility. *Inf. Proc. Lett.*, 99(6):246–251, 2006.

16 A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proc. 17th SODA*, pages 368–373, 2006.

17 R. Grossi, A. Orlandi, R. Raman, and S. S. Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *Proc. 26th STACS*, pages 517–528, 2009.

18 R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comp.*, 35(2):378–407, 2005.

19 T. Hagerup, P. Bro Miltersen, and R. Pagh. Deterministic dictionaries. *J. Alg.*, 41(1):69 – 85, 2001.

20 J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006.

21 U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. Comp.*, 22(5):935–948, 1993.

22 G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.

23 J. I. Munro, G. Navarro, and Y. Nekrich. Fast compressed self-indexes with deterministic linear-time construction. *CoRR*, abs/1707.01743, 2017.

24 J. I. Munro, G. Navarro, and Y. Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Proc. 28th SODA*, pages 408–424, 2017.

25 G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comp. Surv.*, 39(1):article 2, 2007.

26 G. Navarro and Y. Nekrich. Time-optimal top-k document retrieval. *SIAM J. Comp.*, 46(1):89–113, 2017.

27 G. Navarro and K. Sadakane. Fully-functional static and dynamic succinct trees. *ACM Trans. Alg.*, 10(3):article 16, 2014.

28 K. Sadakane. Compressed suffix trees with full functionality. *Theor. Comp. Sys.*, 41(4):589–607, 2007.

29 P. Weiner. Linear pattern matching algorithms. In *Proc. 14th FOCS*, pages 1–11, 1973.

Satisfiability Algorithm for Syntactic Read-k-times Branching Programs*

Atsuki Nagao[1], Kazuhisa Seto[2], and Junichi Teruyama[3]

1 **Seikei University, Tokyo, Japan**
`a-nagao@st.seikei.ac.jp`

2 **Seikei University, Tokyo, Japan**
`seto@st.seikei.ac.jp`

3 **National Institute of Informatics, and JST, ERATO, Kawarabayashi Large Graph Project, Tokyo, Japan**
`teruyama@nii.ac.jp`

Abstract

The satisfiability of a given branching program is to determine whether there exists a consistent path from the root to 1-sink. In a syntactic read-k-times branching program, each variable appears at most k times in any path from the root to a sink. We provide a satisfiability algorithm for syntactic read-k-times branching programs with n variables and m edges that runs in time $O\left(\mathrm{poly}(n, m^{k^2}) \cdot 2^{(1-\mu(k))n}\right)$, where $\mu(k) = \frac{1}{4^{k+1}}$. Our algorithm is based on the decomposition technique shown by Borodin, Razborov and Smolensky [Computational Complexity, 1993].

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity: General

Keywords and phrases branching program, read-k-times, satisfiability, moderately exponential time, polynomial space

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.58

1 Introduction

Branching programs (BPs) are well studied computation models in theory and practice. A BP is a directed acyclic graph with a unique root node and two sink nodes. Each nonsink node is labeled using a variable, and the edges correspond to a variable's value of zero or one. Sink nodes are labeled either 0 or 1 depending on the output value. A BP computes a Boolean function naturally: it follows the edge corresponding to the input value from the root node to a sink node.

Given a BP, its satisfiability (BP SAT) involves the determination of whether there exists a consistent path from the root to 1-sink. Recently, BP SAT has become a significant problem because of the connection between satisfiability algorithms and lower bounds. Let C be a class of a circuit. Given a circuit in C, C-SAT is the determination of whether there exists an assignment to the input variables such that the circuit outputs 1. Williams [26] showed that to obtain $\mathbf{NEXP} \not\subseteq C$, it suffices to develop an $O\left(2^{n-\omega(\log n)}\right)$ time algorithm for C-SAT. Barrignton [3] showed that any function in $\mathbf{NC^1}$ can be computed using width-5 BPs of polynomial length. By combining these results, if we would like to prove $\mathbf{NEXP} \not\subseteq \mathbf{NC^1}$, it

* This work was supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (26730007); JST ERATO Grant Number JPMJER1201, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 58; pp. 58:1–58:10
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is sufficient to develop an $O\left(2^{n-\omega(\log n)}\right)$ time algorithm for width-5 BP SAT. In addition, the hardness of BP SAT implies the hardness of the Edit Distance and Longest Common Sequence problem [1]. Thus, the designing of a fast algorithm for BP SAT is one of the important tasks in the field of computational complexity.

For the SAT of some restricted BPs, polynomial or moderately exponential time algorithms are known. An ordered binary decision diagram (OBDD) is a BP that has the same order of variables in all paths from the root to any sink. By checking the reachability from the root to 1-sink, the OBDD SAT can be solved in polynomial time. A k-OBDD is a natural extension of an OBDD with k layers; all layers are OBDDs with the same order of variables. Bollig, Sauerhoff, Sieling, and Wegener [6] provided a polynomial time algorithm that solves the k-OBDD SAT for any constant k. A k-indexed binary decision diagram (k-IBDD) is the same as a k-OBDD, except that an OBDD in each layer may have a different order of variables. A k-IBDD SAT is known to be NP-complete when $k \geq 2$ [6]. Nagao, Seto, and Teruyama [18] proposed a satisfiability algorithm for any instances of k-IBDD SAT with cn edges, and its running time is $O\left(2^{(1-\mu_k(c))n}\right)$, where $\mu_k(c) = \Omega\left(\frac{1}{(\log c)^{2^{k-1}-1}}\right)$. Chen, Kabanets, Kolokolova, Shaltiel, and Zuckerman [10] showed that general BP SAT with $o(n^2)$ nodes can be determined in time $O\left(2^{n-\omega(\log n)}\right)$. However, there are not so much researches on BP SAT.

In this paper, we focus on syntactic read-k-times BPs. There exist two models of read-k-times BPs: *semantic* and *syntactic*. A read-k-times BP is *syntactic* if each variable appears at most k times in any path. It is *semantic* if each variable appears at most k times in any "computational" path. The semantic model is substantially stronger than the syntactic model. Beame, Saks, and Thathachar [5] showed that polynomial-size semantic read-twice BP can compute functions requiring exponential size on any syntactic read-k-times BP. To the best of our knowledge, non-trivial lower bounds on semantic read-twice BP are not known; however, the syntactic model is well-studied. Borodin, Razborov, and Smolensky [7] exhibited an explicit function of the lower bound of $\exp\left(\Omega\left(\frac{n}{k^3 4^k}\right)\right)$. Jukna [17] provided an explicit function f such that nondeterministic read-once BPs of polynomial size can compute $\neg f$ (i.e., the negation of f); however, to compute f, nondeterministic read-k-times BPs require a size of $\exp\left(\Omega\left(\frac{\sqrt{n}}{k^{2k}}\right)\right)$. Thathachar [25] showed that for any k, the computational power of read-$(k+1)$-times BPs is strictly stronger than that of read-k-times BPs. Sauerhoff [21] proved the exponential lower bound for randomized read-k-times BPs with a two-sided error.

When $k = 1$, syntactic read-k-times BP SAT can be determined in polynomial time by solving the reachability from the root to 1-sink. However, even when $k = 2$, this problem is known to be NP-complete; to the best our knowledge, there is no algorithm that is faster than the brute-force search. Therefore, we present a moderately exponential time algorithm for any constant $k \geq 2$. Our algorithm is based on the decomposition technique by Borodin, Razborov, and Smolensky [7].

▶ **Theorem 1.** *There exists a deterministic and polynomial space algorithm for a non-deterministic and syntactic read-k-times BP SAT with n variables and m edges that runs in time $O\left(\mathrm{poly}(n, m^{k^2}) \cdot 2^{(1-4^{-k-1})n}\right)$.*

1.1 Our Techniques

Our satisfiability algorithm consists of two steps as follows: **[Step 1: Decomposition]** Given a syntactic read-k-times BP B of m edges, we obtain the representation of a function computed by B as a disjunction of at most m^{2k^2} decomposed functions by using the decomposition algorithm proposed by Borodin, Razborov, and Smolensky [7]. It is sufficient to check the satisfiability of each decomposed function in the running time of Theorem 1, because if one of these functions is satisfiable then the input B is also satisfiable. Moreover, the property of the decomposition algorithm states that each decomposed function is a conjunction of at most $2k^2$ functions on small variable sets. Let us represent a conjunction of functions as a set of functions $\mathcal{F} = \{f_1, f_2, \dots, f_\ell\}$, where $\ell \leq 2k^2$. **[Step 2: Satisfiability Checking]** To check the satisfiability of $\mathcal{F}$, we find an assignment that all functions f_i are satisfied at the same time. Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a partition of $\mathcal{F}$. In addition, let X_1 and X_2 be sets of input variables appearing in only $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively and X_3 be a set of input variables appearing in both F_1 and F_2. If X_3 is an empty set, we can check the satisfiability of $\mathcal{F}_1$ and $\mathcal{F}_2$ independently in time $O(2^{|X_1|} + 2^{|X_2|})$ by exhaustive search on each set X_1 and X_2. If both $\mathcal{F}_1$ and $\mathcal{F}_2$ are satisfiable, we know that $\mathcal{F}$ is also satisfiable. Our algorithm assigns 0/1 value to the variables in X_3 and then performs the exhaustive search on each set X_1 and X_2. Assuming that $|X_1| + |X_2| + |X_3| = n$, we obtain the satisfiability of $\mathcal{F}$ in time $O(2^{|X_3|}(2^{|X_1|} + 2^{|X_2|})) = O(2^{n-\min\{|X_1|,|X_2|\}})$. Further, using probabilistic method, we show that the existence of a partition $(\mathcal{F}_1, \mathcal{F}_2)$ of $\mathcal{F}$ such that the value $\min\{|X_1|, |X_2|\}$ is adequately large to imply the running time in Theorem 1. Thus, we can save the running time of our satisfiability algorithm.

1.2 Related Work

A circuit satisfiability problem is, given a Boolean circuit, to find an assignment to the inputs of the circuit such that the circuit outputs 1. Recently, this problem has been studied extensively, and excellent algorithms that can outperform a brute-force search have been known for some restricted circuit classes such as conjunctive normal forms [2, 8, 12, 13, 14, 19, 22], $\mathbf{AC}^0$ [4, 9, 15], $\mathbf{ACC}^0$ [27], depth-2 threshold circuits [16], De Morgan formulas [11, 20, 24], and formulas over the full binary basis [23].

Paper Organization

The remainder of this paper is organized as follows. In Section 2, we provide the notation and definitions. In Section 3, we provide two algorithms. One is a decomposition algorithm based on the technique in [7]. The other is a satisfiability algorithm for a specific class of Boolean functions. In Section 4, we propose our satisfiability algorithm for syntactic read-k-times BPs.

2 Preliminaries

A set of integers $\{1, 2, \dots, n\}$ is denoted by $[n]$. For a set S, $|S|$ denotes the cardinality of S. Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables, and for $x \in X$, $\overline{x}$ denotes the negation of x. A *branching program* (*BP*), denoted by $B = (V, E)$, is a rooted directed acyclic multigraph. A BP has a unique root node r and two sink nodes (0-sink and 1-sink); 0-sink and 1-sink are nodes labeled by **0** and **1**, respectively. Each node except for the sink nodes is labeled from X. Each edge $e \in E$ has a label 0 (*0-edge*) or 1 (*1-edge*). We call node v an x_i*-node* when v's label is x_i. A BP B is *deterministic* if any nodes except the two sink nodes in B have exactly two outgoing edges: one is a 0-edge, and the other is a 1-edge. Otherwise, B is *nondeterministic*. For an edge $e = (u, v) \in E$, u is a *parent* of v and the *head* of e. The *in-degree* of v is defined as the number of parents of v.

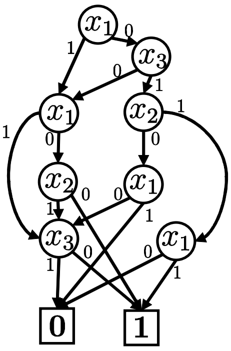


Figure 1 Syntactic read-twice branching program.

For a BP B on X, each input $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ activates all α_i-edges leaving the x_i-nodes in B, where $1 \le i \le n$. A *computation path* is a path from r to a 0-sink or from r to a 1-sink using only activated edges. A BP B outputs 0 if there is no computation path from the root r to a 1-sink; otherwise, B outputs 1. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. A BP B *represents* f if $f(\alpha)$ is equal to the output of B for any assignment $\alpha \in \{0,1\}^n$. Two BPs B_1 and B_2 are *equivalent* if B_1 and B_2 represent the same function. The size of B, denoted by $|B|$, is defined as the number of edges in B. A BP is *syntactic read-k-times* if each variable appears at most k times in each path. Figure 1 is an example of syntactic read-twice BPs ($k = 2$). A BP is *semantic read-k-times* if each variable appears at most k times in each computation path. In this paper, we use only the syntactic model and for simplicity we call it read-k-times BP.

For a BP B and two nodes v, w, a *subbranching program* $\langle B, v, w\rangle$ is a BP that contains v as the root node, w as the sink node, and every nodes and edges in all v-w paths in B. Given a BP B and nodes $v, w \in V$, $\langle B, v, w\rangle$ is constructed as follows:

1. Let V' be the subset of V such that $u \in V'$ is reachable from v and to w.
2. Output the subgraph of B induced by V'.

Note that, for any pair of nodes v and w, we can construct $\langle B, v, w\rangle$ in $O(|B|)$.

A *partial assignment* to $x = (x_1, \ldots, x_n)$ is $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1,*\}^n$ such that x_i is unset when $\alpha_i = *$; otherwise x_i is assigned to α_i. For any partial assignment $\alpha \in \{0,1,*\}^n$, a *support* of α is defined as $S(\alpha) := \{x_i \mid \alpha_i \neq *\}$. For partial assignments α and α' such that $S(\alpha)$ and $S(\alpha')$ are disjoint, $\alpha \circ \alpha'$ denotes the *composition* of α and α': $\alpha \circ \alpha'(i) = \alpha(i)$ if $x_i \in S(\alpha)$, $\alpha \circ \alpha'(i) = \alpha'(i)$ if $x_i \in S(\alpha')$, and $\alpha \circ \alpha'(i) = *$ otherwise. For instance, when $\alpha = (1, *, *)$ and $\alpha' = (*, *, 0)$, $\alpha \circ \alpha' = (1, *, 0)$.

3 Key Lemmas

In this section, we provide two key lemmas for our algorithm. First, we introduce the decomposition algorithm developed by Borodin, Razborov, and Smolensky [7]. Their algorithm decomposes a (nondeterministic) read-k-times BP into a set of BPs with a small number of variables. Next, we provide a satisfiability algorithm for a specific class of Boolean functions that have three properties with parameters a and k: (1) Each function is composed of a disjunction of ka subfunctions. (2) Each variable belongs to at most k subfunctions. (3) Each subfunction has at most n/a variables. Our algorithm that checks the satisfiability of such a function is exponentially faster than a brute-force search.

Now, we analyze the running time of a decomposition algorithm by Borodin, Razborov, and Smolensky [7]. We will use this algorithm as a module to solve the syntactic read-k-times BP SAT in Section 4.1.

▶ **Lemma 2** (Theorem 1 in [7]). *Let B be a (nondeterministic) syntactic read-k-times BP with n variables and size m and represent a Boolean function $f : \{0,1\}^n \to \{0,1\}$, and let a be a positive integer. There is an algorithm that constructs kam^{ka} BPs $\{B_{i,j}\}$ from B, where $i \in [m^{ka}]$ and $j \in [ka]$, such that the following properties hold:*

1. *Let $f_{i,j}$ be the Boolean function represented by $B_{i,j}$. Then,*
$$f = \bigvee_{i\in[m^{ka}]} \bigwedge_{j\in[ka]} f_{i,j}.$$
2. *Let $X_{i,j}$ be the set of variables that appear in $B_{i,j}$. For each i and j, $|X_{i,j}|$ is at most $\lceil n/a \rceil$. For each i, each variable x belongs to at most k sets of $\{X_{i,j}\}_{j=1,\dots,ka}$.*

▶ **Lemma 3.** *Given a (nondeterministic) syntactic read-k-times BP B with n variables and size m, the running time of algorithm in Lemma 2 is at most $O(kam^{ka+1})$.*

Proof. Let us observe the construction given in the proof of Theorem 1 in [7]. Let B be a nondeterministic and syntactic read-k-times BP with n variables and size m. For each pair of nodes $(v,w) \in V^2$, $X(v,w)$ denotes the set of all variables that appear in the labels on all possible paths from v to w except for the label of w.

We call a sequence $e_1 := (w_1, v_2), e_2 := (w_2, v_3), \dots, e_\ell := (w_\ell, v_{\ell+1})$ of edges a *trace* if and only if the following properties hold:

(a) For each j with $1 \le j \le \ell+1$, we have $|X(v_j, w_j)| < n/a$.
(b) For each j with $1 \le j \le \ell$, we have $|X(v_j, v_{j+1})| \ge n/a$,

where we set v_1 as the root and $w_{\ell+1}$ as the 1-sink.

Note that any path from r to the 1-sink contains a unique trace. Let $\mathcal{T}$ be the set of all traces. For each trace $T = (e_1 = (w_1, v_2), \dots, e_\ell = (w_\ell, v_{\ell+1})) \in \mathcal{T}$ and $1 \le j \le \ell$, let $B_{T,j}$ be a BP constructed as follows:

1. Prepare the subbranching program $\langle B, v_j, w_j \rangle$, 0-sink, and 1-sink.
2. Create an edge from w_j to the 1-sink with the same label of (w_j, v_{j+1}).
3. If some node v does not have a 0-edge (resp. 1-edge) as an outgoing edge, create a 0-edge (resp. 1-edge) from v to the 0-sink.

Intuitively, $B_{T,j}$ contains all paths from v_j to v_{j+1} through w_j. Note that the index i of the statement corresponds to each trace T. Let $g_{T,j}$ be the function represented by $B_{T,j}$. Then, we have $f = \bigvee_{T\in\mathcal{T}} \bigwedge_{j=1}^{\ell+1} g_{T,j}$. Each function $g_{T,j}$ depends on at most $\lceil n/a \rceil$ variables by property (a). Because B is a syntactic read-k-times BP, for each trace T and each variable x, at most k functions $g_{T,j}$ depend on x. By property (b), we have

$$\sum_{j=1}^{\ell} |X(v_j, v_{j+1})| + |X(v_{\ell+1}, w_{\ell+1})| \ge \frac{n\ell}{a} + |X(v_{\ell+1}, w_{\ell+1})|,$$

where $w_{\ell+1}$ is the 1-sink. Since each variable belongs to at most k subbranching programs, the left-hand side can be bounded above by kn. Then, $\ell \le ka$ holds. Moreover, $\ell = ka$ holds only if $|X(v_{\ell+1}, w_{\ell+1})| = 0$, in which case $g_{T,\ell+1}$ is a constant function. If this constant is 0, then $\bigwedge_{j=1}^{\ell+1} g_{T,j}$ is equal to 0 and we can drop whole terms. If it is 1, we can drop $g_{T,\ell+1}$. Therefore, each conjunction part consists of at most ka terms. The number of traces $|\mathcal{T}|$ is at most m^{ka} because $\ell \le ka$ holds.

The rest of the proof is to analyze the running time of the above construction. First, we find $X(v,w)$ by dynamic programming in $O(m)$ time for each pair of nodes $v, w \in$

V. Then, the running time for enumerating all $X(v,w)$ is at most $O(m^3)$. Using the database of $X(v,w)$, we enumerate all traces by a DFS-like search. The running time for enumerating all traces is at most $O(mka \cdot |\mathcal{T}|)$. For each trace $T \in \mathcal{T}$, we construct at most ka branching programs $B_{t,j}$ in $O(mka)$ time. Then, the total running time is at most $O(m^3) + O(mka \cdot |\mathcal{T}|) = O(kam^{ka+1})$. ◀

Next, we prove the following lemma for the satisfiability algorithm for a specific class of Boolean functions.

▶ **Lemma 4.** *Let a and k be positive integers with $a \leq n$. Suppose that we are given a set of ka functions f_i that satisfy the following properties:*

1. *Each f_i depends on only at most $\lceil n/a \rceil$ variables $X_i \subset X$, i.e., $|X_i| \leq \lceil n/a \rceil$.*
2. *Each variable x belongs to at most k sets X_i.*
3. *Each function f_i can be computed in a time of at most t and a space of at most s.*

Then, there exists a deterministic algorithm for counting the satisfiable assignments of the function $f = \bigwedge_i f_i$ that runs in time $O\left(2^{ka}kan\right) + O(kat) \cdot 2^{\left(1-\frac{2}{4^{k+1}}\left(1-\frac{k}{a}\right)\right)n}$ and space $O(s + kan)$.

Proof. Suppose that n is even. (In the case when n is odd, we also obtain the same result in a similar way.) We can also assume that each variable x belongs to at least one set X_i. If all sets X_i do not contain a variable x, then $\sum_i |X_i| \leq k(n-1)$. This implies that there exists a set X_i such that $|X_i| \leq (n-1)/a < \lceil n/a \rceil$. Then, we can put the variable x into the set X_i while preserving the properties.

Let $\mathcal{F}$ be the family of all subsets of $[ka]$. The size of $\mathcal{F}$, i.e., $|\mathcal{F}|$ is 2^{ka}. For $F \in \mathcal{F}$, $\bar{F}$ is defined as $[ka] \setminus F$. We define the set of variables $V_F := \left(\bigcup_{i\in F} X_i\right) \setminus \left(\bigcup_{i\in \bar{F}} X_i\right)$. The set V_F contains all variables that belong to only $\bigcup_{i\in F} X_i$. By definition, for any $F \in \mathcal{F}$, V_F and $V_{\bar{F}}$ are disjoint.

Find the set $F \in \mathcal{F}$ that maximizes $\min\{|V_F|, |V_{\bar{F}}|\}$ in $|\mathcal{F}| \cdot O(kan) = O(2^{ka}kan)$ time by an exhaustive search for $\mathcal{F}$. Let $Y := \{x_1, \ldots, x_n\} \setminus (V_F \cup V_{\bar{F}})$. Apply some partial assignment α whose support is Y. Then, all $f_i|_\alpha$ for $i \in F$ (resp. $i \in \bar{F}$) depend on only the variables in V_F (resp. $V_{\bar{F}}$). Let A_F be a set of partial assignments α_F such that $S(\alpha_F) = V_F$, and $f_i|_\alpha(\alpha_F) = 1$ holds for all $i \in F$. Similarly, let $A_{\bar{F}}$ be a set of partial assignments $\alpha_{\bar{F}}$ such that $S(\alpha_{\bar{F}}) = V_{\bar{F}}$, and $f_i|_\alpha(\alpha_{\bar{F}}) = 1$ holds for all $i \in \bar{F}$. By an exhaustive search for all partial assignments whose support is V_F (resp. $V_{\bar{F}}$), we count the number of elements of A_F (resp. $A_{\bar{F}}$). Since $f = \bigwedge_{i=1}^{ka} f_i$, for $\alpha_F \in A_F$ and $\alpha_{\bar{F}} \in A_{\bar{F}}$, $f(\alpha \circ \alpha_F \circ \alpha_{\bar{F}}) = 1$ holds. Then, the number of assignments that satisfy f and contain a partial assignment α is $|A_F| \cdot |A_{\bar{F}}|$.

We can count the satisfiable assignments of f by the above operations for all partial assignments α where $S(\alpha) = Y$. For each i, the number of times for computing the function f_i is at most $2^{|Y|} \cdot 2^{\max\{|V_F|,|V_{\bar{F}}|\}}$. Using $|Y| + |V_F| + |V_{\bar{F}}| = n$, we have $2^{|Y|} \cdot 2^{\max\{|V_F|,|V_{\bar{F}}|\}} = 2^{n-\min\{|V_F|,|V_{\bar{F}}|\}}$. Thus, the running time is at most

$$O(2^{ka}kan) + kat \cdot 2^{n-\min\{|V_F|,|V_{\bar{F}}|\}}.$$

Now, we show that $\max_{F\in\mathcal{F}} \min\{|V_F|, |V_{\bar{F}}|\}$ is at least $\frac{2}{4^{k+1}}\left(1-\frac{k}{a}\right)n - \frac{1}{2}$. It follows that the running time of our algorithm is

$$\begin{aligned} O(2^{ka}kan) + kat \cdot 2^{n-\min\{|V_F|,|V_{\bar{F}}|\}} &\leq O\left(2^{ka}kan\right) + kat \cdot 2^{n-\frac{2}{4^{k+1}}\left(1-\frac{k}{a}\right)n+\frac{1}{2}} \\ &= O\left(2^{ka}kan\right) + \sqrt{2}kat \cdot 2^{\left(1-\frac{2}{4^{k+1}}\left(1-\frac{k}{a}\right)\right)n}. \end{aligned}$$

Let S be the set of variables $\{x_1, \ldots, x_{n/2}\}$ and L be the set of variables $\{x_{(n/2)+1}, \ldots, x_n\}$. Now, we define good/bad pairs of variables. This notation is used in the proof of Theorem 6 in [7]. A pair $(x, x') \in S \times L$ is *good* iff there is no $i \in [ka]$ such that $x \in X_i$ and $x' \in X_i$ and *bad* otherwise. For each i, the number of bad pairs for X_i is $|S \cap X_i| \cdot |L \cap X_i|$. Since $|S \cap X_i| + |L \cap X_i| = |X_i| \leq \lceil \frac{n}{a} \rceil$ hold, we have

$$|S \cap X_i| \cdot |L \cap X_i| \leq \frac{1}{4} \cdot \left\lceil \frac{n}{a} \right\rceil^2 .$$

By summing the number of bad pairs for all X_i, the total number of bad pairs is at most $\frac{ka}{4} \cdot \lceil \frac{n}{a} \rceil^2$. Then, using $\lceil \frac{n}{a} \rceil < \frac{n}{a} + 1$ and $a \leq n$, the number of good pairs is at least

$$\frac{n^2}{4} - \frac{ka}{4} \cdot \left\lceil \frac{n}{a} \right\rceil^2 > \frac{1}{4}\left(1 - \frac{k}{a}\right) n^2 - \frac{3k}{4} n.$$

Let us consider that the set $F \in \mathcal{F}$ is chosen uniformly at random. For each good pair $(x, x') \in S \times L$, $\mathbf{Pr}[x \in V_F, x' \in V_{\bar{F}}] \geq 4^{-k}$. Hence,

$$\mathop{\mathbf{E}}_{F \in \mathcal{F}}[|\{(x, x') \mid x \in S \cap V_F, x' \in L \cap V_{\bar{F}}\}|] \geq \frac{1}{4^k}\left[\frac{1}{4}\left(1 - \frac{k}{a}\right) n^2 - \frac{3kn}{4}\right]$$

holds. This implies that there exists a set F such that

$$|V_F| \cdot |V_{\bar{F}}| \geq |S \cap V_F| \cdot |L \cap V_{\bar{F}}| \geq \frac{1}{4^k}\left[\frac{1}{4}\left(1 - \frac{k}{a}\right) n^2 - \frac{3kn}{4}\right].$$

For such a set $F \in \mathcal{F}$, if $|S \cap V_F| \cdot |L \cap V_{\bar{F}}| \geq M$ for some value M, then we have

$$\min\{|S \cap V_F|, |L \cap V_{\bar{F}}|\} \geq \frac{M}{\max\{|S \cap V_F|, |L \cap V_{\bar{F}}|\}} \geq \frac{2M}{n}.$$

We used the fact that $|S \cap V_F|, |L \cap V_{\bar{F}}| \leq \frac{n}{2}$. Since $\min\{|V_F|, |V_{\bar{F}}|\} \geq \min\{|S \cap V_F|, |L \cap V_{\bar{F}}|\}$ holds and n and k are nonnegative integers, we have

$$\begin{aligned} \min\{|V_F|, |V_{\bar{F}}|\} &\geq \frac{2}{4^k n}\left[\frac{1}{4}\left(1 - \frac{k}{a}\right) n^2 - \frac{3kn}{4}\right] \\ &= \frac{2}{4^{k+1}}\left(1 - \frac{k}{a}\right) n - \frac{6k}{4^{k+1}} \\ &> \frac{2}{4^{k+1}}\left(1 - \frac{k}{a}\right) n - \frac{1}{2}. \end{aligned}$$

The last inequality is by the fact that for any $k \geq 1$, $\frac{6k}{4^{k+1}} < \frac{1}{2}$ holds.

We need the computational space $O(kan)$ for finding the set $F \in \mathcal{F}$ that maximizes $\min\{|V_F|, |V_{\bar{F}}|\}$, and $O(s)$ for computing functions f_i. ◀

4 Satisfiability Algorithms for Syntactic Read-k-times BPs

4.1 Satisfiability Algorithm

In this section, we detail our satisfiability algorithm for syntactic read-k-times BPs and analyze its running time. We describe the outline of our algorithm. Our algorithm consists of two steps.

First, applying the decomposition algorithm in Lemma 2 with $a = 2k$, we decompose the input syntactic read-k-times BP B into a disjunction of at most m^{2k^2} BPs. Then, B is satisfiable iff at least one of these decomposed BPs is satisfiable. In addition, each decomposed BP consists of a conjunction of at most $2k^2$ BPs.

Second, we determine the satisfiability of each decomposed BP by checking whether there exists an assignment that satisfies all BPs. Let a decomposed BP be a conjunction of BPs

$\{B_1, \dots, B_\ell\}$, where $\ell \leq 2k^2$. Applying Lemma 4 with $a = 2k$, we count the number of satisfiable assignments that satisfy all BPs.

Repeating the above operations for all decomposed BPs, we can determine the satisfiability of the input B.

▶ **Theorem 5** (Restatement of Theorem 1). *There exists a deterministic and polynomial space algorithm for a nondeterministic and syntactic read-k-times BP SAT with n variables and m edges that runs in time $O\left(\mathrm{poly}(n, m^{k^2}) \cdot 2^{(1-4^{-k-1})n}\right)$.*

Proof. Our algorithm consists of the following two steps: (1) decomposition and (2) satisfiability checking.

[Step 1: Decomposition]
Setting $a = 2k$ in Lemma 2, construct the set of BPs $\{B_{i,j}\}$ from the input B. Let f and $f_{i,j}$ be Boolean functions represented by B and $B_{i,j}$, respectively. Let $X_{i,j}$ be the set of variables that appear in $B_{i,j}$. Then, the following properties hold:

1. $f = \bigvee_{i \in [m^{2k^2}]} \bigwedge_{j \in [2k^2]} f_{i,j}$.
2. For each i and j, $|X_{i,j}|$ is at most $\lceil \frac{n}{2k} \rceil$. For each i, each variable x belongs to at most k sets of $\{X_{i,j}\}_{j=1,\dots,2k^2}$.

The computational time required in Step 1 is at most $O\left(2k^2 m^{2k^2+1}\right)$.

[Step 2: Satisfiability Checking]
In order to check the satisfiability of B, we check whether there exists an assignment that satisfies all branching programs $B_{i,1}, \dots, B_{i,2k^2}$ for each $i \in [m^{2k^2}]$. Let us consider a fixed i. We denote $B_{i,j}$, $f_{i,j}$, and $X_{i,j}$ simply by B_j, f_j, and X_j, respectively. Note that each function f_j can be computed in $O(m)$ time and $O(m)$ space by simulating the computation of B_j.

Our goal in this step is to determine whether there is an assignment that satisfies all f_j for $j \in [2k^2]$. By applying Lemma 4 and setting $a = 2k$, $t = O(m)$, and $s = O(m)$, we count the satisfiable assignments that satisfy all f_j in a time of at most

$$O\left(2^{2k^2} k^2 n\right) + O(k^2 m) \cdot 2^{(1-\frac{1}{4k+1})n}.$$

Therefore, the running time of Step 2 is at most

$$m^{2k^2} \cdot \left\{O\left(2^{2k^2} k^2 n\right) + O(k^2 m) \cdot 2^{\left(1-\frac{1}{4k+1}\right)n}\right\} = \mathrm{poly}\left(n, m^{k^2}\right) \cdot 2^{\left(1-\frac{1}{4k+1}\right)n}.$$

Combining the analyses of Step 1 and Step 2, the running time of our algorithm is at most

$$O\left(2k^2 m^{2k^2+1}\right) + \mathrm{poly}\left(n, m^{k^2}\right) \cdot 2^{\left(1-\frac{1}{4k+1}\right)n} = \mathrm{poly}\left(n, m^{k^2}\right) \cdot 2^{\left(1-\frac{1}{4k+1}\right)n}.$$

Note that if a given B is a deterministic and syntactic read-k-times BP, then any satisfiable assignment of B satisfies only one conjunction part of the decomposed BPs. Then, the number of satisfiable assignments of B is equal to the sum of the results of Step 2. ◀

References

1 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 375–388, 2016.

2 Vikraman Arvind and Rainer Schuler. The quantum query complexity of 0-1 knapsack and associated claw problems. In *Proceedings of the 14th International Symposium on Algorithms and Computation (ISAAC)*, pages 168–177, 2003.

3 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

4 Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#AC^0$ SAT. In *Proceedings of the 27th Conference on Computational Complexity (CCC)*, pages 117–125, 2012.

5 Paul Beame, T. S. Jayram, and Michael E. Saks. Time-space tradeoffs for branching programs. *J. Comput. Syst. Sci.*, 63(4):542–572, 2001.

6 Beate Bollig, Martin Sauerhoff, Detlef Sieling, and Ingo Wegener. Hierarchy theorems for $k$obdds and $k$ibdds. *Theoretical Computer Science*, 205(1-2):45–60, 1998.

7 Allan Borodin, Alexander A. Razborov, and Roman Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993.

8 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 252–260, 2006.

9 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Revised Selected Papers from the 4th International Workshop on Parameterized and Exact Computation*, volume 5917 of *LNCS*, pages 75–85, 2009.

10 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 262–273, 2014.

11 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small De Morgan formulas. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II*, pages 165–176, 2014.

12 Evgeny Dantsin, Edward A. Hirsch, and Alexander Wolpert. Algorithms for SAT based on search in Hamming balls. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 141–151, 2004.

13 Evgeny Dantsin, Edward A. Hirsch, and Alexander Wolpert. Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms. In *Proceedings of the 6th International Conference on Algorithms and Complexity (CIAC)*, pages 60–68, 2006.

14 Edward A. Hirsch. Exact algorithms for general CNF SAT. In *Encyclopedia of Algorithms*. 2008.

15 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 961–972, 2012.

16 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 479–488, 2013.

17 Stasys Jukna. A note on read-k times branching programs. *ITA*, 29(1):75–83, 1995.

18 Atsuki Nagao, Kazuhisa Seto, and Junichi Teruyama. A moderately exponential time algorithm for k-IBDD satisfiability. In *Proceedings of Algorithms and Data Structures - 14th International Symposium (WADS)*, pages 554–565, 2015.

19 Pavel Pudlák. Satisfiability - algorithms and logic. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 129–141, 1998.

20 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 183–192, 2010.

21 Martin Sauerhoff. Lower bounds for randomized read-k-times branching programs. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 105–115, 1998.

22 Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, 54(1):40–44, 2005.

23 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013.

24 Avishay Tal. #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, 22(114), 2015.

25 Jayram S. Thathachar. On separating the read-k-times branching program hierarchy. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 653–662, 1998.

26 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

27 Ryan Williams. Nonuniform ACC circuit lower bounds. *Journal of ACM*, 61(1):2:1–2:32, 2014.

Fully Dynamic Connectivity Oracles under General Vertex Updates

Kengo Nakamura

Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
kengo_nakamura@mist.i.u-tokyo.ac.jp

Abstract

We study the following dynamic graph problem: given an undirected graph G, we maintain a connectivity oracle between any two vertices in G under any on-line sequence of vertex deletions and insertions with incident edges. We propose two algorithms for this problem: an amortized update time deterministic one and a worst case update time Monte Carlo one. Both of them allow an arbitrary number of new vertices to insert. The update time complexity of the former algorithm is no worse than the existing algorithms, which allow only limited number of vertices to insert. Moreover, for relatively dense graphs, we can expect that the update time bound of the former algorithm meets a lower bound, and that of the latter algorithm can be seen as a substantial improvement of the existing result by introducing randomization.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Dynamic Graph, Connectivity, Depth-First Search

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.59

1 Introduction

In this paper, we consider the *dynamic graph connectivity problem.* Given an undirected graph G, the goal for this problem is to build a data structure which can process an on-line sequence of *graph updates* and *queries.* Here for the query $C(u, v)$, the data structure should answer whether there is a path between two vertices u and v in G. There are some variants for this problem with respect to the kinds of operations allowed as the graph updates.

- *Dynamic subgraph connectivity* (DSGC): a binary status is associated with each vertex in G, and we can switch it between "on" and "off". The query is to answer whether there is a path between two vertices in the subgraph of G induced by the "on" vertices.
- *Fully dynamic graph connectivity under edge updates* (FGCE): we can delete an existing edge e from G and insert a new edge e' to G.
- *Fully dynamic graph connectivity under general vertex updates* (FGCV): we can delete an existing vertex w from G, and insert a new vertex v and its incident edges to G.

In this paper, we study the FGCV problem.

Among these three problems, the FGCV problem is the most general framework when we focus on vertex updates. First, an FGCV data structure allows us to insert new vertices, while a DSGC data structure does not. Second, the FGCV problem can be somewhat solved by an FGCE data structure as follows, but there is a limitation on the number of new vertices. In the preprocessing, we add some isolated vertices to G. Then when a new vertex insertion occurs, we select one of the isolated vertices and regard it as the new vertex. Since single vertex update amounts to $O(n)$ edge updates (insertions or deletions) for a graph with n vertices and m edges, the FGCE data structure can process vertex updates. However, in

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 59; pp. 59:1–59:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Comparison of the vertex update time for the (fully) dynamic graph connectivity. “A”, “M”, and “D” in the first column mean that the corresponding rows show the time complexity of amortized update time deterministic, worst case update time Monte Carlo, and worst case update time deterministic algorithms, respectively. The query time is all $O(\log n)$ or $o(\log n)$.

	FGCE *		FGCV	
A	$O(n\frac{\log^2 n}{\log\log n})$	[14]	$O(\sqrt{m}\log^{1.25} n + l\frac{\log^2 n}{\log\log n} + n)$	**X**
M	$O(n\log^5 n)$	[10]	$O(\sqrt{ml}\log^{2.75} n + n)$	**Y**
D	$O(n\sqrt{\frac{n(\log\log n)^2}{\log n}})$	[11]	$O(\sqrt{mn}\log^{1.25} n)$	[12]

* They are multiplied by n since we focus on vertex updates.

this approach we cannot insert an arbitrary number of new vertices. On the other hand, an FGCV data structure can solve the DSGC and FGCE problems (in FGCE setting, one edge update can be converted to two vertex updates).

The FGCE problem is well-studied for years, and various kinds of algorithms for this problem were developed even recently, e.g. an amortized update time deterministic one [14], a worst case update time Monte Carlo one [10], and a worst case update time deterministic one [11]. There were also these kinds of algorithms for the DSGC problem [6, 8, 7]. However, there exist almost no FGCV algorithms which allow us to insert an arbitrary number of new vertices. The only exception is the algorithm of Baswana et al. [2], which maintains a depth-first search (DFS) tree of undirected graphs. Their worst case deterministic update time is recently improved by Nakamura and Sadakane [12]. The comparison of the “vertex” update time of these algorithms is shown in Table 1. Here the update time of the FGCE algorithm is multiplied by n since single vertex update amounts to $O(n)$ edge updates. Note that the update time for the DSGC algorithms [6, 8, 7] is omitted since they have $O(m^{\alpha}\mathrm{polylog}(n))$ query time with $\alpha \geq 1/5$. This is much slower than $O(\log n)$, which is the upper bound for the query time of the algorithms in Table 1.

1.1 Our Results

We develop two data structures for the FCGV problem, both of which allow us to insert an arbitrary number of new vertices. One is an amortized update time deterministic algorithm (algorithm X), and the other is a worst case update time Monte Carlo algorithm (algorithm Y). Both algorithms X and Y have a query time of $O(\log n)$. Their time bounds for single vertex update are shown in the right column of Table 1.

Our time bounds in Table 1 depend on l, that is, the number of leaves of a *DFS forest* (a spanning forest generated by DFS) of G at some point. Both algorithms X and Y internally rebuild a DFS forest of G periodically, and l is in fact the number of leaves of it. Since $l \leq n$, algorithm X can solve the FGCV problem no slower than using the FGCE data structure [14]. Indeed, we can expect $l \ll n$ for relatively dense graphs as described in Sect. 7.

For algorithm X, its update time complexity becomes $O(n)$ if $l = O(n/\log^2 n)$ (unless $m = \Omega(n^2/\log^{2.5} n)$), which is a firm lower bound since the size of input incident edges around the inserted vertex may become $\Theta(n)$. In addition to this, both algorithms X and Y permit G to have some edges initially, while the amortized update time FGCE algorithm [14] assumes G has no edges initially. In summary, the advantages of using algorithm X over using amortized update time FGCE data structure [14] directly is as follows.

- Algorithm X allows us to insert an arbitrary number of new vertices.
- Algorithm X permits G to have some edges initially.

- The update time complexity of algorithm X is no slower than using [14] directly. For relatively dense graphs it is expected to become $O(n)$ which is a firm lower bound.

For algorithm Y, if $l \ll n$, the time bound $O(\sqrt{ml} \cdot \mathrm{polylog}(n))$ can be seen as a considerable improvement of that of [12] by introducing randomization. Moreover, in Sect. 7 it is shown that under ER model [9], which is a popular model of random graphs, the time bound becomes $O(n \log^{3.25} n)$ with high probability, which is faster than using the Monte Carlo FGCE data structure [10] directly. Again note that algorithm Y allows us to insert an arbitrary number of vertices while the Monte Carlo FGCE data structure [10] does not.

Our work can be summarized as follows. Our algorithms use a *disjoint tree partitioning*, which is used in the dynamic DFS algorithm of Baswana et al. [2], and the FGCE data structures ([14] for algorithm X, [10] for algorithm Y). First, we develop an efficient method to maintain disjoint tree partitioning (Sect. 3.1), which reduces the update time when the number of incident edges around the new vertex is small. Second, we define some queries on the graph and show an efficient way to solve them (Sect. 4). We believe these queries are of independent interest. Third, we find a good property of the disjoint tree partitioning for the amortized time complexity analysis (Lemma 2), and develop an algorithm which fully adopts this property (Sect. 5). Lastly, we develop a method to convert the amortized update time algorithm to a worst case update time one (Sect. 6). Note that this kind of technique is also employed in various dynamic graph algorithms such as dynamic DFS [2, 12] and dynamic all-pairs shortest paths [1], but in our situation we need some additional considerations.

2 Preliminaries

Throughout this paper, n and m denote the numbers of vertices and edges of a graph, respectively. We use $\log(\cdot)$ as the base-2 logarithm; the natural logarithm is denoted by $\ln(\cdot)$. Note that they differ only by a constant factor, thus $\ln x = \Theta(\log x)$.

Given a spanning forest T of an undirected graph G each tree in which is a rooted tree, the parent vertex of a vertex v is denoted by $par(v)$. A subtree τ of T is said to be *hanging from* a path p if the root r of τ satisfies both $r \notin p$ and $par(r) \in p$. Two vertices x and y are said to have *ancestor-descendant relation* if $x = y$, x is an ancestor of y, or y is an ancestor of x. A path p in T is said to be an *ancestor-descendant path* if the endpoints of p have ancestor-descendant relation. A spanning forest T of G is a *DFS forest* of G iff each tree in T is a DFS tree of the corresponding connected component of G. The number of leaves of a DFS forest T is the sum of that of each tree in T.

The DFS tree satisfies the following property. Let G be a connected undirected graph and T be a rooted spanning tree of G. Then T is a DFS tree of G, iff every edge in G connects two vertices which have ancestor-descendant relation. We call this *DFS property*.

3 Disjoint Tree Partitioning

In this section, we refer to a *disjoint tree partitioning* [2], and develop an efficient way to maintain this partition. The disjoint tree partitioning is defined as follows.

▶ **Definition 1** ([2]). Given a DFS forest T of an undirected graph G and a set U of vertices, the forest $T - U$ obtained by deleting the vertices in U from T is considered. Then the *disjoint tree partitioning* of $T - U$ is a partition of $T - U$ into a set $\mathcal{P}$ of ancestor-descendant paths in T with $|\mathcal{P}| \leq |U|$ and a set $\mathcal{T}$ of subtrees of T.

From now we abbreviate disjoint tree partitioning as DTP.

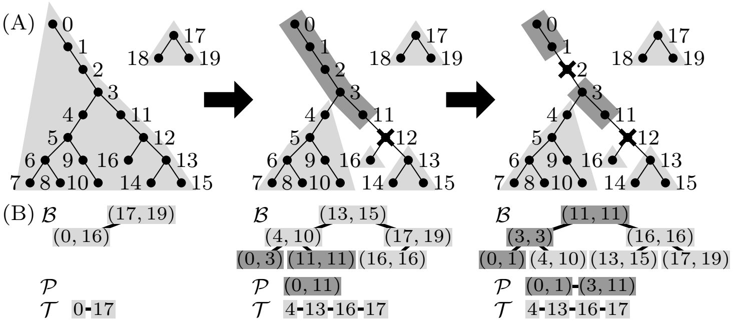


Figure 1 (A) An example of the building process of DTP. Subtrees $\tau \in \mathcal{T}$ are colored with light gray and ancestor-descendant paths $p \in \mathcal{P}$ with dark gray. (B) Corresponding binary trees $\mathcal{B}$ and lists $\mathcal{P}$ and $\mathcal{T}$. Here the pointers from $\mathcal{B}$ to $\mathcal{P}$ or $\mathcal{T}$ are omitted.

In [2] the way to construct DTP is also given. First, if $U = \emptyset$ then we set $\mathcal{P} = \emptyset$ and add all DFS trees in the DFS forest T to $\mathcal{T}$. Next, the DTP of $T - (U \cup \{v\})$ can be obtained by modifying the DTP of $T - U$ as follows. If $v \in \exists p \in \mathcal{P}$ we remove p from $\mathcal{P}$ and add (at most) two paths obtained by deleting v from p to $\mathcal{P}$. Otherwise if $v \in \exists \tau \in \mathcal{T}$, we remove τ from $\mathcal{T}$ and add a path p' from $par(v)$ to the root of τ to $\mathcal{P}$. Then we add all subtrees hanging from v or p' to $\mathcal{T}$. An example of this process is shown in Fig. 1(A). Note that since the number of paths in $\mathcal{P}$ is increased by at most one during each operation, $|\mathcal{P}| \leq |U|$ holds. This operation takes $O(n)$ time for each v, thus the DTP of $T - U$ can be calculated one by one in total $O(|U|n)$ time [2].

3.1 More Efficient Construction

Now we show a more efficient method of maintaining DTP we develop. First, if T is connected, a heavy-light (HL) decomposition [13] of T is calculated, and the order $\mathcal{L}$ of vertices is decided according to the pre-order traversal of T, such that for the first time a vertex v is visited, the next vertex to visit is one that is directly connected with a heavy edge derived from the HL decomposition. Then the vertices of T are numbered from 0 to $n - 1$ according to $\mathcal{L}$; the vertex id of v is denoted by $f(v)$. If T is disconnected, we calculate the order of vertices for each DFS tree in the same way and vertices are numbered by consecutive integers from 0 to $n - 1$. Note that this numbering originates in the dynamic DFS algorithm of Baswana et al. [2], but they utilize this in order to solve some other queries on G. An example of this numbering is shown in Fig. 1(A).

The important point is that the vertices of $\tau \in \mathcal{T}$ occupy single interval in $\mathcal{L}$ since $\mathcal{L}$ is a pre-order traversal of T, and those of $p \in \mathcal{P}$ occupy $O(\log n)$ intervals thanks to HL decomposition. Now we maintain these intervals by a balanced binary search tree $\mathcal{B}$. Here the key of each element is the lower endpoint of its interval. We can say $|\mathcal{B}| \leq n$ and $|\mathcal{B}| = O(|\mathcal{T}| + |\mathcal{P}| \log n)$. Besides this, $\mathcal{P}$ and $\mathcal{T}$ are retained by lists; each $p \in \mathcal{P}$ is expressed by a pair of its endpoints and each $\tau \in \mathcal{T}$ by its root. Here all vertices are stored as the vertex id $f(\cdot)$. Additionally, we add a pointer from each element in $\mathcal{B}$ to the corresponding $x \in \mathcal{P} \cup \mathcal{T}$. Examples of $\mathcal{B}$ and the lists are shown in Fig. 1(B).

Thanks to $\mathcal{B}$, we can efficiently update DTP when a vertex v is deleted. First, search $f(v)$ in $\mathcal{B}$ and detect $p \in \mathcal{P}$ or $\tau \in \mathcal{T}$ which contains v, which takes $O(\log |\mathcal{B}|) = O(\log n)$ time. Then if $v \in \exists p \in \mathcal{P}$, remove p from $\mathcal{P}$ and corresponding intervals from $\mathcal{B}$, and add at most two paths obtained by deleting v from p to $\mathcal{P}$ and corresponding intervals to $\mathcal{B}$. These processes take $O(\log^2 n)$ time, since they amount to $O(\log n)$ deletions and insertions of elements on $\mathcal{B}$. If $v \in \exists\tau \in \mathcal{T}$, first remove τ from $\mathcal{T}$ and a corresponding interval from $\mathcal{B}$. Then while traversing a path p' from $par(v)$ to the root of τ, add subtrees hanging from v or p' to $\mathcal{T}$ and corresponding intervals to $\mathcal{B}$. Finally, add p' to $\mathcal{P}$ and corresponding $O(\log n)$ intervals to $\mathcal{B}$. These take $O(\log^2 n + |p'| + \delta \log n)$ time, where δ is the number of hanging subtrees. In fact, we can limit the sum of $|p'|$ and δ as follows.

▶ **Lemma 2.** *Given a graph G, its DFS forest T and a set of vertices $U = \{v_1, \dots, v_{|U|}\}$, suppose the DTP of $T - \{v_1, \dots, v_i\}$ $(i = 1, \dots, |U|)$ is calculated one by one by the above process. In calculating the DTP of $T - \{v_1, \dots, v_i\}$ by modifying that of $T - \{v_1, \dots, v_{i-1}\}$, if $v_i \in \exists\tau \in \mathcal{T}$, let p'_i be the traversed path (i.e. the path from $par(v_i)$ to the root of τ) and δ_i be the number of hanging subtrees from p'_i or v_i. (if $v_i \in \exists p \in \mathcal{P}$ set $p'_i = \emptyset$ and $\delta_i = 0$). Then $\sum_{i=1}^{|U|} |p'_i| \leq n$ and $\sum_{i=1}^{|U|} \delta_i \leq l + |U|$ hold, where l is the number of leaves of T.*

Proof. For any vertex v, once v is contained in some p'_i, v is always contained in one of the paths in $\mathcal{P}$ until deleted. This means that v cannot be contained in more than one of p'_i. Then $\sum_{i=1}^{|U|} |p'_i| \leq n$ holds. Next, it can be pointed out that $|\mathcal{T}|$ cannot be more than l at any time, since every $\tau \in \mathcal{T}$ has at least one distinct leaf of T. In the process of calculating DTP, $|\mathcal{T}|$ increases by 0 if $v_i \in \exists p \in \mathcal{P}$ or $\delta_i - 1$ if $v_i \in \exists\tau \in \mathcal{T}$. Therefore $\sum_{i=1}^{|U|}(\delta_i - 1) \leq l$. ◀

Note that in the preprocessing, the HL decomposition can be calculated in $O(n)$ time and the initialization of DTP can be done in $O(t)$ time, where $t \leq l$ is the number of connected components of T. Hence we can immediately obtain the following result from Lemma 2.

▶ **Lemma 3.** *By the process described above, the DTP of $T - U$ can be calculated one by one in total $O(|U| \log^2 n + n + l \log n)$ time.*

4 Queries on the Disjoint Tree Partitioning

In this section, we define some queries related to the DTP and show an efficient solution for them. We consider the following queries Q and Q'.

▶ **Definition 4.** An undirected graph G and its DFS forest T are given. Then for any subtree τ of T and ancestor-descendant path p in T, $Q(\tau, p)$ returns one of the edges in G which directly connect τ and p if exists, or $\emptyset$ otherwise. Here we assume τ and p have no common vertices. Similarly, for any two disjoint ancestor-descendant paths p_1, p_2 in T, $Q'(p_1, p_2)$ returns one of the edges in G which directly connect p_1 and p_2 if exists, or $\emptyset$ otherwise.

The motivation to consider these queries is as follows. Roughly speaking, our algorithms proposed later treat paths $p \in \mathcal{P}$ and subtrees $\tau \in \mathcal{T}$ derived from the DTP of $T - U$ as virtual vertices and maintain a data structure to answer connectivity queries among them. Therefore for any distinct $\tau_1, \tau_2 \in \mathcal{T}$ and $p_1, p_2 \in \mathcal{P}$ it is important to judge quickly whether there are some edges in $G - U$ between p_1 and p_2 or between p_1 and τ_1. Here it is noted that thanks to DFS property, there are no edges in $G - U$ between τ_1 and τ_2.

Indeed, the query very similar to $Q(\tau, p)$ is utilized in the dynamic DFS algorithm by Baswana et al. [2], and is revealed to be efficiently solved with the vertex numbering described in Sect. 3 and the *orthogonal range search problem* [2, 12].

▶ **Definition 5.** On grid points in a 2-dimensional plane, k points are given. Then for any rectangular region $R = [x_1, x_2] \times [y_1, y_2]$, the *orthogonal range one reporting query* returns one of the points within R if exists, or $\emptyset$ otherwise. We abbreviate it as ORR query.

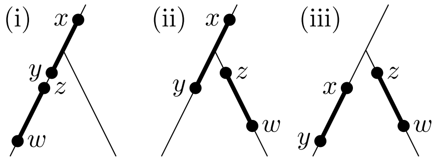


Figure 2 The possible configurations of two ancestor-descendant paths in T.

The queries $Q(\tau, p)$ and $Q'(p_1, p_2)$ can be converted to the ORR query in the following way. First, the vertices of T are numbered from 0 to $n-1$ in the same way as Sect. 3. Second, we consider a grid $\mathcal{G}$ and, for each edge (v, w) of G, put two points on the coordinates $(f(v), f(w))$ and $(f(w), f(v))$ in $\mathcal{G}$. This is equivalent to consider the adjacency matrix of G, therefore $2m$ points are placed. Now a careful case analysis shows the following results. Since Lemma 6 is almost the same as what is proved in [12], we only show the proof of Lemma 7.

▶ **Lemma 6** ([12]). *For any subtree τ of T and ancestor-descendant path p in T, the query $Q(\tau, p)$ can be answered by solving single ORR query on $\mathcal{G}$.*

▶ **Lemma 7.** *For any ancestor-descendant paths p_1, p_2 in T, the query $Q'(p_1, p_2)$ can be answered by solving $O(\log n)$ ORR queries on $\mathcal{G}$.*

Proof. Let x, y be the endpoints of p_1 with $f(x) \le f(y)$ and z, w be those of p_2 with $f(z) \le f(w)$. W.l.o.g. we can assume $f(x) < f(z)$. Due to the HL decomposition, the vertices of p_2 occupy $O(\log n)$ intervals $[a_1, b_1], \dots, [a_k, b_k]$ in the vertex id. Now we assume that p_1 and p_2 are in the same connected component in G. Then there are three patterns on the configuration of p_1 and p_2 as drawn in Fig. 2, and two patterns on the vertex id: (a) $f(x) \le f(y) < f(z) \le f(w)$ and (b) $f(x) < f(z) \le f(w) < f(y)$.

When (a) holds, the answer for $Q'(p_1, p_2)$ can be obtained by solving ORR queries on $\mathcal{G}$ with $R = [f(x), f(y)] \times [a_i, b_i]$ for $i = 1, \dots, k$ and combining these results. The inequality (a) can appear in all configurations in Fig. 2. In (i), it may be that $[f(x), f(y)]$ contains some branches forking from p_1, but it makes no problem since there are no edges between these branches and p_2 thanks to DFS property. The same argument can be applied to (ii). In (iii), the answer for $Q'(p_1, p_2)$ is $\emptyset$ due to DFS property, and each ORR query also returns $\emptyset$. Note that even if p_1 and p_2 are in different connected components in G, (a) holds and the above procedure returns $\emptyset$ correctly. When (b) holds, the answer can be obtained in a similar way except that the rectangles are $R = [f(x), f(\mathrm{LCA}(y, z))] \times [a_i, b_i]$, where $\mathrm{LCA}(y, z)$ is the lowest common ancestor of y and z in T. The inequality (b) can appear in only (ii). In (ii), all edges between p_1 and p_2 are indeed between the path from x to $\mathrm{LCA}(y, z)$ and p_2, and again it does not matter $[f(x), f(\mathrm{LCA}(y, z))]$ contains some branches forking from p_1. Note that the LCA query can be solved in $O(1)$ time with a data structure constructed in $O(n)$ time [5]. This construction time is absorbed in the cost of HL decomposition. ◀

Recently, Belazzougui and Puglisi [4] proved that an ORR query with k points in a rank space can be solved in $O(\log^{\varepsilon} k)$ time with a data structure of $O(k)$ space constructed in $O(k\sqrt{\log k})$ time. With a standard conversion between a rank space and a general grid via bit vectors (see e.g. [12]), we can apply it to Lemma 6 and 7, and obtain the following lemma.

▶ **Lemma 8.** *The queries $Q(\tau, p)$ and $Q'(p_1, p_2)$ can be solved in $O(\log^{\varepsilon} n)$ time and $O(\log^{1+\varepsilon} n)$ time for arbitrary $0 < \varepsilon < 1$, respectively, with a data structure of $O(m)$ space constructed in $O(m\sqrt{\log n})$ time.*

5 Amortized Update Time Algorithm

In this section, we show an amortized update time FGCV algorithm. First we give an overview of our algorithm. As described in Sect. 4, paths $p \in \mathcal{P}$ and subtrees $\tau \in \mathcal{T}$ derived from the DTP of $T - U$ are treated as virtual vertices. If we deal with only deletion of vertices, all we have to do is maintain a connectivity oracle (i.e. a data structure to answer connectivity queries) among them. However, in the fully dynamic setting we deal with insertion of vertices and their incident edges. Thus we also treat each inserted vertex as a virtual vertex, i.e. we maintain a connectivity oracle among $\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$, where $\mathcal{V}$ is a set of inserted vertices. Then our amortized update time algorithm can be described as follows. First, perform DFS on G to build a DFS forest T and initialize the DTP of T, an FGCE data structure (a connectivity oracle) $\mathcal{C}$, and the data structure of Lemma 8 to solve Q and Q'. Second, for the first $\Delta(\leq n)$ vertex updates, if vertex insertion occurs then insert the new vertex to $\mathcal{V}$ and update $\mathcal{C}$, otherwise if vertex deletion occurs then update $\mathcal{V}$ (if the deleted vertex is in $\mathcal{V}$) or the DTP (otherwise) and also update $\mathcal{C}$. Third, when Δ vertex updates are processed, again perform DFS on G to rebuild the DFS forest T and reinitialize the DTP, the connectivity oracle, and the data structure in Lemma 8, which is used for the next Δ updates. In summary, we initialize data structures periodically after every Δ updates.

We proceed to the detailed description of the initialization. In the initialization, we have to construct the FGCE data structure $\mathcal{C}$. From now we call the vertex in $\mathcal{C}$ *node* to avoid confusion. Since in the edge update setting we cannot change the number of nodes, we must decide at first the number of nodes $\mathcal{C}$ has. Because $\mathcal{C}$ maintains connectivity among $\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$, it suffices to prepare M nodes for $\mathcal{C}$ where M is an upper bound of $|\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}|$ while Δ updates are being processed. The following lemma ensures us that $l + \Delta$ nodes are sufficient. Note that $\mathcal{C}$ is initialized to have no edges between any nodes since at first $\mathcal{P} = \mathcal{V} = \emptyset$ and each $\tau \in \mathcal{T}$ is indeed a connected component of G.

▶ **Lemma 9.** *While Δ updates are being processed from the initialization, $|\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}|$ is not more than $l + \Delta$ at any time, where l is the number of leaves of T.*

Proof. It suffices to show that $|\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}|$ is not more than $l + \Delta$ “when” Δ updates are processed. Let Δ_D and Δ_I be the numbers of deleted and inserted vertices during Δ updates, respectively. First, it is already shown that $|\mathcal{T}| \leq l$ at any time. Second, from the definition of DTP, $|\mathcal{P}| \leq \Delta_D$. Finally, $|\mathcal{V}| \leq \Delta_I$ holds trivially (the case $|\mathcal{V}| < \Delta_I$ occurs when some inserted vertices are deleted). Then $|\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}| \leq l + \Delta_D + \Delta_I = l + \Delta$. ◀

Since each element in $\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$ can be deleted, we may have to reuse the nodes of deleted elements when new elements are created. This can be addressed by numbering the nodes in $\mathcal{C}$, storing the corresponding node id for each $p \in \mathcal{P}$, $\tau \in \mathcal{T}$ and $v \in \mathcal{V}$, and maintaining the unused nodes by list. We assume this node recycling runs in background, and for simplicity, the node in $\mathcal{C}$ representing $x \in \mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$ is denoted by $C(x)$.

We next describe the update procedure. Let n be the number of vertices of G at the initialization. Since the initial vertices are numbered from 0 to $n - 1$, the newly inserted vertices, i.e. the vertices in $\mathcal{V}$, are numbered one by one from n.

First we describe how to update the data structures when vertex insertion occurs. At this time the list $\mathcal{A}$ of vertices the newly inserted vertex v is incident with is given. Let $k(\geq n)$ be the vertex id of v. First, convert each element of $\mathcal{A}$ to the vertex id $f(\cdot)$ and store in an array $A[k][\cdot]$ sorted in ascending order. A is an adjacency list for the newly inserted vertices. If $\mathcal{A}$ has a vertex $u \in \mathcal{V}$, append k to $A[f(u)]$, and connect $C(v)$ with $C(u)$. Next, for each $x \in \mathcal{P} \cup \mathcal{T}$ judge whether v is incident with x and connect $C(v)$ with $C(x)$ if so.

These judgments can be performed by traversing $\mathcal{B}$ while scanning $A[k]$ from left to right, since $A[k]$ is sorted. For each interval $[a,b]$ in $\mathcal{B}$, if $[a,b]$ contains some elements in $A[k]$ then we mark the corresponding $x \in \mathcal{P} \cup \mathcal{T}$ incident with v. If $y \in \mathcal{P} \cup \mathcal{T}$ is not marked when the traversal of $\mathcal{B}$ finishes, y is not incident with v.

Next we describe what to do when the deletion of a vertex w occurs. Let $k = f(w)$. The vertex u with $f(u) = x$ is denoted by $f^{-1}(x)$. First, if $w \in \mathcal{V}$, i.e. $k \geq n$, then undo what is performed when inserting w. More specifically, first disconnect $C(f^{-1}(A[k][i]))$ from $C(w)$ for each i such that $A[k][i] \geq n$, then judge whether w is incident with each $x \in \mathcal{T} \cup \mathcal{P}$ and disconnect $C(x)$ from $C(w)$ if so. This judgment can be done in the same way as described above. Next, if $w \notin \mathcal{V}$, detect $y \in \mathcal{P} \cup \mathcal{T}$ which contains w by a search on $\mathcal{B}$, and then update DTP and $\mathcal{C}$ simultaneously as described later. When the DTP is updated as in Sect. 3, some of the following four operations may occur: adding a path p to $\mathcal{P}$, removing p from $\mathcal{P}$, adding a subtree τ to $\mathcal{T}$, and removing τ from $\mathcal{T}$.

Now we focus on a path p, and show how to judge whether $x \in \mathcal{P} \cup \mathcal{T} \cup \mathcal{V} \setminus \{p\}$ is incident with p, i.e. there are some edges between x and p. For each $u \in \mathcal{V}$ we can judge it by the following way. Let $[a_1, b_1], \ldots, [a_k, b_k]$ be the intervals the vertices of p occupy in the vertex id, and $\mathrm{lb}(A[i], j)$ be the smallest element in $A[i]$ which is not less than j, which can be obtained by a binary search on $A[i]$. Then p is incident with u iff there exists i such that $\mathrm{lb}(A[f(u)], a_i) \leq b_i$. For each $p' \in \mathcal{P}$ and $\tau' \in \mathcal{T}$ we can judge the incidence by the queries $Q'(p', p)$ and $Q(\tau', p)$, respectively. Using these judging frameworks, we can update $\mathcal{C}$ when the addition or removal of p occurs: for each $x \in \mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$ incident with p, disconnect $C(x)$ from $C(p)$ when p is removed from $\mathcal{P}$, or connect $C(x)$ with $C(p)$ when p is added to $\mathcal{P}$.

We can cope with the addition or removal of a subtree τ in a similar way. Let $[a,b]$ be the interval the vertices of τ occupy in the vertex id. Then τ is incident with $u \in \mathcal{V}$ iff $\mathrm{lb}(A[f(u)], a) \leq b$. For each $p' \in \mathcal{P}$, we can judge whether p' is incident with τ by the query $Q(\tau, p')$. Again note that τ is not incident with any $\tau' \in \mathcal{T} \setminus \{\tau\}$ due to DFS property. The update procedure for $\mathcal{C}$ is the same: for each $x \in \mathcal{P} \cup \mathcal{V}$ incident with τ, disconnect $C(x)$ from $C(\tau)$ when τ is removed from $\mathcal{T}$, or connect $C(x)$ with $C(\tau)$ when τ is added to $\mathcal{T}$.

Finally we show how to answer the connectivity query between v and w. First we detect $x \in \mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$ containing v. Even if $v \notin \mathcal{V}$ we can determine $x \in \mathcal{P} \cup \mathcal{T}$ by searching the interval which contains $f(v)$ on $\mathcal{B}$. The same argument can be applied to the conversion from w to $y \in \mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$. If $x = y$ then v and w are obviously connected, otherwise the answer can be obtained by querying on $\mathcal{C}$ whether $C(x)$ and $C(y)$ are connected.

5.1 Time Complexity Analysis

We proceed to the time complexity analysis of this algorithm. In our analysis, we use the FGCE data structure proposed by Wulff-Nilsen [14] as $\mathcal{C}$, which has $O(\log^2 k / \log\log k)$ amortized update time and $O(\log k / \log\log k)$ query time for a graph with k nodes. Since $k = l + \Delta \leq n + n$ as in Lemma 9, these are bounded by $O(T_u)$ amortized time for update and $O(T_q)$ time for query, with $T_u = \log^2 n / \log\log n$ and $T_q = \log n / \log\log n$. First of all, the query time of our algorithm is $O(\log n)$, since a search on $\mathcal{B}$ takes $O(\log |\mathcal{B}|) = O(\log n)$ time and a query on $\mathcal{C}$ takes $O(T_q) = o(\log n)$ time. From now the update time is considered.

First, we consider the time consumed by the (periodic) initialization, which is amortized over Δ updates. The data structure in Lemma 8 can be built in $O(m\sqrt{\log n})$ time and $\mathcal{C}$ in $O(k \log k) = O((l+\Delta) \log n)$ time (though the initialization cost of $\mathcal{C}$ is not explicitly described in [14], we prove that for a graph with k nodes and no edges $\mathcal{C}$ can be initialized in $O(k \log k)$ time). From Lemma 3, the total time of maintaining DTP is $O(\Delta \log^2 n + n + l \log n)$.

Next, we focus on the vertex insertion. Let m_v be the number of incident vertices of the newly inserted vertex v, i.e. the number of newly inserted edges. Sorting these incident vertices takes $O(m_v \log m_v) = O(m_v \log n)$ time, or $O(n)$ time by bucket sort. Traversing $\mathcal{B}$ while scanning $A[k]$ takes $O(m_v + |\mathcal{B}|) = O(m_v + |\mathcal{T}| + |\mathcal{P}| \log n)$ time. Connecting $C(v)$ with some $C(x)$ occurs at most $|\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}|$ times, thus updating $\mathcal{C}$ takes at most $O((|\mathcal{P}| + |\mathcal{T}| + |\mathcal{V}|)T_u)$ time.

Finally we consider the deletion of a vertex w. There are three cases, namely, $w \in \mathcal{V}$, $w \in \exists p \in \mathcal{P}$ and $w \in \exists \tau \in \mathcal{T}$. If $w \in \mathcal{V}$, the time complexity is almost the same as that of vertex insertion, except that sorting incident vertices is not needed. If not, we can detect $y \in \mathcal{P} \cup \mathcal{T}$ which contains v in $O(\log |\mathcal{B}|) = O(\log n)$ time. Then if $y = p \in \mathcal{P}$, one path is removed from $\mathcal{P}$ and at most two paths are added to $\mathcal{P}$. For each removal or addition of a path, judging the incidence takes $O(\log^2 n)$ time for each $u \in \mathcal{V}$ (since this amounts to at most $O(\log n)$ binary searches on $A[f(u)]$), $O(\log^{1+\varepsilon} n)$ time for each $p' \in \mathcal{P}$, and $O(\log^{\varepsilon} n)$ time for each $\tau' \in \mathcal{T}$ (Lemma 8). Updating $\mathcal{C}$ takes at most $O((|\mathcal{P}| + |\mathcal{T}| + |\mathcal{V}|)T_u)$ time. Then the total cost is bounded by $O((|\mathcal{P}| + |\mathcal{T}|)T_u + |\mathcal{V}| \log^2 n)$ time. The most complicated case is $y = \tau \in \mathcal{T}$. In this case, one subtree is removed from $\mathcal{T}$, one path is added to $\mathcal{P}$, and δ_w subtrees are added to $\mathcal{T}$, where δ_w is the number of hanging subtrees as described in Sect. 3. Here judging the incidence between $\tau \in \mathcal{T}$ and each $u \in \mathcal{V}$ takes $O(\log n)$ time, since this amounts to one binary search on $A[f(u)]$. Then a similar analysis shows that the total cost is bounded by $O((|\mathcal{P}| + |\mathcal{T}|)T_u + |\mathcal{V}| \log^2 n + \delta_w(|\mathcal{P}| + |\mathcal{V}|)T_u)$.

Now we sum up all of the costs described above. The most crucial point is that we can amortize the sum of δ_w over Δ updates by Lemma 2: $\sum_w \delta_w \leq l + \Delta$. Since $|\mathcal{P}| \leq \Delta$, $|\mathcal{T}| \leq l$ and $|\mathcal{V}| \leq \Delta$, the amortized cost for the update procedure (other than initialization) over Δ updates is bounded by $O(lT_u + \Delta \log^2 n + \min\{\overline{m} \log n, n\})$ per update, where $\overline{m} = \sum_v m_v / \Delta$ is the average number of newly inserted edges per update. The overall amortized update time is obtained by adding the initialization cost divided by Δ to it. Some terms are absorbed in lT_u and $\Delta \log^2 n$ terms and we obtain the following bound: $O((n + m\sqrt{\log n})/\Delta + \Delta \log^2 n + lT_u + \min\{\overline{m} \log n, n\})$. By taking $\Delta = \lceil \sqrt{m} / \log^{0.75} n \rceil$, this bound becomes $O(\sqrt{m} \log^{1.25} n + l \log^2 n / \log \log n + n)$. If $m = \Omega(n/\sqrt{\log n})$, the n/Δ term is absorbed in $m\sqrt{\log n}/\Delta$ term and the last n term becomes $\min\{\overline{m} \log n, n\}$.

▶ **Theorem 10.** *There exists a deterministic fully dynamic connectivity algorithm under general vertex updates such that each update can be processed in amortized $O(\sqrt{m} \log^{1.25} n + l \log^2 n / \log \log n + n)$ time and each query in $O(\log n)$ time, where l is the number of leaves of a DFS forest of G at some point. If $m = \Omega(n/\sqrt{\log n})$, the amortized update time complexity is reduced to $O(\sqrt{m} \log^{1.25} n + l \log^2 n / \log \log n + \min\{\overline{m} \log n, n\})$, where $\overline{m}$ is the average number of newly inserted edges per update.*

6 Worst Case Update Time Algorithm

In this section, we show a worst case update time FGCV algorithm. In our worst case update time algorithm, the procedure for processing graph updates and queries is kept same as the amortized update time algorithm in Sect. 5. We alter the periodic initialization. The principles to achieve "worst case" update time are as follows: (i) to perform simultaneously the processing of graph updates and queries and the initialization of data structures, and (ii) to utilize the data structures built from a DFS forest with "less" number of leaves. The idea (i) is used for various worst case update time dynamic graph algorithms such as dynamic DFS [2, 12] and dynamic all-pairs shortest paths [1]. The idea (ii) is due to the observation that in the amortized update time algorithm, smaller l leads to a better update time bound.

The overview of our algorithm is described as follows. Let $a > 1$ be a positive constant and $\Delta_0, \Delta_1, \dots$ be positive integers (their values are decided later). We virtually divide the sequence of graph updates into *phases*; the first Δ_0 updates are called phase 0, the next Δ_1 updates are called phase 1, and similarly from $(\Delta_0 + \cdots + \Delta_{j-1} + 1)$-st to $(\Delta_0 + \cdots + \Delta_j)$-th updates are called phase j. Let G_j be the graph at the end of phase $(j-1)$. First, given an original undirected graph G, calculate a DFS forest T_0 of G to get the number of leaves l_0 of T_0, initialize the data structure $\mathcal{D}_0$ for G and T_0, and use $\mathcal{D}_0$ for processing updates and queries in phase 0 and 1. Here $\mathcal{D}_0$ is indeed a collection of the DTP of T_0, the FGCE data structure $\mathcal{C}$, and the data structure to solve Q and Q'. Besides this, in phase $j \geq 1$, in the first half (i.e. first $\Delta_j/2$ updates) the followings are performed gradually: calculate a DFS forest T_j of G_j to get the number of leaves l_j of T_j and initialize the data structure $\mathcal{D}_j$ for G_j and T_j. Then if $l_j > a l_{j-1}$ (i.e. the number of leaves of T_j is too much), do nothing other than processing updates and queries in the second half, and in the next phase $(j+1)$ the data structure used in phase j is consecutively utilized for processing updates and queries. If $l_j \leq a l_{j-1}$, in the second half apply the Δ_j vertex updates (from $(\Delta_0 + \cdots + \Delta_{j-1} + 1)$-st to $(\Delta_0 + \cdots + \Delta_j)$-th) on $\mathcal{D}_j$ gradually. This can be done by applying two updates on $\mathcal{D}_j$ during each graph update. In this way $\mathcal{D}_j$ is ready to use for processing updates and queries in the end of phase j, and $\mathcal{D}_j$ is utilized in the next phase $(j+1)$.

6.1 Probability and Time Complexity Analysis

Now we consider the probability of correctness and the update time complexity. Here we use the Monte Carlo FGCE data structure proposed by Kapron et al. [10] as $\mathcal{C}$, which has $O(\log^5 k)$ worst case update time and $O(\log k / \log\log k)$ query time for a graph with k nodes. Their algorithm has only one-sided error: if their algorithm answers "yes" for the query, the answer is always correct, otherwise the answer is correct with probability at least $1 - k^{-c}$ for any fixed constant c. If the data structures are used for processing updates and queries in the same way as Sect. 5, the most time consuming case occurs when a vertex w with $w \in \exists\tau \in \mathcal{T}$ is deleted, which causes $\delta_w \leq l$ subtrees to be added to $\mathcal{T}$. When Δ updates are already processed, $|\mathcal{P}| \leq \Delta$ and $|\mathcal{V}| \leq \Delta$. Therefore the worst case cost for single update in phase j is bounded by $O(l\Delta \log^5 n + n)$ when Δ updates are processed, where l is the number of leaves of a DFS forest the data structures used in phase j are built from. Note that the $O(n)$ term derived from the vertex insertion is also not negligible.

First we consider the probability of correctness. The connectivity oracle by Kapron et al. [10] maintains a spanning forest of the graph internally. Indeed, their oracle guarantees that this spanning forest is maintained correctly with probability at least $1 - k^{-c}$. This means that in our algorithm the spanning forest of a graph with vertex set $\mathcal{P} \cup \mathcal{T} \cup \mathcal{V}$ is maintained correctly in $\mathcal{C}$ with probability at least $1 - k^{-c}$. Later we set $k \geq \sqrt{n}$, then our algorithm answers the query correctly with probability at least $1 - n^{-c/2}$.

Next we consider the time complexity. In the analysis, we assume the number of edges of G is not drastically changed during each phase for simplicity. In other words, let m_j be the number of edges of G_j, then we assume $c_l m_{j-1} \leq m_j \leq c_u m_{j-1}$ for all $j \geq 1$ with some fixed constants c_l and c_u (note that this assumption is also implicitly imposed on the analysis of the dynamic DFS algorithms [2, 12]). We set $a = 4$, $\Delta_0 = \lfloor \sqrt{m/l_0} / \log^{2.25} n \rfloor$ and $\Delta_j = \lfloor \sqrt{m/l_{j-1}} / \log^{2.25} n \rfloor$ $(j \geq 1)$.

In phase $j \geq 1$, performing DFS and initializing the data structure to solve Q and Q' takes $O((m\sqrt{\log n} + n)/(\Delta_j/2)) = O(\sqrt{m l_{j-1}} \log^{2.75} n + n)$ time per update (similar argument can be applied to phase 0). If $l_j \leq 4 l_{j-1}$, in the second half of phase j applying Δ_j updates on $\mathcal{D}_j$ takes $O(l_j \Delta_j \log^5 n + n) = O(\sqrt{m l_{j-1}} \log^{2.75} n + n)$ time per update.

The most important point is how many updates each data structure $\mathcal{D}_t$ is processed. It seems to be difficult to analyze it because even if m does not change drastically during each phase, l may change drastically. However, we can obtain the following lemma.

▶ **Lemma 11.** *During our algorithm, $\mathcal{D}_t$ processes at most $O(\sqrt{m/l_t}/\log^{2.25} n)$ updates.*

Proof. Suppose $\mathcal{D}_t$ $(t \geq 1)$ is used for processing updates and queries from phase $(t+1)$ to $(t+k)$. Then $\mathcal{D}_t$ processes $\Delta_t + \cdots + \Delta_{t+k}$ updates overall. Due to the assumption of the periodic initialization described above, we can say $al_{t-1} \geq l_t < l_{t+1}/a < \cdots < l_{t+k-1}/a^{k-1}$. Therefore $\Delta_{t+i} \leq (\sqrt{m/l_t}/\log^{2.25} n)/2^{i-1}$ $(i = 0, \ldots, k)$ (with $a = 4$). Since the sum of geometric series converges to a constant, $\Delta_t + \cdots + \Delta_{t+k} \leq (\sqrt{m/l_t}/\log^{2.25} n) \cdot (2 + 1 + \cdots + 1/2^{k-1}) = O(\sqrt{m/l_t}/\log^{2.25} n)$. Similar arguments can be applied to $\mathcal{D}_0$. ◀

Then the worst case cost of processing single update with $\mathcal{D}_t$ is bounded by $O(l_t \log^5 n \cdot \sqrt{m/l_t}/\log^{2.25} n + n) = O(\sqrt{ml_t} \log^{2.75} n + n)$. If $\mathcal{D}_t$ $(t \geq 1)$ is used for processing updates and queries in phase $j > t$, we can say $l_t < l_{j-1}/a^{j-1-t}$, so the bound can be written as $O(\sqrt{ml_{j-1}} \log^{2.75} n + n)$. Similar arguments hold for the cases $t = 0$ and $t = j$.

We do not care the cost of $\mathcal{C}$'s initialization, but this does not cause trouble. From the choice of Δ_j, $k = \lceil \max\{l_j + 2\sqrt{m}/\log^{2.25} n, \sqrt{n}\} \rceil$ is enough for $\mathcal{D}_j$. The initialization cost of $\mathcal{C}$ is $O(k \log^4 n)$ [10] since $\mathcal{C}$ is initialized to have no edges as in Sect. 5. Here $l_j \log^4 n/\Delta_j$, $\sqrt{m} \log^{1.75} n/\Delta_j$ and $\sqrt{n} \log^4 n/\Delta_j$ are absorbed in $l_j \Delta_j \log^5 n$, $m\sqrt{\log n}/\Delta_j$ and n/Δ_j, respectively. Similarly, the initialization cost of DTP is also negligible. Overall, it can be said that the worst case update time complexity is $O(\sqrt{ml_0} \log^{2.75} n + n)$ in phase 0 and $O(\sqrt{ml_{j-1}} \log^{2.75} n + n)$ in phase j. Now we obtain the following theorem.

▶ **Theorem 12.** *There exists a Monte Carlo fully dynamic connectivity algorithm under general vertex updates such that each update can be processed in worst case $O(\sqrt{ml} \log^{2.75} n+n)$ time and each query in $O(\log n)$ time, where l is the number of leaves of a DFS forest of G at some point. If this algorithm answers "yes" for the query, the answer is always correct, otherwise correct with probability at least $1 - n^{-c}$ for any fixed constant c.*

7 The Number of Leaves of DFS Forest

In this section, we focus on the value of l, that is, the number of leaves of the DFS forest. Now we state that for relatively dense random graphs $l = o(n)$ holds with high probability.

Here we consider the ER model [9] $G(n, p)$. In a random graph $G(n, p)$ which is an undirected graph with n vertices, for every pair (v, w) of vertices an edge between v and w is added to the graph with probability p independent of other pairs. The average number of edges is $\overline{M} = Np$ with $N = \binom{n}{2} \leq n^2/2$. Recently, Baswana et al. [3] proved the following result, which is about the property of DFS on $G(n, p)$.

▶ **Lemma 13** ([3])**.** *Given a random graph $G(n, p)$ with $p = (\ln n_0 + c)/n_0$ for any integers $n_0 \leq n$ and $c \geq 1$, the DFS on $G(n, p)$ proceeds without moving backward for the first $n - n_0$ vertices with probability at least $1 - 2/e^c$.*

The DFS on a graph can be seen as a sequence of *moving forward* and *moving backward*; if there exist unvisited adjacent vertices then it moves forward, otherwise it moves backward. This lemma implies that with high probability the number of leaves of the DFS forest of $G(n, p)$ is less than n_0 since the first $n - n_0$ vertices are all non-leaf vertices in the DFS forest. The proof of Lemma 13 in [3] is very simple: the probability that the DFS on $G(n, p)$ proceeds without moving backward for the first $n - n_0$ vertices is $\prod_{j=1}^{n-n_0} \{1 - (1 - p)^{n-j}\}$, and this probability is lower bounded by $1 - 2/e^c$ using some elementary inequalities.

In Lemma 13, if we set $\overline{M} = n \ln^{1+\alpha} n/2$ then $n_0 \leq n/\ln^{\alpha} n$ and $\overline{M} = n^{1+\varepsilon}/2$ then $n_0 \leq n^{1-\varepsilon} \ln n$. If $l = O(n/\log^2 n)$, the amortized update time complexity in Theorem 10 becomes $O(n)$ (unless $m = \Omega(n^2/\log^{2.5} n)$), which is a firm lower bound. Simple calculations show that $l \leq n_0 = O(n/\log^2 n)$ is achieved with high probability when $\overline{M} = \Omega(n \log^3 n)$ or $\Omega(n^{1+\varepsilon})$. Moreover, if we set $\overline{M} = gn$ with $g \geq 1$ then $n_0 \leq n \ln n/2g$ in Lemma 13. This means that under ER model, $\overline{M}l \leq \overline{M}n_0 \leq n^2 \ln n/2$ holds with high probability. Therefore the worst case update time complexity in Theorem 12 becomes $O(n \log^{3.25} n)$ (which is faster than $O(n \log^5 n)$ [10]) also with high probability.

It is regrettable that maintaining a connectivity oracle of dense $G(n,p)$ is often useless since $G(n,p)$ with $\overline{M} > n \ln n/2$ is almost surely connected. However, these observations suggest that for a graph with a few parts each of which is dense (e.g. a graph with a few isolated and dense connected components), algorithms X and Y work fast. We think this kind of graph may appear in a social graph with a few isolated or almost isolated communities.

Acknowledgements. The author would like to thank Kunihiko Sadakane for helpful comments and discussion on this work.

References

1 I. Abraham, S. Chechik, and S. Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proc. SODA*, pages 440–452, 2017. doi:10.1137/1.9781611974782.28.

2 S. Baswana, S. R. Chaudhury, K. Choudhary, and S. Khan. Dynamic DFS in undirected graphs: breaking the $O(m)$ barrier. In *Proc. SODA*, pages 730–739, 2016. doi:10.1137/1.9781611974331.ch52.

3 S. Baswana, A. Goel, and S. Khan. Incremental DFS algorithms: a theoretical and experimental study. arXiv:1705.02613, 2017. arXiv:1705.02613.

4 D. Belazzougui and S. J. Puglisi. Range predecessor and Lempel-Ziv parsing. In *Proc. SODA*, pages 2053–2071, 2016. doi:10.1137/1.9781611974331.ch143.

5 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proc. LATIN*, pages 88–94, 2000. doi:10.1007/10719839_9.

6 T. M. Chan, M. Pătraşcu, and L. Roditty. Dynamic connectivity: connecting to networks and geometry. *SIAM J. Comput.*, 40:333–349, 2011. doi:10.1137/090751670.

7 R. Duan. New data structures for subgraph connectivity. In *Proc. ICALP, Part I*, pages 201–212, 2010. doi:10.1007/978-3-642-14165-2_18.

8 R. Duan and L. Zhang. Faster randomized worst-case update time for dynamic subgraph connectivity. In *Proc. WADS*, pages 337–348, 2017. doi:10.1007/978-3-319-62127-2_29.

9 P. Erdős and A. Rényi. On random graphs I. *Publ. Math.*, 6:290–297, 1959.

10 B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proc. SODA*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

11 C. Kejlberg-Rasmussen, T. Kopelowitz, S. Pettie, and M. Thorup. Faster worst case deterministic dynamic connectivity. In *Proc. ESA*, pages 53:1–53:15, 2016. doi:10.4230/LIPIcs.ESA.2016.53.

12 K. Nakamura and K. Sadakane. A space-efficient algorithm for the dynamic DFS problem in undirected graphs. In *Proc. WALCOM*, pages 295–307, 2017. doi:10.1007/978-3-319-53925-6_23.

13 D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26:362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

14 C. Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proc. SODA*, pages 1757–1769, 2013. doi:10.1137/1.9781611973105.126.

Finding Pairwise Intersections of Rectangles in a Query Rectangle*

Eunjin Oh[1] and Hee-Kap Ahn[2]

1 **Department of Computer Sience and Engineering, POSTECH, Korea**
jin9082@postech.ac.kr

2 **Department of Computer Sience and Engineering, POSTECH, Korea**
heekap@postech.ac.kr

Abstract

We consider the following problem: Preprocess a set $\mathcal{S}$ of n axis-parallel boxes in $\mathbb{R}^d$ so that given a query of an axis-parallel box Q in $\mathbb{R}^d$, the pairs of boxes of $\mathcal{S}$ whose intersection intersects the query box can be reported efficiently. For the case that $d = 2$, we present a data structure of size $O(n \log n)$ supporting $O(\log n + k)$ query time, where k is the size of the output. This improves the previously best known result by de Berg et al. which requires $O(\log n \log^* n + k \log n)$ query time using $O(n \log n)$ space. There has been no known result for this problem for higher dimensions, except that for $d = 3$, the best known data structure supports $O(\sqrt{n} + k \log^2 \log^* n)$ query time using $O(n\sqrt{n} \log n)$ space. For a constant $d > 2$, we present a data structure supporting $O(n^{1-\delta} \log^{d-1} n + k \operatorname{polylog} n)$ query time for any constant $1/d \leq \delta < 1$. The size of the data structure is $O(n^{\delta d} \log n)$ if $1/d \leq \delta < 1/2$, or $O(n^{\delta d - 2\delta + 1})$ if $1/2 \leq \delta < 1$.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Geometric data structures, axis-parallel rectangles, intersection

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.60

1 Introduction

Range searching is one of the fundamental problems, which has been studied extensively in computational geometry [2]. Typical problems of this type are formulated as follows. Preprocess a set $\mathcal{I}$ of input geometric objects so that given a query of geometric object Q, the objects in $\mathcal{I} \cap Q$ can be reported or counted efficiently. There are a number of variants of the problem, including checking the emptiness of $\mathcal{I} \cap Q$, finding the minimum (or maximum) weight of the objects in $\mathcal{I} \cap Q$, and computing the sum of the weights of the objects in $\mathcal{I} \cap Q$.

In this paper, we consider a variant of the range searching problem, which is stated as follows. Given a set $\mathcal{S}$ of n axis-parallel boxes in $\mathbb{R}^d$, preprocess $\mathcal{S}$ so that given a query of an axis-parallel box Q in $\mathbb{R}^d$, all the pairs (S, S') of boxes in $\mathcal{S}$ with $S \cap S' \cap Q \neq \emptyset$ can be reported efficiently. The desired running time for the query algorithm is of form $O(f(n) + k(g(n)))$ for some functions $f(n) = o(n)$ and $g(n) = o(n)$, where k is the size of the output. One straightforward way is to compute all boxes of $\mathcal{S}$ intersecting Q and check whether each pair (S, S') of them has their intersection $S \cap S'$ in Q. However, this straightforward algorithm takes $\Omega(n)$ time in the worst case even when $k = 0$.

This problem occurs in a number of real-world applications. For instance, suppose that we are given a collection of personal qualities (or personality traits) of n clients stored in a

* Supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 60; pp. 60:1–60:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

database, each of them is represented as an interval of values. A pair of clients is said to be compatible each other if there is a common subinterval over every quality of them. A typical query on such a collection is composed of a range on each of the qualities, which represents a certain criterion of selecting some compatible pairs of clients that match the query criterion.

If we are allowed to use $\Omega(n^2)$ space in the database, we may precompute all compatible pairs in advance and store them to answer queries efficiently. Otherwise, it is desirable to devise a way of storing the data using less amount of space while the query time remains the same or does not increase much. That is, we need to construct a data structure to answer such a query efficiently in both the query time and the size of the data structure. This is the goal of the problem we study in this paper.

Previous Work. There are a few results on this problem [6, 8, 9]. Consider a simpler problem in which input objects are orthogonal line segments. Orthogonal line segments can be considered as degenerate axis-parallel rectangles. Gupta [8] presented a data structure of size $O(n \log^2 n)$ supporting $O(\log^2 n + k)$ query time for this problem, where k is the size of the output and n is the size of the input. Later, the size of the data structure and the query time were improved to $O(n \log n)$ and $O(\log n + k)$, respectively by Rahul et al. [9].

For axis-parallel rectangles in the plane, de Berg et al. [6] presented a data structure of size $O(n \log n)$ that supports $O(\log n \log^* n + k \log n)$ query time. We observe that their data structure can be improved to support $O(\log n + k \log n)$ query time by simply replacing the range searching algorithm in [10] with the one in [1]. For details, see Section 2.2.1.

The algorithm by de Berg et al. [6] does not extend to higher dimensions directly. Using more observations and techniques, they presented a data structure of size $O(n\sqrt{n} \log n)$ supporting $O(\sqrt{n} + k \log^2 n \log^* n)$ query time in $\mathbb{R}^3$. For fat rectangles, the space and query time are improved to $O(\alpha^3 n \log^2 n)$ and $O(\alpha^2 (k+1) \log^2 \log^* n)$, respectively, where α is the maximum ratio between the lengths of the longest and the shortest edges of input rectangles.

One might be concerned on the preprocessing time as well as the size of the data structure. In this type of problems, however, queries are supposed to be made in a repetitive fashion and the preprocessing time can be seen as being amortized over the queries to be made later on [3]. Therefore, we focus mainly on the space requirement of the data structure and the query time for the problem as other previous works did.

Our Result. In this paper, we first present a data structure of size $O(n \log n)$ for two-dimensional case that supports $O(\log n + k)$ query time. This improves the data structure of de Berg et al. [6]. Recall that our problem is a generalization of the problem studied by Rahul et al. [9]. Although our problem is more general, our data structure with its query algorithm requires the same storage and running time as theirs.

Moreover, our data structure is almost optimal. To see this, observe that our problem can be reduced to the *2D orthogonal range reporting problem.* Given a set $\mathcal{P}$ of points in $\mathcal{R}^2$, the 2D orthogonal range reporting problem asks to preprocess them so that given a query of an axis-parallel rectangle, the points of $\mathcal{P}$ contained in the query rectangle can be reported. To solve this problem using the data structure for our problem, we map each point p in $\mathcal{P}$ to two points lying on p (two degenerate boxes). Then we construct a data structure for our problem on the set of the degenerate boxes for all points in $\mathcal{P}$. The data structure reports the pairs (S, S') of degenerate boxes such that S and S' lie on the same position and are contained in a query rectangle. Therefore, we can answer the 2D orthogonal range reporting problem using the data structure for our problem without increasing the running time. For the 2D orthogonal range reporting problem, it is known that on a pointer machine model, a query time of $O(\text{polylog}\, n + k)$, where k is the size of the output, can only be achieved at the expense of $\Omega(n \log n / \log\log n)$ storage [4]. Moreover, on a pointer machine model, a query time of $o(\log n + k)$ cannot be achieved regardless of the size of the data structure. Therefore, our query time is optimal, and the size of our data structure is almost optimal.

We also consider the problem in higher dimensions $\mathbb{R}^d$. For a constant $d > 2$, we present a data structure that supports $O(n^{1-\delta} \log^{d-1} n + k \log^{d-1} n)$ query time for any constant δ with $1/d \leq \delta < 1$. The size of the data structure is $O(n^{\delta d} \log n)$ if $1/d \leq \delta < 1/2$, or $O(n^{\delta d - 2\delta + 1})$ if $1/2 \leq \delta < 1$. A constant δ shows a trade-off between storage and query time. This is the first result on the problem in higher dimensions.

Preliminaries. We are given a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n axis-parallel boxes (hyperrectangles) in $\mathbb{R}^d$ for some constant $d \geq 2$. For any two boxes $S_i, S_j \in \mathcal{S}$, we use $I(i,j)$ to denote the intersection of S_i and S_j.

Our goal is to preprocess $\mathcal{S}$ so that for a query of an axis-parallel box Q, we can report all pairs (S_i, S_j) of boxes in $\mathcal{S}$ with $I(i,j) \cap Q \neq \emptyset$ efficiently. We use $\mathcal{U}(Q)$ and $k(Q)$ to denote the output and the size of the output for a query Q, respectively. We simply use $\mathcal{U}$ and k to denote $\mathcal{U}(Q)$ and $k(Q)$, respectively, if they are understood in context.

Due to lack of space, some of the proofs and details are omitted.

2 Planar Case

In this section, we consider the problem in the plane, that is, we are given a set $\mathcal{S}$ of n axis-parallel rectangles in the plane. We present a data structure of size $O(n \log n)$ that supports $O(\log n + k)$ query time for queries of axis-parallel rectangles. This improves the previously best known data structure with its query algorithm by de Berg et al. [6]. Their data structure has size of $O(n \log n)$ and supports $O(\log^* n \log n + k \log n)$ query time.

2.1 Configurations of Two Intersecting Rectangles

An axis-parallel rectangle has four sides: the top, bottom, left and right sides. We call the top and bottom sides the *horizontal sides*, and the left and right sides the *vertical sides*.

Consider a side ab of a rectangle $S \in \mathcal{S}$ with endpoints a and b. Let $a'b'$ be the segment on ab such that a' and b' are the points closest to a and b, respectively, among all intersection points of ab with input rectangles other than S. We call $a'b'$ the *stretch* of S on ab. Note that ab has no stretch if ab intersects no rectangles of $\mathcal{S} \setminus \{S\}$. There is at most one stretch for each side of a rectangle in $\mathcal{S}$. Let $\mathcal{S}_\ell$ be the set of all stretches of the rectangles of $\mathcal{S}$.

For any pair (S_i, S_j) of rectangles in $\mathcal{S}$ with $I(i,j) \cap Q \neq \emptyset$, it is not difficult to see that the pair belongs to one of the following three cases: (1) Q is contained in one of the two rectangles of the pair, (2) Q contains a corner of $I(i,j)$, or (3) Q contains no corner of $I(i,j)$. Here we provide another way of describing all the cases in terms of stretches so that the query time can be improved without increasing the size of the data structures compared to the one in [6]. Each of these cases can be rephrased into one or two configurations in Observation 1. More precisely, case (1) corresponds to C1, case (2) corresponds to C2 and C3, and case (3) corresponds to C4 and C5 of Observation 1.

▶ **Observation 1** (Five Configurations of Intersections). *For any pair (S_i, S_j) of rectangles in $\mathcal{S}$ with $I(i,j) \cap Q \neq \emptyset$, one of the followings holds. See Figure 1.*

- *C1. S_i or S_j contains Q.*
- *C2. Q contains an endpoint of a stretch of S_i or S_j.*

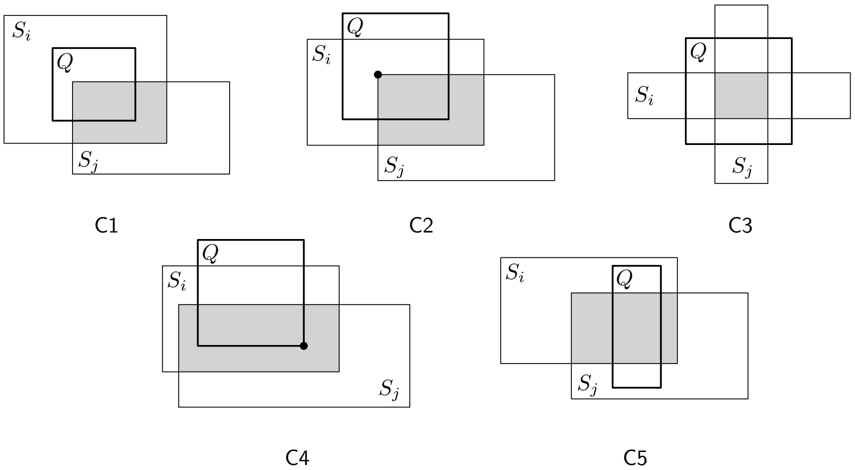


Figure 1 Five configurations of (S_i, S_j) and Q.

- *C3. A stretch of S_i and a stretch of S_j cross Q in different directions.*
- *C4. $I(i,j)$ contains a corner of Q.*
- *C5. $I(i,j)$ and Q cross each other.*

We consider the configurations one by one in our query algorithm. We first report all pairs satisfying C1 (simply, all *C1-pairs*), then we report all pairs satisfying C2 (simply, all *C2-pairs*), and so on. There might be a pair (S_i, S_j) of input rectangles that belongs to more than one configuration. To avoid reporting the same pair more than once, we give a priority order to the configurations such that our algorithm reports a pair exactly once in the configuration of the highest priority. Since there are only five configurations and we can check in constant time whether a pair belongs to a configuration or not, this does not increase the asymptotic time complexity of our algorithm.

2.2 Reporting All Pairs, except C5-pairs

We first show how to construct data structures for finding all pairs (S_i, S_j) of input rectangles with $I(i,j) \cap Q \neq \emptyset$. except C5-pairs. In Section 2.3, we show how to find all C5-pairs.

2.2.1 Data Structures

We construct four data structures for four different problems: the *orthogonal segment intersection problem*, the *point enclosure problem*, the *orthogonal range reporting problem*, and the *rectangle crossing problem*. There has been a fair amount of work on these problems. We observe that the last problem reduces to the 3D orthogonal range reporting problem with a four-sided query box, which has also been studied well. Thus we borrow data structures for these four problems after slightly modifying them to achieve our purpose.

Orthogonal Segment Intersection Problem: SegInt. The *orthogonal segment intersection problem* asks to preprocess horizontal input segments so that given a query of a vertical segment, the horizontal input segments intersected by the query can be computed efficiently. Chazelle [3] gave a data structure called the *hive-graph* to solve this problem efficiently. The hive-graph is a planar orthogonal graph with $O(N)$ cells, each of which has a constant number of edges on its boundary, where N is the number of the input segments.

The query algorithm first finds the cell of the hive-graph containing an endpoint of the query segment and traverses the hive-graph along the query segment from the endpoint to the other endpoint. The edges of the hive-graph encountered during the traversal are the horizontal input segments intersected by the query. In this way, the algorithm finds all such segments in order sorted along the query. The query algorithm takes constant time per output segment, excluding the time for the point location.

In our problem, we construct two hive-graph data structures, one for the horizontal sides of the rectangles of $\mathcal{S}$ and one for the vertical sides of the rectangles of $\mathcal{S}$. Then for each endpoint of the stretches of $\mathcal{S}_\ell$, we find the cells of the hive-graphs that contain the endpoint in the preprocessing phase. This saves the time for point locations in our query algorithm. Specifically, we can find the sides of the rectangles of $\mathcal{S}$ crossed by a stretch ℓ of $\mathcal{S}_\ell$ in the sorted order along ℓ from one endpoint of ℓ. This takes constant time per output side. We denote this data structure by SegInt.

Point Enclosure Problem: PtEnc and EPtEnc. The *point enclosure problem* asks to preprocess input rectangles so that all input rectangles containing a query point can be computed efficiently. Chazelle [3] gave a data structure for this problem. We construct this data structure on $\mathcal{S}$ in the preprocessing time, and denote the data structure by PtEnc. It has size of $O(n)$ and allows us to find all rectangles of $\mathcal{S}$ containing a query rectangle in $O(\log n + K)$ time, where K is the size of the output in this subproblem. Moreover, it allows us to check whether there exists such a rectangle in $O(\log n)$ time.

In our query algorithm, we consider this problem for two different purposes: finding all rectangles of $\mathcal{S}$ containing a *corner* of Q, and finding all rectangles of $\mathcal{S}$ containing an *endpoint* of a stretch of $\mathcal{S}_\ell$. We perform the former task at most four times in our query algorithm since Q has four corners. Thus we simply use PtEnc for this task. However, we will perform the latter task $\Theta(k)$ times in the worst case, which takes $\Omega(k \log n)$ time. Here k is the size of the output in our query algorithm. Note that we have the endpoints of the stretches of $\mathcal{S}_\ell$ in the preprocessing phase, and therefore the latter task can be done in the preprocessing phase.

To do this, we show how the data structure by Chazelle [3] works. Its primary structure is a balanced binary search tree on the rectangles of $\mathcal{S}$ with respect to the x-coordinates of their vertical sides. Each node of the binary search tree corresponds to a vertical line, and it is augmented by the hive-graph on the set of the rectangles of $\mathcal{S}$ intersecting its corresponding vertical line. The query algorithm finds $O(\log n)$ nodes of the binary search tree, and then searches on the hive-graphs associated with the nodes. This takes $O(\log n + K)$ time due to fractional cascading, where K is the size of the output in this subproblem.

This means that we consider $O(\log n)$ hive-graphs and spend $O(\log n)$ time to find the cells containing a query point on the hive-graphs. By finding such cells in the preprocessing phase, we can save the $\log n$ term in the running time of the query algorithm. Note that we need $O(n \log n)$ space to store the cells containing endpoints of the stretches of $\mathcal{S}_\ell$. Then we can find all rectangles of $\mathcal{S}$ containing an endpoint of a stretch of $\mathcal{S}_\ell$ in $O(1+K)$ time, where K is the size of the output. Note that $O(1+K) = O(K)$ since each endpoint is contained in at least two rectangles of $\mathcal{S}$, and thus $K > 1$. We denote this data structure (PtEnc associated with pointers for the endpoints of the stretches) by EPtEnc.

Orthogonal Range Reporting Problem: RecEnc. We want to preprocess all endpoints of the stretches of $\mathcal{S}_\ell$ so that the endpoints contained in a query rectangle can be computed efficiently. Chazelle [3] presented a data structure for this problem that has $O(n \log n / \log\log n)$ size and supports $O(\log n + K)$ query time, where K is the size of the output. We denote this data structure by RecEnc.

Rectangle Crossing Problem: RecCross and RecInt. We want to preprocess the stretches in $\mathcal{S}_\ell$ so that all stretches crossing a query rectangle can be computed efficiently. De Berg et al. [6] also considered this problem. To do this, they reduce to this problem to the orthogonal range reporting problem in three dimensional space as follows. Let $[a,b] \times [c,d]$ be a query rectangle. The query rectangle is crossed by a vertical stretch $x_1 \times [y_1, y_2]$ if and only if $x_1 \in [a,b]$, $y_1 \in [-\infty, c]$, and $y_2 \in [d, \infty]$. Using this observation, they map each vertical stretch $x_1 \times [y_1, y_2]$ to the point (x_1, y_1, y_2) in $\mathbb{R}^3$. Then we can find all vertical stretches crossing the query rectangle by finding all points contained in the orthogonal region $[a,b] \times [-\infty, c] \times [d, \infty]$. Similarly, we can do this for horizontal stretches. However, they did not use the fact that a query is unbounded: it is four-sided in $\mathbb{R}^3$. In this case, we can use a more efficient algorithm given by Afshani et al. [1] instead of the one in [10]. The algorithm by Afshani et al. takes $O(\log n + K)$ time for four-sided query boxes using a data structure of $O(n \log n / \log\log n)$ size, where K is the size of the output. We denote this data structure by RecCross. This data structure is of size $O(n \log n / \log\log n)$ and allows us to find all vertical (or horizontal) stretches of $\mathcal{S}_\ell$ crossing a query rectangle in $O(\log n + K)$ time, where K is the size of the output.

A rectangle S of $\mathcal{S}$ intersects a query rectangle Q if and only if (1) Q crosses a side of S, (2) Q contains a corner of S, or (3) Q is contained in S. To find all rectangles of $\mathcal{S}$ intersecting a query rectangle, we use RecCross for case (1), use RecEnc for case (2), and use PtEnc for case (3). We call the combination of these data structures RecInt. We can find all rectangles of $\mathcal{S}$ intersecting Q in $O(\log n + K)$ time using RecInt, where K is the size of the output in this subproblem.

2.2.2 Query Algorithms.

Assume that we have the data structures of size $O(n \log n)$ described in Section 2.2.1. Then, we can find all pairs (S_i, S_j) of $\mathcal{S}$ with $I(i,j) \cap Q \neq \emptyset$, except C5-pairs, in $O(\log n + k)$ time.

Reporting C1-pairs of Q. We can find the C1-pairs of Q in $O(\log n + k(Q))$ time. A pair of rectangles in $\mathcal{S}$ is a C1-pair of Q if one rectangle in the pair contains all four corners of Q and the other rectangle intersects Q.

We find the rectangles of $\mathcal{S}$ containing all four corners of Q by finding all rectangles of $\mathcal{S}$ containing each corner of Q using PtEnc. Note that there are $O(k(Q) + 1)$ rectangles that contain a corner of Q simply because every pair of the rectangles containing the corner is in $\mathcal{U}(Q)$. (We need "+1" since it is possible that there is just one rectangle containing the corner, but $k(Q)$ is zero.) Thus, we can compute such rectangles in $O(\log n + k(Q))$ time. Let $\mathcal{S}_1$ denote the set of all rectangles containing all four corners of Q.

Let $\mathcal{S}_2$ denote the set of all rectangles intersecting Q. If $\mathcal{S}_1$ is not empty, we find all rectangles of $\mathcal{S}_2$ in $O(\log n + K)$ time using RecInt, where K is the number of such rectangles. Since $\mathcal{S}_1$ is not empty, K is at most $k(Q)$. We report every pair (S_1, S_2) with $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$ as a C1-pair of Q, which takes $O(\log n + k(Q))$ time. It is clear that we report all C1-pairs of Q in this way.

Reporting C2-pairs of Q. We can find the C2-pairs of Q in $O(\log n + k(Q))$ time. A pair of rectangles in $\mathcal{S}$ is a C2-pair of Q if Q contains an endpoint of a stretch ℓ of one of them and the other intersects $\ell \cap Q$. We find all stretches of $\mathcal{S}_\ell$ whose endpoints are in Q in $O(\log n + k(Q))$ time using RecEnc. The number of such stretches is $O(k(Q))$ because each endpoint of the stretches of $\mathcal{S}_\ell$ is contained in at least two rectangles of $\mathcal{S}$ and there are at most four stretches from one rectangle of $\mathcal{S}$.

For each such stretch ℓ, we want to find all rectangles S of $\mathcal{S}$ with $S \cap \ell \cap Q \neq \emptyset$. Such rectangles S satisfy one of the followings: $\ell \cap Q$ is intersected by the boundary of S or $\ell \cap Q$ is contained in S. For the former case, we use SegInt. Starting from the endpoint of ℓ contained in Q, we traverse the hive-graph along ℓ until we escape from Q or we arrive at the other endpoint of ℓ. We find all rectangles S whose sides intersect $\ell \cap Q$ in time linear to the number of such rectangles using SegInt. For the latter case, we compute all rectangles containing the endpoint of ℓ that is also in Q in time linear to the number of such rectangles using EPtEnc. Therefore, for each stretch ℓ with an endpoint in Q, we can find all rectangles of $\mathcal{S}$ intersecting $\ell \cap Q$ in time linear to the number of such rectangles.

By applying this procedure for every stretch with an endpoint in Q, we can find all C2-pairs of Q in $O(k(Q))$ time, excluding the time for finding all such stretches. Therefore, we can compute all C2-pairs of Q in $O(\log n + k(Q))$ time in total.

Reporting C3-pairs of Q. We can find the C3-pairs of Q in $O(\log n + k(Q))$ time. A pair of rectangles in $\mathcal{S}$ is a C3-pair of Q if two stretches, one from each rectangle, cross Q in different directions. Let $\mathcal{S}_v$ be the set of the rectangles of $\mathcal{S}$ whose vertical stretches cross Q. Let $\mathcal{S}_h$ be the set of the rectangles of $\mathcal{S}$ whose horizontal stretches cross Q.

We first check whether $\mathcal{S}_v$ or $\mathcal{S}_h$ is empty in $O(\log n)$ time using RecCross. If both of them are nonempty, we compute $\mathcal{S}_v$ and $\mathcal{S}_h$ in $O(\log n + k(Q))$ time using RecCross. The size of $\mathcal{S}_v$ and $\mathcal{S}_h$ is $O(k(Q))$ since every rectangle of $\mathcal{S}_v$ intersects every rectangle of $\mathcal{S}_h$ in Q. Then we report the pairs (S, S') with $S \in \mathcal{S}_v$ and $S' \in \mathcal{S}_h$ as the C3-pairs in $O(\log n + k(Q))$ time in total.

Reporting C4-pairs of Q. We can report the C4-pairs of Q in $O(\log n + k(Q))$ time. A pair of rectangles in $\mathcal{S}$ is a C4-pair of Q if the intersection of the rectangles contains a corner of Q. We first check whether there exists a rectangle of $\mathcal{S}$ containing a corner of Q in $O(\log n)$ time using PtEnc. Again, the number of the rectangles of $\mathcal{S}$ containing a corner of Q is $O(k(Q))$ as every pair of such rectangles is in $\mathcal{U}(Q)$.

If there exists such a rectangle, we find all such rectangles in $O(\log n + k(Q))$ time using PtEnc. Then we report all pairs consisting of such rectangles. We do this for each of the other corners of Q. Then we can report all C4-pairs in $O(\log n + k(Q))$ time.

2.3 Reporting C5-pairs

We have shown how to find all pairs consisting of two rectangles of $\mathcal{S}$ intersecting each other in Q, except for the C5-pairs. There might be some pairs of rectangles that belong to both C5 and one of the other configurations. As mentioned earlier, this can be checked in constant time per pair of rectangles. Since we use a priority order over the configurations, we assume that they have already been reported by the algorithm for the configurations other than C5.

A pair of rectangles in $\mathcal{S}$ is a C5-pair of a query rectangle Q if the intersection of the rectangles and Q cross each other. In the following, we show how to find and report the C5-pairs of Q not belonging to any other configuration such that the horizontal sides of the intersection intersect the vertical sides of Q. The C5-pairs not belonging to any other configuration such that the vertical sides of the intersection intersect the horizontal sides of Q can be found analogously.

One-dimensional segment tree. We construct a *one-dimensional segment tree* T of $\mathcal{S}$ with respect to the x-axis as follows [5]. The segment tree has a balanced binary search tree on the x-projections of the rectangles of $\mathcal{S}$ as a primary structure. Each node v of the balanced

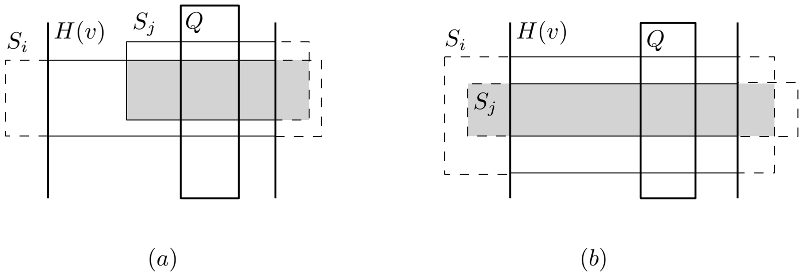


Figure 2 (a) A canonical node v of (i, j, Q). It holds that $S_i \in \mathcal{S}_C(v)$, but $S_j \in \mathcal{S}_B(v)$. (b) A canonical node v of (i, j, Q). It holds that both S_i and S_j are in $\mathcal{S}_C(v)$.

binary search tree corresponds to a closed vertical slab $H(v)$. We say that a rectangle S *crosses* $H(v)$ if S intersects $H(v)$ and no vertical side of S is contained in $H(v)$. Let $\mathcal{S}_C(v)$ be the set of the rectangles of $\mathcal{S}$ that cross $H(v)$ but do not cross $H(u)$ for the parent u of v in T. There are $O(\log n)$ nodes v with $S \in \mathcal{S}_C(v)$. Moreover, the union of $H(v)$'s for all such nodes v contains S. Let $\mathcal{S}_B(v)$ be the set of the rectangles of $\mathcal{S}$ whose left or right side is contained in the interior of $H(v)$. Note that $\mathcal{S}_B(v)$ is empty for every leaf node v. For a rectangle $S \in \mathcal{S}$, there are at most two nodes v of T with $S \in \mathcal{S}_B(v)$ at each level of T, and each such node lies on one of the two paths of T from the root to two leaf nodes w, w' with the left side of S contained in $H(w)$ and the right side of S contained in $H(w')$. We use $\mathcal{S}(v)$ to denote the union of $\mathcal{S}_C(v)$ and $\mathcal{S}_B(v)$. For each node v of T, we store $\mathcal{S}_B(v)$ and $\mathcal{S}_C(v)$. The binary search tree together with the sets $\mathcal{S}_B(\cdot)$ and $\mathcal{S}_C(\cdot)$ forms the segment tree of $\mathcal{S}$.

Canonical nodes of a C5-pair. Consider any C5-pair (S_i, S_j) of Q. Note that there are $O(\log n)$ nodes v of T such that $S_i, S_j \in \mathcal{S}(v)$ and $I(i, j) \cap Q \cap H(v) \neq \emptyset$. If we traverse T and find C5-pairs at each node, then the same pair is found at $O(\log n)$ nodes of T, and therefore the total running time is $\Omega(k \log n)$ for k pairs in the worst case. Instead, we use a number of *canonical nodes* (to be defined below) such that there is a unique canonical node of (i, j, Q) in T for any C5-pair. We will show how to find the canonical nodes and report all C5-pairs efficiently in the subsequent sections. See Figure 2.

▶ **Definition 2.** For a rectangle Q and a pair (S_i, S_j) of the rectangles of $\mathcal{S}$ with $I(i,j) \cap Q \neq \emptyset$, a node v of T is called the *canonical node* of (i, j, Q) if the left side of Q is contained in $H(v)$ and both S_i and S_j are in $\mathcal{S}(v)$ satisfying $S_i \in \mathcal{S}_C(v)$ or $S_j \in \mathcal{S}_C(v)$.

Note that not every canonical node of some triple (i, j, Q) defines a C5-pair of Q, though $I(i, j) \cap Q \neq \emptyset$. However, there is a canonical node of (i, j, Q) in T for each C5-pair of Q.

▶ **Lemma 3.** *For any C5-pair (S_i, S_j) of Q, there is a canonical node of (i, j, Q) in T.*

Proof. Consider the C5-pairs of Q such that the horizontal sides of the intersection intersects the vertical sides of Q. The other cases can be analyzed in a similar way. Let p be the intersection between the left side of Q and the top side of $I(i, j)$. Then there is a path π from the root node to some leaf node u with $p \in H(u)$ in T. Consider a node w in π. Since p lies on the left side of Q, the slab $H(w)$ contains the left side of Q. Moreover, $H(w)$ intersects both S_i and S_j.

We claim that there is a canonical node of (i, j, Q) in π. By the construction of the segment tree, $S_i \in \mathcal{S}_B(v)$ for the root node v and $S_i \notin \mathcal{S}_B(u)$ for the leaf node u of π. Thus, there is a node w_i of π with $S_i \in \mathcal{S}_C(w_i)$. For a node w closer to the root node than w_i,

$S_i \in \mathcal{S}_B(w)$. For a node w' closer to the leaf node than w_i along π, $S_i \notin \mathcal{S}(w')$. This also holds for S_j, so there is a node w_j of π with $S_j \in \mathcal{S}_C(w_j)$. Without loss of generality, we assume that w_i lies between the root node and w_j (including them) along π. Then we have $S_i \in \mathcal{S}_C(w_i)$ and $S_j \in \mathcal{S}(w_i)$. Since w_i is in π, $H(w_i)$ contains the left side of Q. Therefore, s_i is a canonical node of (i, j, Q) in π. ◀

We need the following lemma to bound the total number of canonical nodes for Q over all pairs of rectangles in $\mathcal{S}$ by $O(k(Q))$.

▶ **Lemma 4.** *For any rectangle Q and any pair (S_i, S_j) of the rectangles of $\mathcal{S}$ with $I(i,j) \cap Q \neq \emptyset$, there is at most one canonical node of (i, j, Q) in T.*

▶ **Corollary 5.** *The total number of canonical nodes for a query rectangle Q is $O(k(Q))$.*

Our general strategy is the following. Given a query rectangle Q, we find a set of nodes of the segment tree T that contains the canonical node of (i, j, Q) for every C5-pair (S_i, S_j) not belonging to any other configuration in $O(\log n + k(Q))$ time. The size of this set is $O(k(Q))$. For each such node v, we find all C5-pairs (S_i, S_j) such that v is a canonical node of (i, j, Q) in time linear to the number of the output.

2.3.1 Finding all Canonical Nodes for C5-pairs

In this subsection, we present data structures and their query algorithms to find a set of canonical nodes of (i, j, Q) with $I(i, j) \cap Q \neq \emptyset$ for a query rectangle Q. This set contains all canonical nodes of (i, j, Q) for every C5-pair (S_i, S_j) not belonging to any other configuration. We show how to find such C5-pairs such that the horizontal sides of $I(i, j)$ intersect the vertical sides of Q. Similarly, we can find the other C5-pairs such that the vertical sides of $I(i, j)$ intersect the horizontal sides of Q.

Data Structures. For each node v of T and each rectangle S in $\mathcal{S}(v)$, we trim S as follows. We first remove the parts of S lying outside of $H(v)$. Then we remove the parts of the resulting rectangle that lie outside of the smallest horizontal slab enclosing $\bigcup_{S' \in \mathcal{S}_C(v)} S'$ with $S \cap S' \neq \emptyset$. We call the resulting rectangle the *trimmed rectangle* for (S, v). Let $\mathcal{L}$ be the set of the horizontal sides of all trimmed rectangles for all nodes of T. Note that $|\mathcal{L}| = O(n \log n)$.

We construct the hive-graph on $\mathcal{L}$, which allows us to report all horizontal sides of $\mathcal{L}$ intersecting a query vertical segment ℓ in sorted order along ℓ in $O(\log n + K)$ time, where K is the size of output [3]. Since the size of $\mathcal{L}$ is $O(n \log n)$, the hive-graph has $O(n \log n)$ size. We make each segment in $\mathcal{L}$ to point to the rectangle in $\mathcal{S}$ from which the segment comes.

Query Algorithm. Given a query rectangle Q, our query algorithm finds all sides of $\mathcal{L}$ intersecting the left side of Q using the hive-graph on $\mathcal{L}$. Then for each such side, our query algorithm marks the node of T pointed by the side as a canonical node in $O(\log n + k)$ time due to the following lemmas.

▶ **Lemma 6.** *The query algorithm finds the canonical node of (i, j, Q) for every C5-pair (S_i, S_j) not belonging to any other configuration.*

▶ **Lemma 7.** *The number of the sides of $\mathcal{L}$ intersecting the left side of Q is $O(k(Q))$.*

▶ **Lemma 8.** *Given a query rectangle Q, we can find a set of k nodes of T containing all canonical nodes for C5-pairs not belonging to any other configuration in $O(\log n + k)$ time.*

2.3.2 Handling Each Canonical Node to Find All C5-pairs

Let $\mathcal{V}$ be the set of all nodes we found in Section 2.3.1. For each node $v \in \mathcal{V}$, we show how to find all $\mathsf{C5}$-pairs (S_i, S_j) not belonging to any other configuration such that v is a canonical node of (i, j, Q). Here, we consider only the case that $S_i \in \mathcal{S}_C(v)$ and $S_j \in \mathcal{S}(v)$. The other case that $S_j \in \mathcal{S}_B(v)$ and $S_i \in \mathcal{S}(v)$ can be handled analogously. Moreover, we consider only the $\mathsf{C5}$-pairs such that the horizontal sides of $I(i, j)$ intersect the vertical sides of Q. The other case can be handled analogously.

For each node v, we spend $O(1+k(v))$ time, where $k(v)$ is the number of $\mathsf{C5}$-pairs (S_i, S_j) such that v is an canonical node of (i, j, Q). Once we do this for every node in $\mathcal{V}$, we can obtain all $\mathsf{C5}$-pairs for the canonical nodes of (i, j, Q) not belonging to any other configuration in $O(k)$ time, excluding the time for computing all such canonical nodes.

Data Structures and Preprocessing. While computing the set $\mathcal{V}$ in Section 2.3.1, we obtain all rectangles $S_j \in \mathcal{S}(v)$ for each node $v \in \mathcal{V}$ such that a horizontal side of the trimmed rectangle for (S_j, v) intersects the left side of Q. Moreover, a horizontal side of the trimmed rectangle for (S_j, v) intersects the left side of Q if there is a rectangle $S_i \in \mathcal{S}_C(v)$ such that (S_i, S_j) belongs to $\mathsf{C5}$, but does not belong to any other configuration and the vertical sides of $I(i, j)$ intersect the horizontal sides of Q.

Therefore, we can find all such $\mathsf{C5}$-pairs as follows. For each such rectangle S_j, we find all pairs (S_i, S_j) for $S_i \in \mathcal{S}_C(v)$ such that $\mathsf{y}(S_j)$, $\mathsf{y}(S_i)$ and $\mathsf{y}(Q)$ contain a common point, where $\mathsf{y}(A)$ is the y-projection of a set $A \subseteq \mathbb{R}^2$. Let S'_j be the trimmed rectangle for (S_j, v). Note that for such rectangle, $\mathsf{y}(S'_j)$, $\mathsf{y}(S_i)$ and $\mathsf{y}(Q)$ also contain a common point.

Thus there are two cases: $\mathsf{y}(S'_j) \cap \mathsf{y}(Q)$ contains an endpoint of $\mathsf{y}(S_i)$, or $\mathsf{y}(S'_j) \cap \mathsf{y}(Q)$ is contained in $\mathsf{y}(S_i)$. For the first case, a horizontal side of S_i intersects $Q \cap S'_j$. For the second case, a horizontal side of S'_j is contained in S_i. We maintain two data structures, one for finding the pairs of the first case and the other for finding the pairs of the second case.

The first one is as follows. For each node v of T, we maintain two sorted lists of the rectangles in $\mathcal{S}_C(v)$, one with respect to their top sides and the other with respect to their bottom sides. We make each rectangle S_j in $\mathcal{S}_B(v)$ to point to the rectangle in $\mathcal{S}_C(v)$ with highest bottom side (and highest top side) lying below the top side (and bottom side) of S'_j. Similarly, we make S_j to point to the rectangle in $\mathcal{S}_C(v)$ with lowest top side (and lowest bottom side) lying above the bottom side (and top side) of S'_j.

For the second one, we use a *partially persistent data structure* of a linked list. Once we update a linked list and destroy the old versions, we cannot search any element on an old version any longer. But a partially persistent data structure allows us to access any version at any time by keeping the changes on the linked list. Driscoll et al. [7] presented a general method to make a data structure based on pointers partially persistent. Using their method, we can construct a partially persistent data structure of a linked list.

In our case, the linked list has rectangles in $\mathcal{S}_C(v)$ as its elements. We consider a y-coordinate as a time stamp. A rectangle $S_i \in \mathcal{S}_C(v)$ is appended to the linked list at time t_1 and deleted from the linked list at time t_2, where t_1 is the y-coordinate of the top side of S_i and t_2 is the y-coordinate of the bottom side of S_i. Each insertion and deletion can be done in constant time, which is subsumed in the preprocessing time. For each horizontal side of S'_j, we need an extra pointer that points to the first element of the persistent data structure at time t, where t is the y-coordinate of the horizontal side. The size of the partially persistent data structure is linear to the size of $\mathcal{S}(v)$. Due to the partially persistent data structure and the pointers associated with the horizontal sides of the rectangles of $\mathcal{S}(v)$, we can find all rectangles in S_i containing a horizontal side of S'_j in $O(1 + K)$ time, where K is the number of such S_i's.

Query Algorithm. Given a query rectangle Q, for each node v in $\mathcal{V}$ and each rectangle $S_j \in \mathcal{S}(v)$ such that the trimmed rectangle for (S_j, v) intersects the left side of Q, we want to find all C5-pairs (S_i, S_j) with $S_i \in \mathcal{S}_C(v)$ such that $\mathsf{y}(S_j)$, $\mathsf{y}(S_i)$ and $\mathsf{y}(Q)$ contain a common point. Recall that there are two cases: $\mathsf{y}(S'_j) \cap \mathsf{y}(Q)$ contains an endpoint of $\mathsf{y}(S_i)$, or $\mathsf{y}(S'_j) \cap \mathsf{y}(Q)$ is contained in $\mathsf{y}(S_i)$.

To find the C5-pairs (S_i, S_j) belonging to the first case, we do the followings. We search the sorted list of the rectangles in $\mathcal{S}_C(v)$ with respect to their top sides starting from the rectangle of $\mathcal{S}_C(v)$ with lowest top side lying above the bottom side of S'_j if the bottom side of S'_j intersects Q. Note that we can obtain the starting point using the pointer that the bottom side of S'_j has. We stop searching the sorted list when we reach the top side of S'_j or the top side of Q. Similarly, we search the sorted list of the rectangles in $\mathcal{S}_C(v)$ with respect to their bottom sides starting from the rectangle that the bottom side of S'_j points to until we reach the top side of S'_j or the top side of Q if the top side of S'_j intersects Q. In this way, we can find all rectangles S_i in $\mathcal{S}_C(v)$ belongs to the first case in $O(1 + K)$ time, where K is the number of such rectangles.

To find the C5-pairs (S_i, S_j) belonging to the second case, we do the followings. Recall that a horizontal side ℓ of S'_j is contained in Q. Moreover, ℓ is also contained in S_i since $\mathsf{y}(S'_j) \cap \mathsf{y}(Q)$ is contained in $\mathsf{y}(S_i)$. We search the partially persistent data structure at time t, where t is the y-coordinate of ℓ. Starting from the pointer that ℓ points to, we traverse the linked list at time t. All rectangles we encounter are the rectangles containing ℓ. This takes $O(1 + K')$ time, where K' is the number of such rectangles.

Note that both K and K' are at least $k(v)$, where $k(v)$ is the number of the output pairs (S_i, S_j) in $\mathcal{U}(Q)$ such that the canonical node of (i, j, Q) is v. Therefore, we spend $O(1 + k(v))$ time for each node $v \in \mathcal{V}$. Note that $k(v)$ is at least one for every node $v \in \mathcal{V}$ by the construction of $\mathcal{V}$. Once we do this for every node in $\mathcal{V}$, we can obtain $\mathcal{U}(Q)$ in $O(1 + k(Q))$ time in total.

▶ **Lemma 9.** *Given a query rectangle Q, we can find all C5-pairs in $O(\log n + k(Q))$ time.*

Therefore, we have the following theorem.

▶ **Theorem 10.** *We can construct a data structure of size $O(n \log n)$ on a set $\mathcal{S}$ of n axis-parallel rectangles so that for a query axis-parallel rectangle Q, the pairs (S_i, S_j) of $\mathcal{S}$ with $S_i \cap S_j \cap Q \neq \emptyset$ can be reported in $O(\log n + k)$ time, where k is the size of the output.*

3 Higher Dimensional Case

In this section, we consider a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n axis-parallel boxes (hyperrectangles) in $\mathbb{R}^d$ for a constant $d > 2$. Let $\delta \in \mathbb{R}$ be any constant with $1/d \leq \delta < 1$. We present a data structure that supports $O(n^{1-\delta} \log^{d-1} n + k \operatorname{polylog} n)$ query time. The size of the data structure is $O(n^{\delta d} \log n)$ if $1/d \leq \delta < 1/2$, or $O(n^{\delta d - 2\delta + 1})$ if $1/2 \leq \delta < 1$. There has been no known result for this problem in higher dimensions, except that for $d = 3$, the best known data structure has size of $O(n\sqrt{n} \log n)$ and supports $O(\sqrt{n} + k \log^2 \log^* n)$ query time.

Due to lack of space, we gives only a sketch of our data structure and algorithm. For each index $1 \leq t \leq d$, we construct n^δ intervals on the x_t-axis. Consider the x_t-projections of the x_t-facets of $\mathcal{S}$ and choose every $\lfloor n^{1-\delta} \rfloor$th projection. Then we have n^δ projections that define n^δ intervals. Let T_t be the set of such intervals. A *grid cell* is a d-tuple $(v_1, \ldots, v_d)$ of intervals $v_t \in T_t$ for $1 \leq t \leq d$. Note that there are $O(n^{\delta d})$ grid cells. We define the canonical grid cell of a box B, not necessarily in $\mathcal{S}$ in $\mathbb{R}^d$ to be the grid cell containing the corner of B with minimum x_t-coordinates for all $1 \leq t \leq d$.

We construct a data structure for each interval on the x_t-axis, and store a Boolean value to each grid cell. The total size of the data structures is $O(n^{\delta d}\log n)$ if $0<\delta<1/2$, or $O(n^{\delta d-2\delta+1})$ if $1/2\le\delta<1$.

Given a query box Q, we observe that the canonical grid cell of $I(i,j)$ is contained in Q, or $I(i,j)\cap Q$ intersects a grid cell intersecting the boundary of Q for a pair (S_i,S_j) with $I(i,j)\cap Q\neq\emptyset$. We find all pairs of the first case in $O(k\log^{d-1}n)$ time. For the second case, we reduce this problem to the $(d-1)$-dimensional problem and obtain a recurrence equation on the running time of our algorithm. Then we obtained the following theorem.

▶ **Theorem 11.** *We can construct data structures on a set $\mathcal{S}$ of n axis-parallel boxes in $\mathbb{R}^d$ so that for a query axis-parallel box Q for a constant d, the pairs (S_i,S_j) of $\mathcal{S}$ with $S_i\cap S_j\cap Q\neq\emptyset$ can be reported in $O(n^{1-\delta}\log^{d-1}n+k\log^{d-1}n)$ time for any constant $1/d\le\delta<1$, where k is the size of the output. The size of the data structure is $O(n^{\delta d}\log n)$ if $1/d\le\delta<1/2$, or $O(n^{\delta d-2\delta+1})$ if $1/2\le\delta<1$.*

References

1 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *Proceddings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 149–158, 2009.

2 Pankaj K Agarwal, Jeff Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

3 Bernard Chazelle. Filtering search: A new approach to query answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

4 Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)*, 37(2):200–212, 1990.

5 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, 2008.

6 Mark de Berg, Joachim Gudmundsson, and Ali D. Mehrabi. Finding pairwise intersections inside a query range. In *Proceedings of the 14th Algorithms and Data Structures Symposium (WADS 2015)*, pages 236–248, 2015.

7 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

8 Prosenjit Gupta. Algorithms for range-aggregate query problems involving geometric aggregation operations. In *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC 2005)*, pages 892–901, 2005.

9 Saladi Rahul, Ananda Swarup Das, K. S. Rajan, and Kannan Srinathan. Range-aggregate queries involving geometric aggregation operations. In *Proceedings of the 5th International Workshop on Algorithms and Computation (WALCOM 2011)*, pages 122–133, 2011.

10 Sairam Subramanian and Sridhar Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1995)*, pages 378–387, 1995.

A New Balanced Subdivision of a Simple Polygon for Time-Space Trade-off Algorithms*†

Eunjin Oh[1] **and Hee-Kap Ahn**[2]

1 **Department of Computer Science and Engineering, POSTECH, Korea**
jin9082@postech.ac.kr

2 **Department of Computer Science and Engineering, POSTECH, Korea**
heekap@postech.ac.kr

Abstract

We are given a read-only memory for input and a write-only stream for output. For a positive integer parameter s, an s-workspace algorithm is an algorithm using only $O(s)$ words of workspace in addition to the memory for input. In this paper, we present an $O(n^2/s)$-time s-workspace algorithm for subdividing a simple polygon into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$.

As applications of the subdivision, the previously best known time-space trade-offs for the following three geometric problems are improved immediately: (1) computing the shortest path between two points inside a simple n-gon, (2) computing the shortest path tree from a point inside a simple n-gon, (3) computing a triangulation of a simple n-gon. In addition, we improve the algorithm for the second problem even further.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Time-space trade-off, simple polygon, shortest path, shortest path tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.61

1 Introduction

In the algorithm design for a given task, we seek to achieve an efficient algorithm with respect to the time and space complexities. However, one cannot achieve both goals at the same time in many cases: one has to use more space to achieve a faster algorithm and spend more time to reduce the space consumption of the algorithm. Therefore, one has to make a compromise between the running time and the space consumption, considering the goal of the task and the system resources where the algorithm is performed. With this reason, a number of time-space trade-offs were considered even as early as in 1980s. For example, Frederickson [7] presented optimal time-space trade-offs for sorting and selection problems in 1987. After this work, a significant amount of research has been done for time-space trade-offs in the design of algorithms.

The model we consider in this paper is formally described as follows. An input is given in a read-only memory. For a positive integer parameter s which is determined by users, we are allowed to use $O(s)$ words as workspace in addition to the memory for input. We assume that a word is large enough to store a number and a pointer. While processing input,

[*] A full version of the paper is available at [10], https://arxiv.org/abs/1709.09932.

[†] This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP-2017-0-00905) supervised by the IITP(Institute for Information & communications Technology Promotion)

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 61; pp. 61:1–61:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

we write output to a write-only stream without repetition. An algorithm designed in this setting is called an *s-workspace algorithm*.

Most of previous fundamental algorithms and applications assume that they can use workspace without much constraint in size. Typically, they use workspace of at least the size of input. Although the memory is relatively cheap these days, this is not always the case as the amount of data collected and used by various applications has significantly increased over the last years and the memory resource available in the system is relatively smaller compared to the amount of data they use. This constraint implies some restriction in using workspace for the applications.

We assume that input is given in a *read-only memory* under a *random-access model*. The assumption on the read-only memory has been considered in applications where the input is required to be retained in its original state or more than one program may access the input simultaneously. Many time-space trade-offs for fundamental problems have been studied under this read-only assumption.

In this paper, we consider time-space trade-offs for constructing a few geometric structures inside a simple polygon: the shortest path between two points, the shortest path tree from a point, and a triangulation of a simple polygon. With linear-size workspace, optimal algorithms for these problems are known. The shortest path between two points and the shortest path tree from a point inside a simple n-gon can be computed in $O(n)$ time [8]. A triangulation of a simple n-gon can also be computed in $O(n)$ time [5].

For a positive integer parameter s, the following s-workspace algorithms are known.

- **The shortest path between two points inside a simple polygon:** The first non-trivial s-workspace algorithm for computing shortest paths between any two points in a simple n-gon was given by Asano et al. [2]. Their algorithm consists of two phases. In the first phase, they subdivide the input simple polygon into $O(s)$ subpolygons of complexity $O(n/s)$ in $O(n^2)$ time. In the second phase, they compute the shortest path between the two points in $O(n^2/s)$ time using the subdivision. In the paper (and the talk by Asano in a workshop in honor of his 65th birthday during SoCG 2014), they asked whether the first phase can be avoided and the running time can be improved to $O(n^2/s)$. This problem is still open while there are several partial results.

  Har-Peled [9] presented an s-workspace algorithm which takes $O(n^2/s + n \log s \log^4(n/s))$ expected time. Their algorithm takes $O(n^2/s)$ expected time for the case that $s = O(n/\log^2 n)$. For the case that the input polygon is monotone, Barba et al. [4] presented an s-workspace algorithm which takes $O(n^2/s + (n^2 \log n)/2^s)$ time. Their algorithm takes $O(n^2/s)$ time for $\log\log n \leq s < n$.
- **The shortest path tree from a point inside a simple polygon:** Aronov et al. [1] presented an s-workspace algorithm for computing the shortest path tree from a given point. Their algorithm reports the edges of the shortest path tree without repetition in an arbitrary order in $O((n^2 \log n)/s + n \log s \log^5(n/s))$ expected time.
- **A triangulation of a simple polygon:** Aronov et al. [1] presented an s-workspace algorithm for computing a triangulation of a simple n-gon. Their algorithm returns the edges of a triangulation without repetition in $O(n^2/s + n \log s \log^5 (n/s))$ expected time. Moreover, their algorithm can be modified to report the resulting triangles of a triangulation together with their adjacency information in the same time if $s \geq \log n$.

  For a monotone n-gon, Barba et al. [4] presented an $(s \log_s n)$-workspace algorithm for triangulating the polygon in $O(n \log_s n)$ time for a parameter $s \in \{1, \ldots, n\}$. Later, Asano and Kirkpatrick [3] showed how to reduce the workspace to $O(s)$ words without increasing the running time.

1.1 Our Results

We present an s-workspace algorithm to subdivide a simple polygon with n vertices into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$ in $O(n^2/s)$ deterministic time. We obtain this subdivision in three steps. First, we choose every $\max\{n/s, s\}$th vertex of the simple polygon which we call *partition-vertices*. In the second step, for every pair of consecutive partition-vertices, we choose $O(1)$ vertices which we call *extreme-vertices*. Then we draw the vertical extensions from each partition-vertex and each extreme-vertex, one going upwards and one going downwards, until the extensions escape from the simple polygon. These extensions subdivide the polygon so that each subpolygon has complexity of $O(\max\{n/s, s\})$ or has a spiral-like structure. In the third step, we subdivide each spiral-like structure into subpolygons of complexity $O(\max\{n/s, s\})$. Then we show that the resulting subdivision has the desired complexity.

By using this subdivision method together with new ideas, we improve the running times of the following three problems without increasing the size of the workspace.

- **The shortest path between two points inside a simple polygon:** We can compute the shortest path between any two points inside a simple n-gon in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace. The previously best known s-workspace algorithm [9] takes $O(n^2/s + n \log s \log^4(n/s))$ expected time.
- **The shortest path tree from a point inside a simple polygon:** We can compute the shortest path tree from a given point inside a simple n-gon in $O(n^2/s + (n^2 \log n)/s^c)$ expected time for any constant $c > 0$. The previously best known s-workspace algorithm [1] takes $O((n^2 \log n)/s + n \log s \log^5 (n/s))$ expected time. The algorithm in [1] computes the shortest path between two points as a subprocedure. If one uses our shortest path algorithm for this subprocedure, the algorithm takes $O((n^2 \log n)/s)$ expected time.
- **A triangulation of a simple polygon:** The previously best known s-workspace algorithm [1] takes $O(n^2/s + n \log s \log^4(n/s))$ expected time. This algorithm computes the shortest path between two points as a subprocedure. If their algorithm uses our shortest path algorithm for this subprocedure, it takes $O(n^2/s)$ deterministic time.

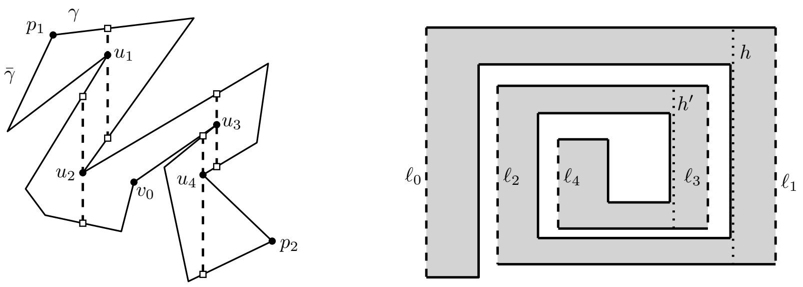
All missing details and proofs can be found in the full version of this paper [10].

2 Preliminaries

Let P be a simple polygon with n vertices. Let $v_0, \ldots, v_{n-1}$ be the vertices of P in clockwise order along ∂P. The vertices of P are stored in a read-only memory in this order. For a subpolygon S of P, we use ∂S to denote the boundary of S and $|S|$ to denote the complexity of S. For any two points p and q in P, we use $\pi(p, q)$ to denote the shortest path between p and q contained in P. We assume that no two distinct vertices of P have the same x-coordinate. We can avoid this assumption by using a shear transformation [6, Chapter 6].

Let v be a vertex of P. We consider two vertical extensions from v, one going upwards and one going downwards, until they escape from P for the first time. A vertical extension from v contains no vertex of P other than v due to the assumption that no two distinct vertices of P have the same x-coordinate. We call the point of ∂P where an extension from v escapes from P for the first time a *foot point* of v. Note that a foot point of a vertex might be the vertex itself. We can compute (report) the foot points of all vertices of P in $O(n^2/s)$ time using $O(s)$ words of workspace.

▶ **Lemma 1.** *We can report the foot points of all vertices of P in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.*



(a)

(b)

Figure 1 (a) Two chains γ and $\bar{\gamma}$ connecting two vertices p_1 and p_2. The set $V_\gamma = \{u_2, u_4\}$. The (L,C)-extreme-vertex of γ is u_2, and the (L,CC)-extreme-vertex of γ is u_4. The (R,C)-extreme-vertex of $\bar{\gamma}$ is u_1, and the (R,CC)-extreme-vertex of $\bar{\gamma}$ is u_3. (b) In the third step, we compute h for (ℓ_0, ℓ_1, ℓ_2) and h' for (ℓ_2, ℓ_3, ℓ_4).

In the following, we compute the extensions from some vertices of P. These extensions form a subdivision of P. We call each such extension a *wall*, and each subpolygon in the subdivision a *cell*. Given an edge of a cell, we can traverse the boundary of the cell starting from the given edge in time linear to the complexity of the cell once we store the walls of the subdivision and their endpoints in clockwise order along ∂P in the workspace.

3 Balanced Subdivision of a Simple Polygon

In this section, we present an s-workspace algorithm to subdivide a simple polygon P into $O(\min\{n/s, s\})$ subpolygons of complexity $O(\max\{n/s, s\})$ using $O(\min\{n/s, s\})$ walls. To do this, we first present an s-workspace algorithm to subdivide P into $O(n/\triangle)$ subpolygons of complexity $O(\triangle)$ using $O(n/\triangle)$ walls in $O(n^2/s)$ time, where $\triangle$ is a positive integer with $\min\{n/s, (s \log n)/n\} \leq \triangle \leq n$ which is determined by s. Since $n/s \leq \triangle$, we have $n/\triangle \leq s$. Thus, we can keep all such walls in the workspace of size $O(s)$. We will set the value of $\triangle$ in Theorem 10 so that we can obtain a subdivision of our desired complexity.

The first step: Subdivision by partition-vertices. We first consider every $\triangle$th vertex of P from v_0 in clockwise order, that is, $v_0, v_\triangle, v_{2\triangle}, \ldots, v_{\lfloor n/\triangle \rfloor \triangle}$. We call them *partition-vertices*. The number of partition-vertices is $O(n/\triangle)$. We compute the foot points of each partition-vertex, which can be done for all partition-vertices in $O(n^2/s)$ time in total by Lemma 1. We sort the foot points along ∂P in $O((n/\triangle)\log(n/\triangle))$ time, which is $O(n^2/s)$ by the fact that $\triangle \geq (s \log n)/n$. We store them together with their vertical extensions using $O(n/\triangle) = O(s)$ words of workspace.

The second step: Subdivision by extreme-vertices between two partition-vertices. The (L,C)-*extreme-vertex* and (L,CC)-*extreme-vertex* of a polygonal curve $\gamma \subset \partial P$ are defined as follows. Let V_γ be the set of all vertices of γ both of whose foot points are on $\partial P \setminus \gamma$ and whose extensions lie locally to the *left* of γ. The (L,C)-extreme-vertex (or the (L,CC)-extreme-vertex) of γ is the vertex in V_γ defining the first extension we encounter while we traverse ∂P in clockwise (or counterclockwise) order from v_0. See Figure 1(a). Similarly, we define the (R,C)-*extreme-vertex* and (R,CC)-*extreme-vertex* of γ. In this case, we consider the vertices of γ whose extensions lie locally to the *right* of γ. We simply call the (L,C)-,(L,CC)-,(R,C)- and (R,CC)-extreme-vertices *extreme-vertices* of γ. Note that γ may not have any extreme-vertex.

In the second step, we compute the extreme-vertices of each polygonal curve connecting two consecutive partition-vertices along ∂P and containing no other partition-vertices. Then we have $O(n/\triangle)$ extreme-vertices. We compute the foot points of all extreme-vertices and store them together with their vertical extensions using $O(n/\triangle) = O(s)$ words of workspace in $O(n^2/s)$ time using Lemma 1 and Lemma 2.

▶ **Lemma 2.** *We can find the extreme-vertices of every polygonal curve connecting two consecutive partition-vertices along ∂P and containing no other partition-vertices in $O(n^2/s)$ total time using $O(s)$ words of workspace.*

The third step: Subdivision by a vertex on a chain connecting three extensions. After applying the first and second steps, we obtain the subdivision induced by the extensions from the partition- and extreme-vertices. Let S' be a subpolygon in this subdivision. We will see later in Lemma 4 that S' has the following property: every chain connecting two consecutive extensions along $\partial S'$, except for two of them, has no extreme-vertex. However, it is still possible that S' contains $\omega(1)$ extensions on its boundary. In this case, S' has a spiral-like structure due to the property of S' mentioned above. See Figure 1(b). In the third step, we subdivide each subpolygon further so that every subpolygon has $O(1)$ extensions on its boundary.

The boundary of S' consists of vertical extensions and polygonal chains from ∂P whose endpoints are partition-vertices, extreme-vertices, or their foot points. We treat the upper and lower extensions defined by one partition- or extreme-vertex (more precisely, the union of them) as one vertical extension.

For every triple (ℓ, ℓ', ℓ'') of consecutive vertical extensions on $\partial S'$ such that ℓ, ℓ' and ℓ'' appear on $\partial S'$ in clockwise order, we consider the part (polygonal curve) of $\partial S'$ from ℓ to ℓ'' in clockwise order (excluding ℓ and ℓ''). Let Γ be the set of all such polygonal curves. For every $\gamma \in \Gamma$, we find a vertex $v(\gamma)$ of $\partial S' \setminus \gamma$ such that one of its foot points lies in γ between ℓ and ℓ', and the other foot point lies in γ between ℓ' and ℓ'' if it exists. If there are more than one such vertex $v(\gamma)$, we choose an arbitrary one.

The extensions of $v(\gamma)$ subdivide S' into three subpolygons each of which contains one of ℓ, ℓ' and ℓ''. In other words, the extensions from $v(\gamma)$ *separate* ℓ, ℓ' and ℓ''. We can compute $v(\gamma)$ and their extensions for every $\gamma \in \Gamma$ in $O(|S'|^2/s + \triangle)$ time in total. See Figure 1(b). The sum of $|S'|$ over all subpolygons S' is $O(n)$ and the number of the subpolygons from the second step is $O(n/\triangle)$ since we construct $O(n/\triangle)$ extensions in the first and second steps. Therefore, we do this for all subpolygons in the subdivision from the second step in $O(n^2/s + n) = O(n^2/s)$ time using $O(s)$ words of workspace.

Analysis. We obtained $O(n/\triangle)$ vertical extensions in $O(n^2/s)$ time using $O(s)$ words of workspace. In the following, we show that these vertical extensions subdivide P into $O(n/\triangle)$ subpolygons of complexity $O(\triangle)$. We call this subdivision the *balanced subdivision* of S. For any two points a, b on ∂P, we use $P[a, b]$ to denote the polygonal curve from a to b (including a and b) in clockwise order along ∂P.

We use the technical lemmas (Lemma 3 to Lemma 6) to show that each subpolygon in the final subdivision is incident to $O(1)$ walls and has complexity of $O(\triangle)$. Then we obtain Theorem 11 by setting a parameter $\triangle$.

▶ **Lemma 3.** *Both $P[a_1, v]$ and $P[v, a_2]$ contain partition-vertices for any extension a_1a_2 from a vertex v constructed from any of the three steps such that $P[a_1, a_2]$ contains v.*

Let S be a subpolygon in the final subdivision and S' be the subpolygon in the subdivision from the second step containing S. We again treat two vertical extensions defined by one vertex as one vertical extension. We label the extensions lying on ∂S as follows. Let ℓ_0 be the first extension on ∂S we encounter while we traverse ∂P from v_0 in clockwise order. We let $\ell_1, \ell_2, \ldots, \ell_k$ be the extensions appearing on ∂S in clockwise order along ∂S from ℓ_0. Similarly, we label the extensions lying on $\partial S'$ from ℓ'_0 to $\ell'_{k'}$ along $\partial S'$ in clockwise order such that ℓ'_0 is the first one we encounter while we traverse ∂P from v_0 in clockwise order. Then we have the following lemmas.

▶ **Lemma 4.** *For any $1 \leq i < k'$, let $a_1a_2 = \ell'_i$ and $b_1b_2 = \ell'_{i+1}$ such that a_1, a_2, b_1 and b_2 appear on ∂P (and $\partial S'$) in clockwise order. Then $P[a_2, b_1]$ has no extreme-vertex.*

▶ **Lemma 5.** *For any $1 \leq i < k-1$, one of ℓ_i, ℓ_{i+1} and ℓ_{i+2} is constructed in the third step.*

▶ **Lemma 6.** *The subpolygon S is incident to $O(1)$ extensions constructed in the third step.*

Proof. Consider an extension ℓ incident to S constructed in the third step. Let v be the vertex defining the extension ℓ. The boundary of S' consists of the walls $\ell'_0, \ldots, \ell'_{k'}$ and the polygonal curves connecting two consecutive walls. Let η_i be the polygonal curve of $\partial S'$ connecting ℓ'_i and ℓ'_{i+1} (excluding the walls) for $0 \leq i < k'$, and $\eta_{k'}$ be the polygonal curve connecting $\ell'_{k'}$ and ℓ'_0 (excluding the walls).

We also claim that there exist at most two vertices $u \in \eta_0$ such that the foot points of u are in $\partial S' \setminus \eta_0$ and the extension of u is incident to S. To see this, let $u_1, u_2 \in \eta_0$ be such vertices if they exist. Let h_1 and h_2 be the extensions from u_1 and u_2, respectively, incident to S. One polygonal chain connecting h_1 and h_2 along ∂S (but not containing them in its interior) is contained in η_0 since u_1 and u_2 are in η_0. The other polygonal chain along ∂S does not intersect η_0. This is because the foot points of u_1 and u_2 are not in η_0, and both h_1 and h_2 are incident to S. Therefore, no other vertex in η_0 with foot points in $\partial S \setminus \eta_0$ has extensions incident to S. This proves the claim. The same holds for $\eta_{k'}$.

Therefore, there are at most four extensions on ∂S constructed in the third step: two of them are extensions of vertices of η_0 and the others are extensions of vertices of $\eta_{k'}$. Thus the lemma holds. ◀

Due to Lemma 5 and Lemma 6, the following corollary holds.

▶ **Corollary 7.** *Every subpolygon in the final subdivision is incident to $O(1)$ extensions.*

▶ **Lemma 8.** *Every subpolygon in the final subdivision has complexity of $O(\triangle)$.*

Proof. Consider a subpolygon S in the final subdivision. By Corollary 7, the boundary of S consists of $O(1)$ vertical extensions and $O(1)$ polygonal curves from the boundary of P connecting two consecutive endpoints of vertical extensions of S. Each polygonal curve from the boundary of P contains at most one partition-vertex in its interior. Otherwise, a vertical extension intersecting the interior of S is constructed in the first or second step, which contradicts that S is a subpolygon in the final subdivision. The number of vertices between two consecutive partition-vertices is $O(\triangle)$. Therefore, S has $O(\triangle)$ vertices on its boundary. ◀

Therefore, we have the following lemma.

▶ **Lemma 9.** *Given a simple n-gon and a parameter $\triangle$ with $\min\{n/s, (s \log n)/n\} \leq \triangle \leq n$, we can compute a set of $O(n/\triangle)$ walls which subdivides the polygon into $O(n/\triangle)$ cells of complexity $O(\triangle)$ in $O(n^2/s)$ time using $O(s)$ words of workspace.*

By setting a parameter, we can obtain the following theorem.

▶ **Theorem 10.** *Given a simple n-gon, we can compute a set of $O(\min\{n/s, s\})$ walls which subdivides the polygon into $O(\min\{n/s, s\})$ cells of complexity $O(\max\{n/s, s\})$ in $O(n^2/s)$ time using $O(s)$ words of workspace.*

Proof. We set $\triangle$ to n/s if $s \leq \sqrt{n}$. We set $\triangle$ to s, otherwise. ◀

4 Applications

Comparison with other subdivision methods. There are several subdivision methods which are used for computing the shortest path between two points in the context of time-space trade-offs. Asano et al. [2] presented a subdivision method that subdivides a simple polygon into $O(s)$ subpolygons of complexity $O(n/s)$ using $O(s)$ chords. Then they showed that the shortest path can be computed in $O(n^2/s)$ time using $O(s)$ words of workspace. However, their algorithm takes $O(n^2)$ time to compute such a subdivision, which dominates the overall running time. Thus, computing such a subdivision is a bottleneck of this problem. In fact, in the paper, they asked whether such a subdivision can be computed more efficiently.

Instead of answering this question directly, Har-Peled [9] presented a way to subdivide a simple polygon into subpolygons of slightly different complexity and show that this subdivision has a structural property similar to the one for the subdivision of Asano et al. The subdivision of Har-Peled consists of $O(n/s)$ subpolygons of complexity $O(s)$ using $O(n/s)$ line segments. The number of line segments defining this subdivision is larger than $\Omega(s)$, so they cannot keep the whole subdivision in the workspace. Instead, they gave a procedure to find the subpolygon containing a query point in $O(n + s \log s \log^4(n/s))$ expected time. They showed that one can find the shortest path between two points using this subdivision in a way similar to the algorithm by Asano et al.

The balanced subdivision that we propose has a structural property similar to the ones by Asano et al. and Har-Peled, so we can use our balanced subdivision to compute the shortest path between any two points. Moreover, our subdivision method has several advantages compared to the ones of Asano et al. and Har-Peled: our balanced subdivision can be computed faster than the one by Asano et al., and we can keep the whole subdivision in the workspace unlike the one by Har-Peled. Due to the first advantage, ours can be used to improve a number of algorithms. Due to the second advantage, we can solve a few other application problems. A specific example is to compute the shortest path between a query point and a fixed point after a preprocessing for the fixed point. See Lemma 17.

Computing the shortest path between two points. Given any two points p and q in P, we can report the edges of the shortest path $\pi(p, q)$ in order in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace. This improves the s-workspace randomized algorithm by Har-Peled [9] which takes $O(n^2/s + n \log s \log^4(n/s))$ expected time.

As mentioned earlier, our subdivision method has properties similar to the ones by Asano et al. [2] and Har-Peled [9]. For $s \geq \sqrt{n}$, we have the subdivision consisting of $O(n/s)$ cells of complexity $O(s)$. We use the algorithm by Har-Peled [9]. His algorithm takes $O(n(T(n) + n)/s)$ time, where $T(n)$ is the time for finding the cell containing a query point. In our case, $T(n) = O(n)$, thus we can compute the shortest path between any two points in $O(n^2/s)$ deterministic time.

For $s < \sqrt{n}$, we have the subdivision consisting of $O(s)$ cells of complexity $O(n/s)$. We use the algorithm by Asano et al. [2] that computes the shortest path between any two points assuming that we are given a subdivision consisting of $O(s)$ cells of complexity $O(n/s)$.

▶ **Theorem 11.** *Given any two points in a simple polygon with n vertices, we can compute the shortest path between them in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.*

Computing the shortest path tree from a point. The *shortest path tree* rooted at p is defined to be the union of $\pi(p, v)$ over all vertices v of P. Aronov et al. [1] gave an s-workspace randomized algorithm for computing the shortest path tree rooted at a given point. It takes $O((n^2 \log n)/s + n \log s \log^5 (n/s))$ expected time and uses the algorithm by Har-Peled [9] as a subprocedure. If one uses Theorem 11 instead of Har-Peled's algorithm, the running time is improved to $O((n^2 \log n)/s)$ expected time. In Section 5, we improve this algorithm even further using our balanced subdivision.

Computing a triangulation of a simple polygon. Aronov et al. [1] presented an s-workspace algorithm for computing a triangulation of a simple polygon by using the shortest path algorithm by Har-Peled [9] as a subprocedure. By replacing this algorithm with ours in Theorem 11, we can obtain a triangulation of a simple polygon in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.

▶ **Theorem 12.** *Given a simple polygon with n vertices, we can compute a triangulation of the simple polygon in $O(n^2/s)$ deterministic time using $O(s)$ words of workspace.*

5 Improved Algorithm for Computing the Shortest Path Tree

In this section, we improve the algorithm for computing the shortest path tree from a given point even further to $O(n^2/s + (n^2 \log n)/s^c)$ expected time for an arbitrary constant $c > 0$. We use the following lemma given by Aronov et al. [1].

▶ **Lemma 13** ([1, Lemma 6]). *For any point p in a simple n-gon, we can compute the shortest path tree rooted at p in $O(n^2 \log n)$ expected time using $O(1)$ words of workspace.*

We apply two different algorithms depending on the size of the workspace: $s = O(\sqrt{n})$ or $s = \Omega(\sqrt{n})$. In this paper, we consider the case of $s = O(\sqrt{n})$ only. The other case can be handled analogously. A main difference is that we do not use Theorem 11 and Lemma 13 for $s = \Omega(\sqrt{n})$. Instead, we use the fact that we can store all edges of each cell in the workspace. The details for the case of $s = \Omega(\sqrt{n})$ can be found in the full version of this paper.

Given a point $p \in P$, we want to report all edges of the shortest path tree rooted at p. For every wall a_1a_2 of the balanced subdivision, we first compute the edges of $\pi(p, a_1) \cup \pi(p, a_2)$ crossing some walls in $O(n^2/s^2)$ time in Section 5.1. We show that the number of such edges is $O(s)$ in total. These edges allow us to compute the shortest path $\pi(p, q)$ for any point q of P in $O(n^2/s^2)$ time. We call an edge a *w-edge* if it crosses a wall.

Then we decompose P into subpolygons associated with vertices in Section 5.2. For each subpolygon, we compute the shortest path tree rooted at its associated vertex inside the subpolygon recursively. If a subpolygon satisfies one of stopping conditions (to be defined later), we compute the shortest path tree inside the subpolygon in different ways. Because of the space restriction, we restrict the depth of the recurrence to be a constant instead of applying the procedure recursively until the problem size becomes $O(s)$.

5.1 Computing w-edges

We compute all w-edges of the shortest paths between p and the endpoints of the walls. The following lemma implies that there are $O(s)$ such w-edges. For any three points x, y and z in P, we call a point x' the *junction* of $\pi(x, y)$ and $\pi(x, z)$ if $\pi(x, x')$ is the maximal common path of $\pi(x, y)$ and $\pi(x, z)$.

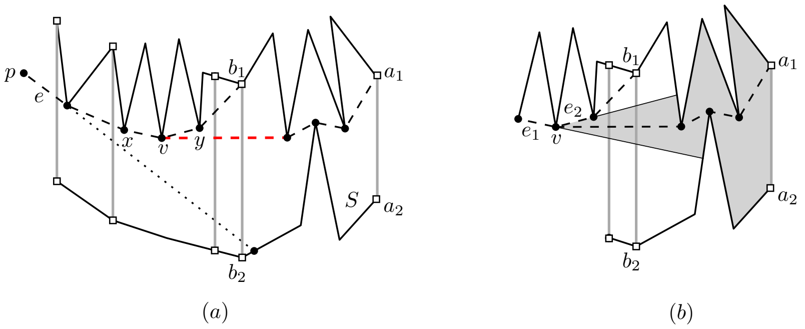


Figure 2 (a) We compute the junction v of $\pi(p, a_1)$ and $\pi(p, b_1)$ by applying binary search on the w-edges of $\pi(p, b_1)$. (b) We extend e_1 and e_2 towards b_1. The gray region contains the edge of $\pi(v, a_1)$ incident to v and has complexity of $O(n/s)$.

▶ **Lemma 14.** *For every wall a_1a_2, there is at most one w-edge of $\pi(p, a_i)$ for $i = 1, 2$ which is not a w-edge of $\pi(p, b) \cup \pi(p, b')$ for any wall bb' crossed by $\pi(p, a_i)$.*

We consider the walls one by one in a specific order and compute such w-edges one by one. To decide the order for considering the walls, we define a *wall-tree* T as follows. Each node α of T corresponds to a wall $d(\alpha)$ of the balanced subdivision of P, except for the root. The root of T corresponds to p and has children each of which corresponds to a wall incident to the cell containing p. A non-root node β of T is the parent of a node α if and only if $d(\beta)$ is the first wall that we encounter during the traversal of $\pi(p, a_1)$ from a_1 for an endpoint a_1 of $d(\alpha)$. We can compute T in $O(n)$ time.

▶ **Lemma 15.** *The wall-tree can be built in $O(n)$ time using $O(s)$ words of workspace.*

After constructing T, we apply depth-first search on T. Let $\mathcal{D}$ be an empty set. When we visit a node α with $a_1a_2 = d(\alpha)$, we compute the w-edges of $\pi(p, a_1) \cup \pi(p, a_2)$ which are not in $\mathcal{D}$ yet, and put them in $\mathcal{D}$. Each w-edge in $\mathcal{D}$ has information on the node of T defining it and the cells of the balanced subdivision containing its endpoints. Due to this information, we can compute the w-edges of $\pi(p, a)$ in order from a in $O(s)$ time for any endpoint a of $d(\alpha)$ and any node α we visited before. Once the traversal is done, $\mathcal{D}$ contains all w-edges in the shortest paths between p and the endpoints of the walls.

We show how to compute the w-edge of $\pi(p, a_1)$ which is not in $\mathcal{D}$ yet. The case for $\pi(p, a_2)$ can be handled analogously. By Lemma 14, there is at most one such edge of $\pi(p, a_1)$. Moreover, by its proof, such an edge is incident to v on $\pi(v, a_1)$. Here, v is the one of the two junctions closer to a_1 than the other among the junction of $\pi(p, b_1)$ and $\pi(p, a_1)$, and the junction of $\pi(p, b_2)$ and $\pi(p, a_1)$, where b_1b_2 is the wall corresponding the parent of α.

Computing junctions. We show how to compute the junction v_1 of $\pi(p, b_1)$ and $\pi(p, a_1)$ in $O(n^2/s^2)$ time assuming that $s = O(\sqrt{n})$. The junction of $\pi(p, b_2)$ and $\pi(p, a_1)$ can be computed analogously. Then we can compute v in the same time.

To do this, we find the w-edges in $\mathcal{D}$ lying on $\pi(p, b_i)$ in order from b_i in $O(s)$ time for $i = 1, 2$ and denote the set of them by $\mathcal{D}(b_i)$. Note that these are the w-edges of $\pi(p, b_i)$. We find two consecutive edges in $\mathcal{D}(b_1)$ containing v_1 between them along $\pi(p, b_1)$ by applying binary search on the edges in $\mathcal{D}(b_1)$.

Given any edge e in $\mathcal{D}(b_1)$, we can determine which side of e along $\pi(p, b_1)$ contains v_1 in $O(n/s)$ time as follows. We first check whether e is also contained in $\pi(p, b_2)$ in constant time using $\mathcal{D}(b_2)$. If so, v_1 is contained in the side of e along $\pi(p, b_1)$ containing b_1. Thus we are done. Otherwise, we extend e towards b_1 until it escapes from S, where S is the cell incident to both a_1a_2 and b_1b_2. See Figure 2(a). Note that the extension crosses b_1b_2 since both $\pi(b_1, v_e)$ and $\pi(b_2, v_e)$ are concave for an endpoint v_e of e. We can compute the point where the extension escapes from S in $O(n/s)$ time by traversing the boundary of S once. If an endpoint of the extension lies on the part of ∂S between a_1 and b_1 not containing a_2, v_1 lies in the side of e containing p along $\pi(p, b_1)$. Otherwise, the junction v_1 is contained in the other side of e. Therefore, we can find two consecutive w-edges in $\mathcal{D}(b_1)$ containing v_1 between them along $\pi(p, b_1)$ in $O((n/s)\log s)$ time since the size of $\mathcal{D}(b_1) = O(s)$.

The edges of $\pi(p, b_1)$ lying between the two consecutive w-edges are contained in the same cell. Let x and y be the endpoints of the two consecutive edges of $\mathcal{D}(b_1)$ contained in the same cell. Then we compute the edges of $\pi(x, y)$ one by one from x to y inside the cell containing x and y. By Theorem 11, we can compute $\pi(x, y)$ in $O(n^2/s^3)$ time since the size of the cell is $O(n/s)$. Here, we use extra $O(s)$ words of workspace for computing $\pi(x, y)$. When the algorithm in Theorem 11 reports an edge f of $\pi(x, y)$, we check which side of f along $\pi(x, y)$ contains v_1 in $O(n/s)$ time as we did before. We do this until we find v_1. This takes $O((n/s)^2)$ time since there are $O(n/s)$ edges in $\pi(x, y)$. Therefore, in total, we can compute the junction v_1 in $O(s + (n/s)\log s + n^2/s^2) = O(n^2/s^2)$ time since $s = O(\sqrt{n})$.

Computing the edge of $\pi(v, a_1)$ incident to the junction v. In the following, we compute the edge of $\pi(v, a_1)$ incident to v. Let e_1 and e_2 be two edges of $\pi(p, b_1)$ incident to v, which can be obtained while we compute v. See Figure 2(b). We extend e_1 and e_2 towards b_1 until they escape from the cell incident to both a_1a_2 and b_1b_2. We consider the subpolygon bounded by the two extensions and containing a_1 on its boundary. Its boundary consists of parts of the boundary of P and three extra line segments: the extensions of e_1 and e_2, and the part of the wall a_1a_2. Thus, the subpolygon can be represented using $O(1)$ words and the number of vertices in the subpolygon is $O(n/s)$. Note that $\pi(v, a_1)$ is contained in the subpolygon. Thus, the edge of $\pi(v, a_1)$ incident to v inside the subpolygon is the edge we want to compute. We can compute it in $O(n^2/s^3)$ time by applying Theorem 11 to this cell.

In summary, we presented a procedure to compute the w-edge of $\pi(p, a_1)$ which is not computed before in $O(n^2/s^2)$ time assuming that we have done this for every node we have visited so far. More specifically, computing the junction of $\pi(p, a_1)$ and $\pi(p, b_i)$ takes $O(n^2/s^2)$ time for $i = 1, 2$, and computing the edge incident to each junction takes $O(n^2/s^3)$ time. One of the edges is the w-edge that we want to compute. Since the size of the wall-tree is $O(s)$, we can do this for every node in $O(n^2/s)$ time in total. Thus we have the following lemma.

▶ **Lemma 16.** *Given a point p in a simple polygon with n vertices, we can compute all w-edges of the shortest paths between p and the endpoints of the walls in $O(n^2/s)$ time using $O(s)$ words of workspace for $s = O(\sqrt{n})$.*

Due to the w-edges, we can compute the shortest path $\pi(p, q)$ in $O(n^2/s^2)$ time for any point q in P. Note that n^2/s^2 is at least n for $s = O(\sqrt{n})$. For a proof, see Section M.

▶ **Lemma 17.** *Given a fixed point p in P and a parameter $s = O(\sqrt{n})$, we can compute $\pi(p, q)$ in $O(n^2/s^2)$ time for any point q in P using $O(s)$ words of workspace after an $O(n^2/s)$-time preprocessing for P and p.*

5.2 Decomposing the Shortest Path Tree into Smaller Trees

We subdivide P into subpolygons each of which is associated with a vertex of it in a way different from the one for the balanced subdivision. Then inside each such subpolygon, we report all edges of the shortest path tree rooted at its associated vertex recursively. We

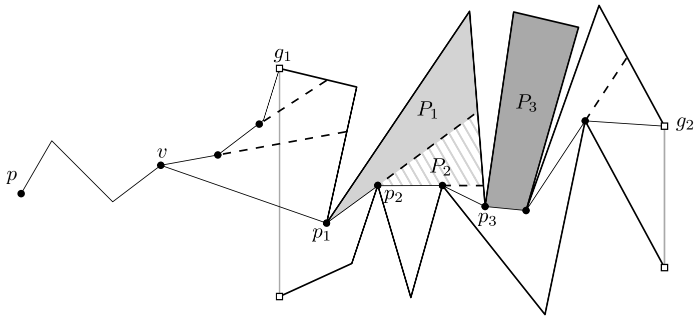


Figure 3 Subdivision of the region bounded by $\pi(v, g_1) \cup \pi(v, g_2)$ and the part of ∂P from g_1 to g_2 in clockwise order along ∂P by extending the edges of $\pi(v, g_1) \cup \pi(v, g_2)$. (P_i, p_i)'s are three of the subproblems of (P, p) for $i = 1, 2, 3$.

guarantee that the edges reported in this way are the edges of the shortest path tree rooted at p. We also guarantee that all edges of the shortest path tree rooted at p are reported. We use a pair (P', p') to denote the problem of reporting the shortest path tree rooted at a point p' inside a simple polygon $P' \subseteq P$. Initially, we are given the problem (P, p).

Structural properties of the decomposition. We use the following two steps of the decomposition. In the first step, we decompose P into a number of subpolygons by the shortest path $\pi(p, a)$ for every endpoint a of the walls. The boundary of each subpolygon consists of polygonal curves from ∂P with endpoints g_1, g_2 and shortest paths $\pi(v, g_1)$ and $\pi(v, g_2)$, where v is the junction of $\pi(p, g_1)$ and $\pi(p, g_2)$. In the second step, we decompose each subpolygon into smaller subpolygons by extending the edges of the shortest paths $\pi(v, g_1)$ and $\pi(v, g_2)$ towards g_1 and g_2, respectively. See Figure 3.

Consider a subpolygon P_i in the resulting subdivision. Its boundary consists of a polygonal curve from ∂P and two line segments sharing a common endpoint p_i. We can represent P_i using $O(1)$ words. Moreover, P_i has complexity of $O(n/s)$. For any point q in P_i, $\pi(p, q)$ is the concatenation of $\pi(p, p_i)$ and $\pi(p_i, q)$. Therefore, the shortest path rooted at p_i of P_i coincides with the shortest path tree rooted at p inside P restricted to P_i. We can obtain the entire shortest path tree rooted at p inside P by solving (P_i, p_i) for every subpolygon P_i in the resulting subdivision and its associated vertex p_i.

The procedure for obtaining this decomposition is described in the full version of this paper. We decompose each problem recursively unless the problem satisfies one of the three stopping conditions in Definition 18. Then we directly solve each base problem (that is, we report the edges of the shortest path tree.) But for non-base problems, we do not report any edge of the shortest path tree. In this way, we report each edge of the shortest path tree at most twice. We can report each edge without repetition using an orientation of each edge.

▶ **Definition 18** (Stopping conditions). There are three stopping conditions for (P_i, p_i): (1) P_i has $O(s)$ vertices, (2) $s \geq \sqrt{|P_i|}$, where $|P_i|$ is the complexity of P_i, and (3) the depth of the recurrence is c, where $c > 0$ is a fixed constant.

When stopping condition (1) holds, we compute the shortest path tree directly using the algorithm by Guibas et al. [8]. When stopping condition (2) holds, we apply the algorithm described in the full version of this paper that computes the shortest path tree rooted at p_i inside P_i in $O(|P_i|^2/s)$ time for the case that $s \geq \sqrt{|P_i|}$, where $|P_i|$ is the complexity of P_i. When stopping condition (3) holds, we compute the shortest path tree using Lemma 13.

For each maximal polygonal curve with endpoints g_1 and g_2 containing no endpoints of walls in its interior, we spend $O(n^2/s^2 + nk/s)$ time, where k is the number of edges of $\pi(v, g_1) \cup \pi(v, g_2)$ for the junction v of $\pi(p, g_1)$ and $\pi(p, g_2)$. Since there are $O(s)$ such maximal polygonal curves and the sum of k over all such maximal polygonal curves is $O(n)$, the running time for decomposing the problem (P, p) into smaller problems is $O(n^2/s)$.

The total time complexity is $O(cn^2/s + (n^2 \log n)/s^c) = O(n^2/s + (n^2 \log n)/s^c)$, and the space complexity is $O(cs) = O(s)$.

▶ **Theorem 19.** *Given a point p in a simple polygon with n vertices, we can compute the shortest path tree rooted at p in $O(n^2/s + (n^2 \log n)/s^c)$ expected time using $O(s)$ words of workspace for an arbitrary constant $c > 0$.*

By setting c to the size of workspace and s to 2, we have the following theorem.

▶ **Theorem 20.** *Given a point p in a simple polygon with n vertices, we can compute the shortest path tree rooted at p in $O((n^2 \log n)/2^s)$ expected time using $O(s)$ words of workspace for $s \leq \log\log n$.*

References

1 Boris Aronov, Matias Korman, Simon Pratt, André van Renssen, and Marcel Roeloffzen. Time-space trade-offs for triangulating a simple polygon. *Journal of Computational Geometry*, 8(1):105–124, 2017.

2 Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Memory-constrained algorithms for simple polygons. *Computational Geometry*, 46(8):959–969, 2013.

3 Tetsuo Asano and David Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *Proceedings of the 13th Algorithms and Data Strucutres Symposium (WADS 2013)*, pages 61–72, 2013.

4 Luis Barba, Matias Korman, Stefan Langerman, Kunihiko Sadakane, and Rodrigo I. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

5 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

6 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, 2008.

7 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System*, 34(1):19–26, 1987.

8 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.

9 Sariel Har-Peled. Shortest path in a polygon using sublinear space. *Journal of Computational Geometry*, 7(2):19–45, 2015.

10 Eunjin Oh and Hee-Kap Ahn. A new balanced subdivision of a simple polygon for time-space trade-off algorithms, 2017. arXiv:1709.09932.

Complexity of Coloring Reconfiguration under Recolorability Constraints*

Hiroki Osawa[1], Akira Suzuki[2], Takehiro Ito[3], and Xiao Zhou[4]

1 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
osawa@ecei.tohoku.ac.jp
2 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
a.suzuki@ecei.tohoku.ac.jp
3 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
takehiro@ecei.tohoku.ac.jp
4 **Graduate School of Information Sciences, Tohoku University, Sendai, Japan**
zhou@ecei.tohoku.ac.jp

Abstract

For an integer $k \geq 1$, k-COLORING RECONFIGURATION is one of the most well-studied reconfiguration problems, defined as follows: In the problem, we are given two (vertex-)colorings of a graph using k colors, and asked to transform one into the other by recoloring only one vertex at a time, while at all times maintaining a proper coloring. The problem is known to be PSPACE-complete if $k \geq 4$, and solvable for any graph in polynomial time if $k \leq 3$. In this paper, we introduce a recolorability constraint on the k colors, which forbids some pairs of colors to be recolored directly. The recolorability constraint is given in terms of an undirected graph R such that each node in R corresponds to a color and each edge in R represents a pair of colors that can be recolored directly. We study the hardness of the problem based on the structure of recolorability constraints R. More specifically, we prove that the problem is PSPACE-complete if R is of maximum degree at least four, or has a connected component containing more than one cycle.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases combinatorial reconfiguration, graph coloring, PSPACE-complete

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.62

1 Introduction

Recently, *reconfiguration problems* [11] have been intensively studied in the field of theoretical computer science. The problem arises when we wish to find a step-by-step transformation between two feasible solutions of a search problem such that all intermediate results are also feasible and each step conforms to a fixed reconfiguration rule, that is, an adjacency relation defined on feasible solutions of the original search problem. (See, e.g., a survey [16] and references in [7, 12].)

One of the most well-studied reconfiguration problems is based on the (vertex-)coloring search problem [1, 2, 3, 4, 6, 8, 9, 13, 17], defined as follows. In the k-COLORING RECONFIGURATION problem, we are given two proper k-colorings f_0 and f_r of the same graph G, and asked to determine whether there is a sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ of proper k-colorings of G such that $f_\ell = f_r$ and f_i can be obtained from f_{i-1} by recoloring only a single vertex in

[*] This work is partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP16K00003, JP16K00004 and JP17K12636, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 62; pp. 62:1–62:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

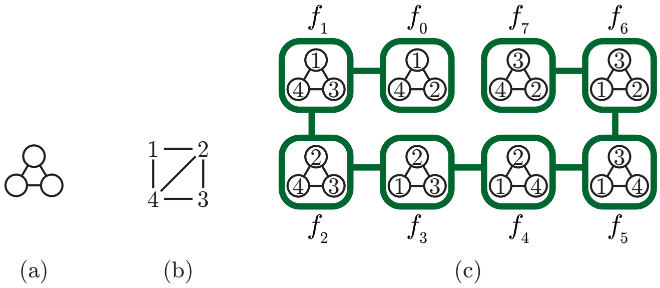


Figure 1 (a) Input graph G, (b) a recolorability graph R with four colors 1, 2, 3 and 4, and (c) an $(f_0 \to f_7)$-reconfiguration sequence.

G for all $i \in \{1, 2, \ldots, \ell\}$. The complexity status of this reconfiguration problem has been clarified based on several "standard" measures (e.g., the number of colors [3, 6] and graph classes [2, 9, 17]) which are used well also for analyzing the original search problem.

In this paper, we propose a new measure to capture the hardness of COLORING RECONFIGURATION according to recoloring steps.

1.1 Our problem

For an integer $k \geq 1$, let C be the *color set* consisting of k colors $1, 2, \ldots, k$. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. Recall that a *k-coloring* f of G is a mapping $f : V(G) \to C$ such that $f(v) \neq f(w)$ holds for each edge $vw \in E(G)$.

In this paper, we introduce the concept of "recolorability" and generalize the adjacency relation on k-colorings. The *recolorability* on the color set C is given in terms of an undirected graph R, called the *recolorability graph* on C, such that $V(R) = C$; each edge $ij \in E(R)$ represents a "recolorable" pair of colors $i, j \in V(R) = C$. Then, two k-colorings f and f' of G are *adjacent* (*under* R) if the following two conditions (a) and (b) hold:

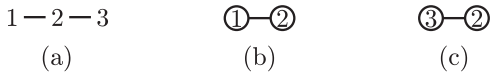
(a) $\big|\{v \in V(G) : f(v) \neq f'(v)\}\big| = 1$, that is, f' can be obtained from f by *recoloring* a single vertex $v \in V(G)$; and

(b) if $f(v) \neq f'(v)$ for a vertex $v \in V(G)$, then $f(v)f'(v) \in E(R)$, that is, the colors $f(v)$ and $f'(v)$ form a recolorable pair.

Figure 1(c) shows eight different 4-colorings of the graph in Figure 1(a). Then, for each $i \in \{1, 2, \ldots, 7\}$, two 4-colorings f_{i-1} and f_i are adjacent under the recolorability graph R in Figure 1(b). As defined above, the known adjacency relation in [1, 2, 3, 4, 6, 8, 9, 13, 17] only requires the condition (a) above, that is, we can recolor a vertex from any color to any color directly. Observe that this corresponds to the case where R is a complete graph of size k, and hence our adjacency relation generalizes the known one.

Given a graph G, and two k-colorings f_0 and f_r of G, the COLORING RECONFIGURATION problem UNDER R-RECOLORABILITY is the decision problem of determining whether there exists a sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ of k-colorings of G such that $f_\ell = f_r$ and f_{i-1} and f_i are adjacent under R for all $i \in \{1, 2, \ldots, \ell\}$; such a desired sequence is called an $(f_0 \to f_r)$-*reconfiguration sequence*. For example, the sequence $\langle f_0, f_1, \ldots, f_7 \rangle$ in Figure 1(c) is an $(f_0 \to f_7)$-reconfiguration sequence.

We emphasize that the concept of recolorability graphs changes the situation drastically from k-COLORING RECONFIGURATION. For example, the $(f_0 \to f_7)$-reconfiguration sequence in Figure 1(c) is a shortest one between f_0 and f_7 under the recolorability graph R in



(a) (b) (c)

Figure 2 (a) Recolorability graph R with three colors 1, 2 and 3, and (b) and (c) 3-colorings f_0 and f_r of a graph consisting of a single edge, respectively.

Figure 1(b). However, in 4-Coloring Reconfiguration (in other words, if R would be K_4 and would have the edge joining colors 1 and 3), we can recolor the vertex from 1 to 3 directly. As another example, the instance illustrated in Figure 2 is a no-instance for our problem even if the number of colors is larger than the number of vertices in an input graph (a single edge), but is clearly a yes-instance for 3-Coloring Reconfiguration.

1.2 Related results

As we have mentioned above, k-Coloring Reconfiguration has been studied intensively from various viewpoints.

From the viewpoint of the number k of colors in the color set C, a sharp analysis has been obtained: Bonsma and Cereceda [3] proved that k-Coloring Reconfiguration is PSPACE-complete if $k \geq 4$. On the other hand, Cereceda et al. [6] proved that k-Coloring Reconfiguration is solvable for any graph in polynomial time if $k \in \{1, 2, 3\}$, despite the fact that the original search problem (i.e., asking for the existence of one 3-coloring in a given graph) is NP-complete. In addition, for any yes-instance of 3-Coloring Reconfiguration, an $(f_0 \to f_r)$-reconfiguration sequence with the shortest length can be found in polynomial time [6, 13].

From the viewpoint of graph classes, Wrochna [17] proved that k-Coloring Reconfiguration remains PSPACE-complete even for graphs with bounded bandwidth (and hence bounded pathwidth). Bonamy et al. [2] gave some sufficient condition with respect to graph structures so that any pair of k-colorings of a graph has a reconfiguration sequence: for example, chordal graphs and chordal bipartite graphs satisfy their sufficient condition.

From the viewpoint of parameterized complexity, the length ℓ of a desired sequence is taken as a parameter for various reconfiguration problems [14]. Bonsma et al. [4] and Johnson et al. [13] independently developed a fixed-parameter algorithm to solve k-Coloring Reconfiguration when parameterized by $k + \ell$. In contrast, if the problem is parameterized only by ℓ, then it is W[1]-hard when k is an input [4] and does not admit a polynomial kernelization when k is fixed unless the polynomial hierarchy collapses [13].

As generalizations of k-Coloring Reconfiguration, reconfiguration problems for H-colorings [18] and circular colorings [5] have been studied. Note that both colorings are generalizations of k-colorings, and always form k-colorings of the same graph; but k-colorings do not always form these colorings. The two reconfiguration problems take the same adjacency relation as the original k-Coloring Reconfiguration (i.e., satisfying only the condition (a) in Section 1.1), but the set of feasible solutions does not always contain all k-colorings. On the other hand, our problem takes the same set of feasible solutions as the original k-Coloring Reconfiguration (i.e., *all* k-colorings), but takes different adjacency relation which obeys a recolorability graph R.

1.3 Our contribution

In this paper, we show the hardness of the COLORING RECONFIGURATION problem UNDER R-RECOLORABILITY based on the graph structure of recolorability graphs. We show the PSPACE-completeness of the problem for two cases. We first prove in Section 3.2 that the problem is PSPACE-complete for any recolorability graph with maximum degree at least four. We then show in Section 3.3 that the problem is PSPACE-complete for any recolorability graph R which contains a connected component R_C such that $|E(R_C)| > |V(R_C)|$, equivalently, for the case where the recolorability graph contains a connected component R_C having at least two cycles. This result implies that there exists a graph R of maximum degree three for which the problem remains PSPACE-complete. We note that our first result is strong in the sense that it shows PSPACE-completeness for *all* recolorability graphs of maximum degree at least four.

Due to the page limitation, we omit some proofs from this extended abstract.

2 Preliminaries

Since we deal with (vertex-)coloring, we may assume without loss of generality that an input graph G is simple, connected and undirected. For a vertex $v \in V(G)$, let $N(G, v) = \{w \in V(G) : vw \in E(G)\}$. We say that a graph H is a *supergraph* of a graph G if both $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$ hold; and hence H can be G itself.

For a graph G and a recolorability graph R on C, we define the *R-reconfiguration graph* on G, denoted by $\mathcal{C}_R(G)$, as follows: $\mathcal{C}_R(G)$ is an undirected graph such that each node of $\mathcal{C}_R(G)$ corresponds to a k-coloring of G, and two nodes in $\mathcal{C}_R(G)$ are joined by an edge if their corresponding k-colorings are adjacent under R. We sometimes call a node in $\mathcal{C}_R(G)$ simply a k-coloring if it is clear from the context. A path in $\mathcal{C}_R(G)$ from a k-coloring f to another one f' is called an *$(f \to f')$-reconfiguration sequence.* Note that any $(f \to f')$-reconfiguration sequence is *reversible*, that is, the path in $\mathcal{C}_R(G)$ forms an $(f' \to f)$-reconfiguration sequence, too. Then, the COLORING RECONFIGURATION problem UNDER R-RECOLORABILITY is the decision problem of determining whether $\mathcal{C}_R(G)$ contains an $(f_0 \to f_r)$-reconfiguration sequence. Note that the problem does not ask for an actual $(f_0 \to f_r)$-reconfiguration sequence as the output.

We introduce the concept of "frozen" vertices from the viewpoint of recoloring, which plays an important role in the paper. For a k-coloring f of a graph G and a recolorability graph R on C, a vertex $v \in V(G)$ is said to be *frozen on f (under R)* if $f'(v) = f(v)$ holds for any k-coloring f' of G such that $\mathcal{C}_R(G)$ has an $(f \to f')$-reconfiguration sequence.

3 PSPACE-completeness

In this section, we clarify the computational hardness of the problem from the viewpoint of recolorability graphs R. In Section 3.1, we introduce the list variant of the problem. Interestingly, the list variant is equivalent to the non-list one in our reconfiguration problem, and hence it suffices to construct reductions to the list variant. In Sections 3.2 and 3.3, we give our hardness results.

3.1 List recolorability

In the *list* variant, each vertex v of a graph G is associated with a subgraph $L_R(v)$ of the common recolorability graph R; we call $L_R(v)$ the *list recolorability of* v, and sometimes call the list assignment (mapping) L_R the *list R-recolorability.* Note that $L_R(v)$ is not necessarily

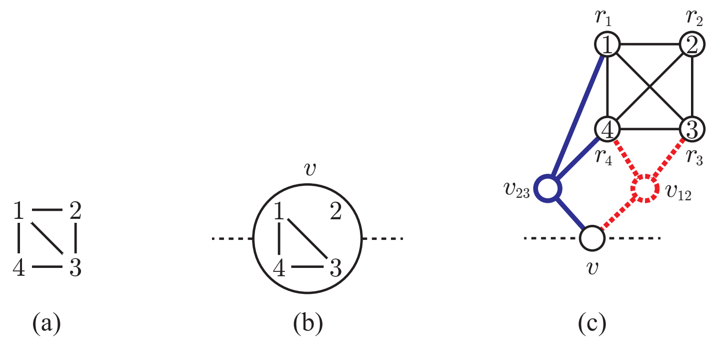


Figure 3 (a) Recolorability graph R such that $L_R(v)$ is a subgraph of R for each vertex $v \in V(G)$, (b) a vertex $v \in V(G)$ whose list recolorability $L_R(v)$ is written inside, and (c) the vertex v in G', where the (red) thick dotted part corresponds to forbidding the pair of colors 1 and 2 in $E(R) \setminus E(L_R(v))$ and the (blue) thick part corresponds to forbidding the pair of colors 2 and 3 in $E(R) \setminus E(L_R(v))$.

a spanning subgraph of R. Let $k = |V(R)|$. Then, a k-coloring f of G is called a *list coloring* of G if $f(v) \in V(L_R(v))$ for all vertices v in G. Observe that for any supergraph R' of R, any list R-recolorability is also list R'-recolorability. We say that two list colorings f and f' are *adjacent under* L_R if they differ in exactly one vertex v such that $f(v)f'(v) \in E(L_R(v))$. Analogous to the R-reconfiguration graph, we define the *L_R-reconfiguration graph* on G, denoted by $\mathcal{C}_{L_R}(G)$, as the undirected graph whose nodes correspond to list colorings of G, and two nodes in $\mathcal{C}_{L_R}(G)$ are joined by an edge if their corresponding list colorings are adjacent under L_R.

Let G be an input graph with a list R-recolorability L_R. Then, for two list colorings f_0 and f_r of G, the COLORING RECONFIGURATION problem UNDER LIST R-RECOLORABILITY (the *list variant*, for short) is the decision problem of determining whether $\mathcal{C}_{L_R}(G)$ contains an $(f_0 \to f_r)$-reconfiguration sequence. Observe that COLORING RECONFIGURATION UNDER R-RECOLORABILITY can be seen as the list variant such that $L_R(v) = R$ holds for every vertex $v \in V(G)$. Furthermore, note that $\mathcal{C}_{L_R}(G)$ forms a subgraph of $\mathcal{C}_R(G)$.

Interestingly, the list variant for our reconfiguration problem is equivalent to the non-list one, as in the following theorem.

▶ **Theorem 1.** COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY *can be reduced to* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *in time polynomial in* $|V(G)|$ *and* $|V(R)|$, *where G is an input graph of the list variant.*

Proof. Let G be an input graph for the list variant with a list R-recolorability L_R, and suppose that we are given two list colorings f_0 and f_r of G. Then, we construct a corresponding instance of COLORING RECONFIGURATION UNDER R-RECOLORABILITY; we denote by G' the corresponding graph, and by f'_0 and f'_r the corresponding initial and target k-colorings of G', respectively, where $k = |V(R)|$.

Indeed, we will give a gadget which forbids recoloring a vertex $v \in V(G)$ directly from a color i to another color j for each pair $ij \in E(R) \setminus E(L_R(v))$. Note that, for each color i in $V(R) \setminus V(L_R(v))$, we can add the vertex i to $L_R(v)$ as an isolated vertex (by adding the forbidding gadgets between i and all colors j such that $ij \in E(R)$). Then, since f_0 and f_r are list colorings of G, both $f_0(v) \neq i$ and $f_r(v) \neq i$ hold and hence v is never recolored to the isolated color i.

To construct such a forbidding gadget, we will use a (newly added) clique of size $k = |V(R)|$ such that all vertices are colored with distinct colors. Notice that no vertex in the clique can be recolored to any color, that is, they are frozen vertices on the k-coloring. We use this property, and construct the corresponding instance, as follows.

We first add to G a new clique K_k of k vertices $r_1, r_2, \dots, r_k$. Then, for each vertex $v \in V(G)$, consider any pair of colors i and j such that $ij \in E(R) \setminus E(L_R(v))$. We add a new vertex v_{ij} to G, and join it with v. In addition, we join v_{ij} with all vertices in $V(K_k) \setminus \{r_i, r_j\}$. (See Figure 3(a)–(c) as an example of the application of this procedure.) Let G' be the resulting graph after applying the procedure above to all vertices $v \in V(G)$ and all pairs $ij \in E(R) \setminus E(L_R(v))$. For notational convenience, we denote by V_F the set of all vertices v_{ij} in G' that are newly added for each vertex $v \in V(G)$ and $ij \in E(R) \setminus E(L_R(v))$. We note that $V(G')$ is partitioned into $V(G), V(K_k)$, and V_F. Furthermore, each vertex $v_{ij} \in V_F$ satisfies $N(G', v_{ij}) \cap V(G) = \{v\}$. We denote by v this unique vertex in $N(G', v_{ij}) \cap V(G)$ for each vertex $v_{ij} \in V_F$. Then, the corresponding k-colorings f'_0 and f'_r of G' are defined as follows: for each $l \in \{0, r\}$ and a vertex $w \in V(G')$,

$$f'_l(w) = \begin{cases} f_l(w) & \text{if } w \in V(G); \\ i & \text{if } w = r_i \in V(K_k); \\ j & \text{if } w = v_{ij} \in V_F \text{ and } f_l(v) = i; \text{ and} \\ i & \text{otherwise, that is, } w = v_{ij} \in V_F \text{ and } f_l(v) \neq i. \end{cases}$$

Then, all vertices $r_1, r_2, \dots, r_k$ are frozen on both f'_0 and f'_r (indeed, under any recolorability graph). This completes the construction of the corresponding instance. This construction can be done in time polynomial in $|V(G)|$ and $k = |V(R)|$.

The correctness proof of our reduction is omitted from this extended abstract. ◀

Recall that for any supergraph R' of R, any list R-recolorability is also a list R'-recolorability, therefore we obtain the following corollary:

▶ **Corollary 2.** *Let R' be an arbitrary supergraph of a recolorability graph R. Then,* Coloring reconfiguration under list R-recolorability *can be reduced to* coloring reconfiguration under R'-recolorability *in time polynomial in $|V(G)|$ and $|V(R')|$, where G is an input graph of the list variant.*

3.2 Recolorability graphs of maximum degree at least four

In this subsection, we consider the case where a recolorability graph is of maximum degree at least four. We emphasize again that the following theorem holds for an arbitrary recolorability graph as long as its maximum degree is at least four.

▶ **Theorem 3.** *Let R' be any recolorability graph whose maximum degree is at least four. Then,* coloring reconfiguration under R'-recolorability *is PSPACE-complete.*

Proof. Observe that the problem can be solved in (most conveniently, nondeterministic [15]) polynomial space, and hence it is in PSPACE. Therefore, we show that the problem is PSPACE-hard for such a recolorability graph R'. Notice that, since R' is of maximum degree at least four, R' is a supergraph of a star $K_{1,4}$. Therefore, by Corollary 2 it suffices to prove that the list variant remains PSPACE-hard even for a list R-recolorability such that $R = K_{1,4}$. (See Figure 4(a).) To show this, we give a polynomial-time reduction from 4-coloring reconfiguration, which is known to be PSPACE-complete [3].

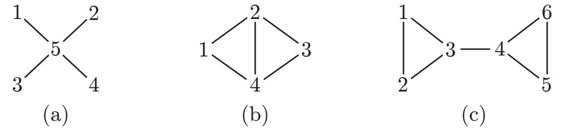


Figure 4 (a) Recolorability graph $K_{1,4}$. In our reduction, the star $K_{1,4}$ with center color 5 is the list recolorability of all vertices $v \in V(G)$. (b) Recolorability graph which is a diamond graph. (c) Recolorability graph which is a $2K_3 + e$ graph.

Let G be an input graph for 4-COLORING RECONFIGURATION, and let f_0 and f_r be two given 4-colorings of G; let $C = \{1, 2, 3, 4\}$ be the color set. As a corresponding instance of the list variant, we take the same graph G in which the list recolorability $L_R(v)$ of each vertex $v \in V(G)$ is a star $K_{1,4}$ such that its center is a new color 5 and its leaves are the four colors $1, 2, 3$ and 4. Then, both f_0 and f_r are list colorings of the corresponding graph G, and we take the 4-colorings f_0 and f_r as the corresponding list colorings. This completes the construction of the corresponding instance, and hence it can be done in polynomial time.

The correctness proof of our reduction is omitted from this extended abstract. ◀

3.3 Recolorability graphs with more than one cycle

In this subsection, we consider the case where a recolorability graph R' contains a connected component having more than one cycle. Our result is expressed as follows:

▶ **Theorem 4.** *Let R' be a recolorability graph which contains a connected component R such that $|E(R)| > |V(R)|$. Then,* COLORING RECONFIGURATION UNDER R'-RECOLORABILITY *is PSPACE-complete.*

To prove Theorem 4, by Corollary 2 it suffices to prove that the list variant remains PSPACE-hard for a list R-recolorability, where R is a connected component in R' such that $|E(R)| > |V(R)|$. We first characterize the structure of R by two small graphs: A graph is called a *diamond* graph if it can be obtained by deleting exactly one edge from a complete graph K_4 of size four (see Figure 4(b)); a $2K_3 + e$ graph is a graph obtained by adding exactly one edge to disjoint union of two triangles K_3 (see Figure 4(c).) We have the following lemma.

▶ **Lemma 5.** *Let R be a connected graph such that $|E(R)| > |V(R)|$. Then, R satisfies at least one of the following statements*:

(a) *R has a vertex whose degree is at least four*;
(b) *R is a supergraph of some subdivision of a diamond graph*; *and*
(c) *R is a supergraph of some subdivision of a $2K_3 + e$ graph.*

If Lemma 5(a) holds for the recolorability graph R, then COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete by Theorem 3. Therefore, it suffices to prove the PSPACE-hardness for the other cases, that is, the recolorability graph R is either (b) any subdivision of a diamond graph, or (c) any subdivision of a $2K_3 + e$ graph.

We now give a sketch of our proof. We first prove that the list variant remains PSPACE-hard for a list R-recolorability when R is either a diamond graph or a $2K_3 + e$ graph (without subdivisions). We use these claims as the bases of inductive proofs for the cases (b) and (c). Due to the page limitation, we only prove the base for the case (b), as follows.

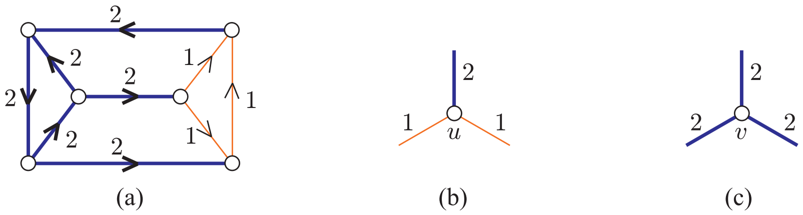


Figure 5 (a) A configuration of an NCL machine, (b) NCL AND vertex u, and (c) NCL OR vertex v.

▶ **Lemma 6.** *Let* D *be a diamond graph. Then,* COLORING RECONFIGURATION UNDER LIST D-RECOLORABILITY *is PSPACE-complete.*

Proof. We give a polynomial-time reduction from NONDETERMINISTIC CONSTRAINT LOGIC (NCL, for short) [10], defined as follows. An NCL "machine" is specified by an undirected graph together with an assignment of weights from $\{1, 2\}$ to each edge of the graph. An (*NCL*) *configuration* of this machine is an orientation (direction) of the edges such that the sum of weights of in-coming arcs at each vertex is at least two. Figure 5(a) illustrates a configuration of an NCL machine, where each weight-2 edge is depicted by a thick (blue) line and each weight-1 edge by a thin (orange) line. Then, two NCL configurations are *adjacent* if they differ in a single edge direction. Given an NCL machine and its two configurations, it is known to be PSPACE-complete to determine whether there exists a sequence of adjacent NCL configurations which transforms one into the other [10].

In fact, the problem remains PSPACE-complete even for AND/OR *constraint graphs*, which consist only of two types of vertices, called "NCL AND vertices" and "NCL OR vertices." A vertex of degree three is called an *NCL* AND *vertex* if its three incident edges have weights 1, 1 and 2. (See Figure 5(b).) An NCL AND vertex u behaves as a logical AND, in the following sense: the weight-2 edge can be directed outward for u if and only if both two weight-1 edges are directed inward for u. Note that, however, the weight-2 edge is not necessarily directed outward even when both weight-1 edges are directed inward. A vertex of degree three is called an *NCL* OR *vertex* if its three incident edges have weights 2, 2 and 2. (See Figure 5(c).) An NCL OR vertex v behaves as a logical OR: one of the three edges can be directed outward for v if and only if at least one of the other two edges is directed inward for v. It should be noted that, although it is natural to think of NCL AND/OR vertices as having inputs and outputs, there is nothing enforcing this interpretation; especially for NCL OR vertices, the choice of input and output is entirely arbitrary because an NCL OR vertex is symmetric. For example, the NCL machine in Figure 5(a) is an AND/OR constraint graph. From now on, we call an AND/OR constraint graph simply an *NCL machine*, and call an edge in an NCL machine an *NCL edge*.

Gadgets. We first subdivide every NCL edge vw into a path $vv'w'w$ of length three by adding two new vertices v' and w'; the newly added vertices v' and w' are called *connectors* for v and w, respectively. (See Figure 6(a) and (b).) We call the edge $v'w'$ a *link edge* between two NCL vertices v and w, and call the edges vv' and ww' *NCL one-third edges* for v and w, respectively. Notice that every vertex in the resulting graph belongs to exactly one of stars $K_{1,3}$ such that the center v of each $K_{1,3}$ corresponds to an NCL AND/OR vertex and the three leaves are connectors for v. Furthermore, these stars are all mutually disjoint, and joined together by link edges. (See Figure 6(c) as an example.)

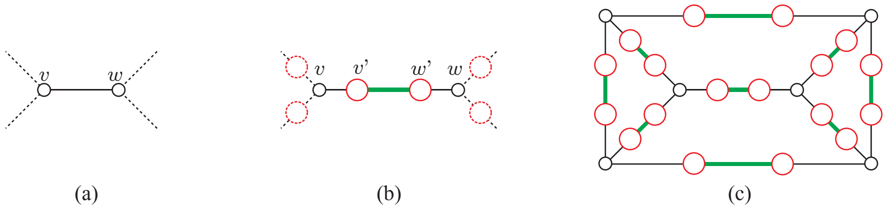


Figure 6 (a) An NCL edge vw, (b) its subdivision into a path $vv'w'w$, and (c) the resulting graph which corresponds to the NCL machine in Figure 5(a), where each connector is depicted by a (red) large circle and each link edge by a thin (green) line.

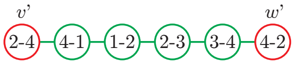


Figure 7 The link edge gadget $G_{v'w'}$ between two connectors v' and w'.

Therefore, our reduction involves constructing three types of gadgets which correspond to link edges and stars of NCL AND/OR vertices. In our gadgets, all connectors v' for NCL AND/OR vertices v have the same list recolorability $L_{\mathsf{D}}(v')$ such that $V(L_{\mathsf{D}}(v')) = \{2, 4\}$ and $E(L_{\mathsf{D}}(v')) = \{24\}$. Then, in our reduction, assigning the color 4 to v' always corresponds to directing the NCL one-third edge vv' from v' to v (i.e., the inward direction for v), while assigning the color 2 to v' always corresponds to directing vv' from v to v' (i.e., the outward direction for v).

(i) Link edge gadget. Figure 7 illustrates our link edge gadget $G_{v'w'}$ for each link edge $v'w'$, where v' and w' are connectors for NCL AND/OR vertices v and w, respectively. The graph in each vertex (circle) indicates the list recolorability of the vertex. Recall that, in a given NCL machine, v and w are joined by a single NCL edge. Therefore, the link edge gadget should be consistent with the orientations of the NCL edge, as follows (see also Figure 8(a) and (b)): If we assign 4 to v' (the inward direction for v), then w' must be colored with 2 (the outward direction for w); conversely, v' must be colored with 2 if we assign 4 to w'. In particular, the gadget must forbid a list coloring which assigns 4 to both v' and w' (the inward directions for both v and w), because such a list coloring corresponds to the direction which contributes to both v and w illegally. On the other hand, assigning 2 to both v' and w' (the outward directions for both v and w) corresponds to the *neutral* orientation of the NCL edge vw which contributes to neither v nor w, and hence we simply do not care such an orientation.

Figure 8(c) illustrates the L_{D}-reconfiguration graph $\mathcal{C}_{L_{\mathsf{D}}}(G_{v'w'})$ on the link edge gadget $G_{v'w'}$. Each rectangle corresponds to a node of $\mathcal{C}_{L_{\mathsf{D}}}(G_{v'w'})$, that is, a list coloring of $G_{v'w'}$, where the underlined bold number represents the color assigned to the vertex. Then, $\mathcal{C}_{L_{\mathsf{D}}}(G_{v'w'})$ is connected, and there is no list coloring which assigns 4 to both v' and w', as claimed above. Furthermore, the reversal of the NCL edge vw can be simulated by the path on $\mathcal{C}_{L_{\mathsf{D}}}(G_{v'w'})$ via the neutral orientation of vw, as illustrated in Figure 8(c). Thus, this gadget works correctly as a link edge.

(ii) And gadget. Figure 9 illustrates our AND gadget G_{and} for each NCL AND vertex v, where v_a, v_b and v_c correspond to the three connectors for v. In the figure, the connectors v_a and v_b come from the two weight-1 NCL edges, while the connector v_c comes from the weight-2 NCL edge. We now explain this gadget works as an NCL AND vertex. Similarly as

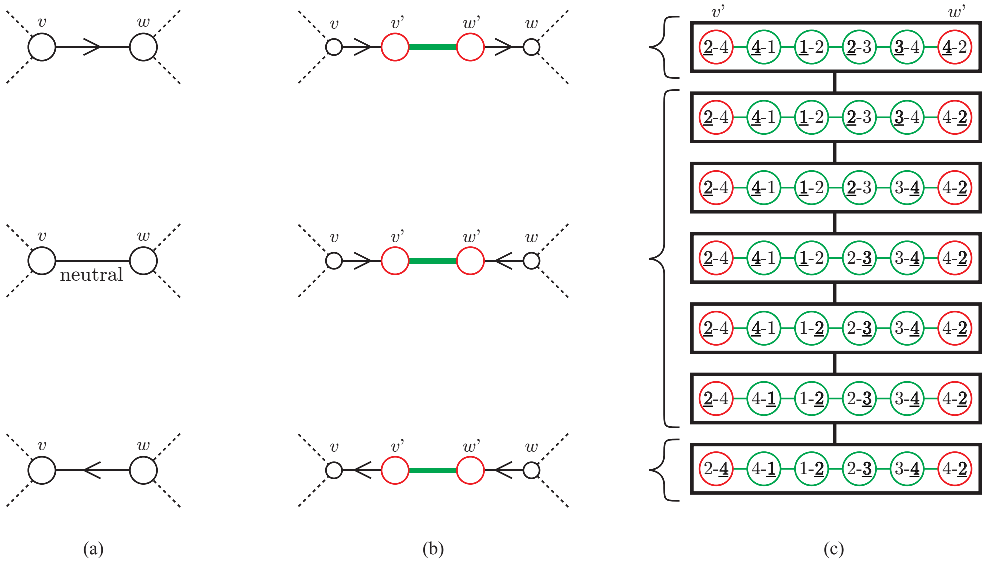


Figure 8 (a) Three orientations of an NCL edge vw, (b) their corresponding orientations of the NCL one-third edges vv' and ww', and (c) all list colorings of the link edge gadget $G_{v'w'}$ in the L_{D}-reconfiguration graph $\mathcal{C}_{L_{\mathsf{D}}}(G_{v'w'})$.

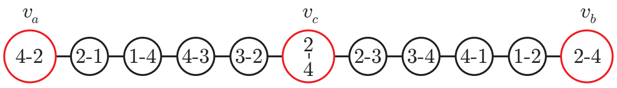


Figure 9 AND gadget G_{and} with three connectors v_a, v_b and v_c.

for the link edge gadget, the AND gadget must forbid the case where all the connectors v_a, v_b and v_c are colored with 2 at the same time (i.e., all NCL one-third edges vv_a, vv_b and vv_c take the outward direction for v). In addition, the gadget must simulate the following situation: v_c can be colored with 2 (i.e., the weight-2 edge vv_c can take the outward direction for v) only when both v_a and v_b are colored with 4 at the same time (i.e., both the weight-1 edges vv_a and vv_b take the inward direction for v).

Figure 10(a) illustrates all feasible orientations of the three NCL one-third edges vv_a, vv_b and vv_c, whose corresponding assignments of colors to the connectors are depicted in Figure 10(b). Due to the space limitation, in Figure 10(b), we only indicate the colors assigned to v_a, v_c and v_b, but Figure 10(c) shows all list (proper) colorings of G_{and} that assign the colors 2, 4 and 4 to v_a, v_c and v_b, respectively. Then, as illustrated in Figure 10(c), these list colorings are "internally connected," that is, any two list colorings are reconfigurable with each other without recoloring any connector of G_{and}. Furthermore, this gadget preserves the "external adjacency" in the following sense: if we contract the list colorings in $\mathcal{C}_{L_{\mathsf{D}}}(G_{\mathsf{and}})$ having the same color assignments to the connectors into a single vertex, then the resulting graph is exactly the graph depicted in Figure 10(a). We have checked by a computer search that these two properties hold for our AND gadget. Therefore, we can conclude that our AND gadget correctly works as an NCL AND vertex.

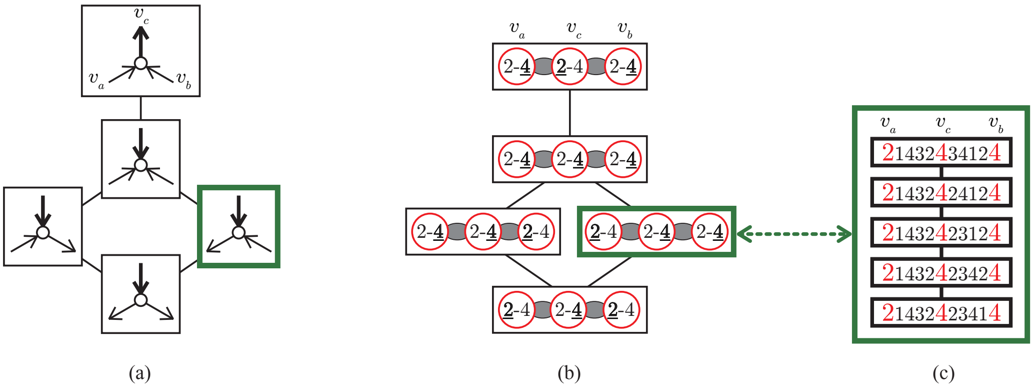


Figure 10 (a) All feasible orientations of the three NCL one-third edges incident to an NCL AND vertex together with their adjacency, (b) image of L_D-reconfiguration graph $\mathcal{C}_{L_\mathrm{D}}(G_\mathsf{and})$ on the AND gadget G_and, and (c) the inside of the rightmost (green) thick box in the image which corresponds to assigning the colors 2, 4 and 4 to v_a, v_c and v_b, respectively, where we simply write the colors assigned to G_and by a sequence of colors.

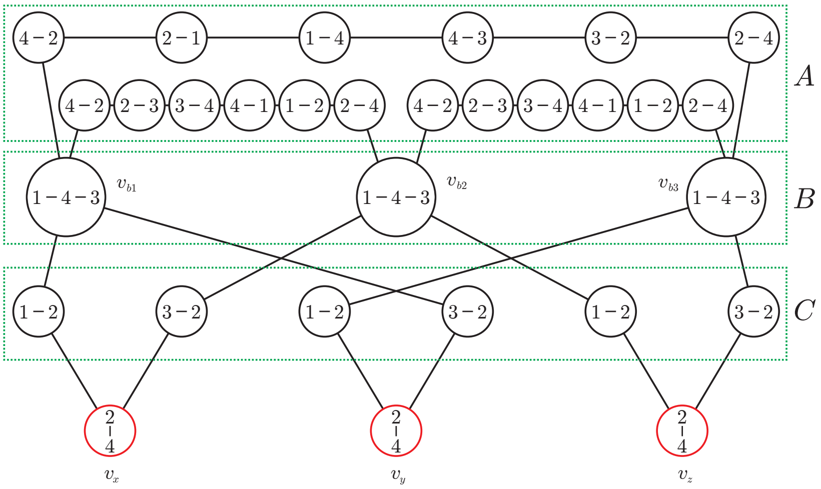


Figure 11 OR gadget G_or with three connectors v_x, v_y and v_z.

(iii) Or gadget. Figure 11 illustrates our OR gadget G_or for each NCL OR vertex v, where v_x, v_y and v_z correspond to the three connectors for v. We now explain this gadget works as an NCL OR vertex. For each NCL OR vertex v, it suffices that at least one of the three NCL edges take the inward direction for v. Thus, the OR gadget must forbid only the case where all the connectors v_x, v_y and v_z are colored with 2 at the same time. Indeed, our gadget in Figure 11 forbids such the case, because otherwise all three vertices v_{b1}, v_{b2} and v_{b3} in Part B must be colored with 4 and this yields that there is no available color for vertices in Part A.

Similarly as for the AND gadget, we have checked by a computer search that our OR gadget is internally connected and preserves the external adjacency. Therefore, we can conclude that our OR gadget correctly works as an NCL OR vertex.

Reduction. As we have mentioned above, we first subdivide every NCL edge vw into a path $vv'w'w$ of length three by adding two connectors v' and w'. (See Figure 6.) Then, we replace each of link edges and NCL AND/OR vertices with its corresponding gadget; let G be the resulting graph. In addition, we construct two list colorings of G which correspond to two given configurations C_0 and C_r of the NCL machine. Note that there are (in general, exponentially) many list colorings which correspond to the same NCL configuration. However, by the construction of the three gadgets, no two distinct NCL configurations correspond to the same list coloring of G. We thus choose any two list colorings f_0 and f_r of G which correspond to C_0 and C_r, respectively. This completes the construction of the corresponding instance for the list variant under list D-recolorability. This construction can be done in polynomial time.

We omit the correctness proof of our reduction from this extended abstract. ◀

References

1 Marthe Bonamy and Nicolas Bousquet. Recoloring bounded treewidth graphs. In *the 7th Latin-American Algorithms, Graphs and Optimization Symposium* (*LAGOS 2013*), volume 44 of *Electronic Notes in Discrete Mathematics*, pages 257–262, 2013.

2 Marthe Bonamy, Matthew Johnson, Ioannis Lignos, Viresh Patel, and Daniël Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27:132–143, 2014.

3 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410:5215–5226, 2009.

4 Paul S. Bonsma, Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *the 9th International Symposium on Parameterized and Exact Computation* (*IPEC 2014*), volume 8894 of *Lecture Notes in Computer Science*, pages 110–121, 2014.

5 Richard C. Brewster, Sean McGuinness, Benjamin Moore, and Jonathan A. Noel. A dichotomy theorem for circular colouring reconfiguration. *Theoretical Computer Science*, 639:1–13, 2016.

6 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67:69–82, 2011.

7 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015.

8 Carl Feghali, Matthew Johnson, and Daniël Paulusma. A reconfigurations analogue of brooks' theorem and its consequences. *Journal of Graph Theory*, 83:340–358, 2016.

9 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98-A:1168–1178, 2015.

10 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343:72–96, 2005.

11 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412:1054–1065, 2011.

12 Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. In *the 12th Annual Conference on Theory and Applications of Models of Computation* (*TAMC 2015*), volume 9076 of *Lecture Notes in Computer Science*, pages 212–223, 2015.

13 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75:295–321, 2016.

14 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78:274–297, 2017.

15 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

16 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160, 2013.

17 Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *CoRR*, abs/1405.0847, 2014.

18 Marcin Wrochna. Homomorphism reconfiguration via homotopy. In *the 32nd International Symposium on Theoretical Aspects of Computer Science* (*STACS 2015*), volume 30 of *Leibniz International Proceedings in Informatics*, pages 730–742, 2015.

Approximate Nearest Neighbors Search Without False Negatives For l_2 For $c > \sqrt{\log \log n}$[*]

Piotr Sankowski[1] **and Piotr Wygocki**[2]

1 **University of Warsaw, Poland**
sank@mimuw.edu.pl
2 **University of Warsaw, Poland**
wygos@mimuw.edu.pl

Abstract

In this paper, we report progress on answering the open problem presented by Pagh [11], who considered the nearest neighbor search without false negatives for the Hamming distance. We show new data structures for solving the c-approximate nearest neighbors problem without false negatives for Euclidean high dimensional space $\mathcal{R}^d$. These data structures work for any $c = \omega(\sqrt{\log \log n})$, where n is the number of points in the input set, with poly-logarithmic query time and polynomial pre-processing time. This improves over the known algorithms, which require c to be $\Omega(\sqrt{d})$.

This improvement is obtained by applying a sequence of reductions, which are interesting on their own. First, we reduce the problem to d instances of dimension logarithmic in n. Next, these instances are reduced to a number of c-approximate nearest neighbor search without false negatives instances in $\left(\mathbb{R}^k\right)^L$ space equipped with metric $m(x, y) = \max_{1 \leq i \leq L}(\|x_i - y_i\|_2)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.3 Probability and Statistics

Keywords and phrases locality sensitive hashing, approximate nearest neighbor search, high-dimensional, similarity search

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.63

1 Introduction

The nearest neighbor search has numerous applications in image processing, search engines, recommendation engines, predictions and machine learning. We define the nearest neighbor problem as follows: for a given input set, a query point and a distance R, return a point (optionally all points) from the input set, which is closer to the query point than R in the given metric (typically l_p for $p \in [1, \infty]$), or report that such a point does not exist. The input set and the distance R are known in advance. Hence, the input set may be preprocessed what may result in reducing the query time. The problem in which the distance R is not known during the preprocessing and our task is to find the nearest neighbor can be efficiently reduced to the problem defined as above [7].[1] Unfortunately, the nearest neighbors search, defined as above, appears to be intractable for high dimensional spaces such as l_p^d for large d. The existence of an algorithm with a sub-linear in the data size and not exponential in

[*] This work was partially supported by grant NCN2014/13/B/ST6/00770 of Polish National Science Center and ERC StG grant TOTAL no. 677651.

[1] Authors used to distinguish between these two problems. The problem in which the radius is known in pre-processing is sometimes called Point Location in Equal Balls (PLEB) [7].

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 63; pp. 63:1–63:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

d query time and with not exponential in d pre-processing, would contradict the strong exponential time hypothesis [12]. In order to overcome this obstacle, the c-approximate nearest neighbors problem with $c > 1$, was introduced. In this problem, the query result is allowed to contain points which are within the distance cR from the query point. In other words, the points within the distance R from the query point are classified as neighbors, the points farther than cR are classified as non-neighbors, while the rest may be classified into any of these two categories. This assumption makes the problem easier, for many metric spaces such as l_p when $p \in [1, 2]$ or the Hamming space [7]. On one hand, sub linear in the input size queries are possible. On the other hand, the queries and pre-processing times are polynomial in the dimension of the space.

Locality sensitive hashing (LSH) is one of the major techniques for solving the c-approximate nearest neighbor search. Many LSH functions are mappings which roughly preserve distances. A random LSH function maps two 'close' points to two 'close' hashes with 'large' probability. Analogously, two 'distant' points are mapped to two 'distant' hashes with 'large' probability. Roughly speaking, the LSH is used to reduce the dimension of the input space, which allows to solve the problem in the lower dimensional space. Thus, the efficiency of the algorithm strongly depends on the quality of LSH functions used. The crucial properties of the LSH functions are the probability of false positives and the probability of false negatives. A false negative is a point which is 'close' to the query point, but its hash is 'far away' from the hash of the query point. Analogously, the false positive is a point whose distance to the query point is 'large', but it is mapped to a 'close' hash.

The previously known algorithms for the c-approximate nearest neighbors (see e.g., [2, 4]) give Monte Carlo guaranties for returned points, i.e., an input point close to the query point is returned with some probability. In other words, there might be false negatives. For example, a common choice of the hash functions is $f(x) = \langle x, v\rangle$ or $f(x) = \lfloor\langle x, v\rangle\rfloor$, where v is a vector of numbers drawn independently from some probability distribution [2, 7, 10]. For a Gaussian distribution, $\langle x, v\rangle$ is also Gaussian with zero mean and standard deviation equal to $\|x\|_2$. It is easy to see that these are LSH functions for l_2, but as mentioned above, they only give probabilistic guarantees. In this paper, we aim to enhance this by focusing on the c-approximate nearest neighbor search without false negatives for l_2. In other words, we consider algorithms, where a point 'close' to the query point is guaranteed to be returned.

Throughout this paper, we assume that $n \gg d$ and $\exp(d) \gg n$. This represents a situation where the exhaustive scan through all the input points, as well as the usage of data structures exponentially dependant on d, become intractable. The typical values to consider could be $n = 10^9$ and $d = 100$. If not explicitly specified, all statements assume the usage of the l_2 norm.

2 Related Work

There exists an efficient, Monte Carlo c-nearest neighbor algorithm for l_1 [7] with the query and the pre-processing complexity equal to $\mathcal{O}(n^{1/c})$ and $\mathcal{O}(n^{1+1/c})$, respectively. For l_2 in turn, there exists a near to optimal [9] algorithm [2] with the query and the pre-processing complexity equal to $n^{1/c^2+o(1)}$ and $n^{1+1/c^2+o(1)}$, respectively. Moreover, the algorithms presented in [7] work for l_p for any $p \in [1, 2]$. There are also data dependent algorithms, which take into account the actual distribution of the input set [3], which achieve query time $dn^{\rho+o(1)}$ and space $\mathcal{O}(n^{1+\rho+o(1)} + dn)$, where $\rho = 1/(2c^2 - 1)$.

Pagh [11] considered the c-approximate nearest neighbor search without false negatives (NN_{wfn}) for the Hamming space, obtaining the results close to those of [7]. He showed that the bounds of his algorithm for $cR = \log(n/k)$ differ by at most a factor of $\ln 4$ in the exponent in comparison to the bounds of [7]. Recently, Ahle [1] showed an optimal [9] algorithm for the nearest neighbor without false negatives for Hamming space and Braun-Blanquet metric. Indyk [5] provided a deterministic algorithm for l_∞ for $c = \Theta(\log_{1+\rho} \log d)$ with the storage $\mathcal{O}(n^{1+\rho} \log^{O(1)} n)$ and the query time $\log^{O(1)} n$ for some tunable parameter ρ. He proved that the nearest neighbor without false negatives for l_∞ for $c < 3$ is as hard as the subset query problem, a long-standing combinatorial problem. This indicates that the nearest neighbor without false negatives for l_∞ might be hard to solve for any $c > 1$. Also, Indyk [6] considered deterministic mappings $l_2^n \to l_1^m$, for $m = n^{1+o(1)}$, which might be useful for constructing efficient algorithms for the nearest neighbor without false negatives [11].

Pacuk et al. [10] presented a schema for solving the nearest neighbor without false negatives for any $p \in [1, \infty]$ for $c = \Omega(d^{\max\{1/2, 1-1/p\}})$. Using the enhanced hash functions, Wygocki [13] presented algorithms with improved complexities. He considered two hashing families, giving different trade-offs between the execution times and the conditions on c. In particular, (Theorem 3, case 2, for $p = 2$ in [13]):

▶ **Theorem 1** ([13]). *For any $c > \tilde{\tau} = 2\sqrt{d}$, there are data structures for the nearest neighbor without false negatives with*

- *$\mathcal{O}(n^{1+\frac{\ln 3}{\ln(c/\tilde{\tau})}})$ pre-processing time and $\mathcal{O}(d|P| + d \log n + d^2)$ query time for the 'fast query' algorithm,*
- *$\mathcal{O}(nd \log n)$ pre-processing time and $\mathcal{O}(d(|P| + n^{\frac{\ln 3}{\ln(3c/\tilde{\tau})}}))$ query time for the 'fast pre-processing' algorithm,*

where $|P|$ is the size of the result.

The dimension reduction with means of random linear mappings was considered previously in a more general context. In particular, Johnson-Lindenstrauss Lemma [8] is the most well known reference here. The concentration bounds used to prove this classic result will be very useful in our reductions:

▶ **Lemma 2** (Johnson-Lindenstrauss). *Let $Y \in \mathbb{R}^d$ be chosen uniformly from the surface of the d-dimensional sphere. Let $Z = (Y_1, Y_2, \ldots, Y_k)$ be the projection onto the first k coordinates, where $k < d$. Then for any $\alpha < 1$:*

$$\mathbb{P}\left[\frac{d}{k}\|Z\|_2^2 \le \alpha\right] \le \exp(\frac{k}{2}(1 - \alpha + \log \alpha)), \tag{1}$$

3 Our contribution

Recently, efficient algorithms were proposed for solving the approximate nearest neighbor search without false negatives for $c = \Omega(\max\{\sqrt{d}, d^{1-1/p}\})$ in l_p for any $p \in [1, \infty]$ [10, 13]. The main problem with these algorithms is the constraint on c. For l_2, the previous result require c to be of order of $\Omega(\sqrt{d})$, thus the nearest neighbor algorithm were allowed to return points within $\mathcal{O}(\sqrt{d}R)$ radius from query point. We relax this to any c, which makes the presented algorithms usable in practical cases. The contribution of this paper is relaxing this condition and improving the complexity of the algorithms for l_2:

- We show that the $\mathsf{NN}_{\mathrm{wfn}}$ can be reduced to d instances of $\mathsf{NN}_{\mathrm{wfn}}$ in $\mathbb{R}^{\log n}$. For our typical settings of parameters, the factor of d is negligible. As a result, reducing the dimension leads to reducing the complexity of the problem. Moreover, it leads to relaxing the conditions on c to $c = \Omega(\sqrt{\log n})$.
- Further reductions lead to algorithms for any $c = \omega(\sqrt{\log \log n})$. We introduce an algorithm with the $n^{o(1)}$ query time and polynomial pre-processing time, which for large c tents to $n^{1+o(1)}$.

The first reduction is interesting on its own since further work on the problem can be done under the assumption that the dimension of the problem is logarithmic in n. This simplifies the problem at a cost of multiplying the complexities by a factor of d. Also, the authors of [10] proved that their construction is tight for $d = \omega(\log n)$, living the case of $d = \Theta(\log n)$ inconclusive.

3.1 Used Methods

In order to relax the conditions on c, we apply a sequence of dimension reductions. In Section 4, we show how to reduce the c-approximate nearest neighbors in l_2^d ($\mathsf{NN}_{\mathrm{wfn}}(c, d)$) to $d/\log(n)$ instances of $\mathsf{NN}_{\mathrm{wfn}}(\mathcal{O}(c), \mathcal{O}(\log n))$. Applying the algorithm of [13] as a black box gives the first improvement over [13]: an efficient algorithm for $c = \Omega(\sqrt{\log n})$. The reduction is based on the well-known Johnson-Lindenstrauss Lemma [8]. We introduce $d/\log(n)$ linear mappings, each reduces the dimension of the original problem. Each mapping roughly preserves the length of the vector and additionally at least one of them does not increase the length of the input vector. The property of not increasing the length of the vector is crucial. For two 'close' vectors $x, y \in \mathbb{R}^d$: $\|x - y\|_2 < 1$ and a linear mapping A, Ax and Ay are 'close' if and only if $\|Ax - Ay\|_2 = \|A(x - y)\|_2 < 1$, so A maps a 'small' vector $x - y$, to a 'small' vector $A(x - y)$.

In Section 5, we show further reductions, which enable us to relax the constraint to $c = \omega(\sqrt{\log \log n})$. We extend the reduction from Section 4 by using a number of mapping families. This leads to an interesting sub-problem of solving the approximate nearest neighbors in $(\mathbb{R}^k)^L$, for norm $max\text{-}l_2(x) := \max_{1 \le i \le L} \|x_i\|_2$ and the induced metric. This norm is present in literature and was denoted as max–product or l_∞–product. Apparently, the c-approximate nearest neighbor search in $max\text{-}l_2$ might be solved using the LSH functions family introduced in [13].

This series of reductions leads to our final results. First we reduce the problem to a number of $\mathsf{NN}_{\mathrm{wfn}}(\mathcal{O}(c), \mathcal{O}(\log n))$ instances, each of which is further reduced to a number of problems in $max\text{-}l_2$, which in turn are solved using the LSH functions presented in [13]. sectionNotations

The c-approximate nearest neighbors search without false negatives with parameter $c > 1$ and the dimension of the space equal to d, will be denoted as $\mathsf{NN}_{\mathrm{wfn}}(c, d)$. The expected query and pre-processing time complexities of $\mathsf{NN}_{\mathrm{wfn}}(c, d)$ will be denoted as $query(c, d)$ and $preproc(c, d)$ respectively. The input set will be denoted as X and it will always be assumed to contain n points. W.l.o.g, throughout this work we will assume, that R – a given radius equals 1 (otherwise, all vectors' lengths might be rescaled by $1/R$). The $\tilde{\mathcal{O}}()$ denotes the complexity up to the poly logarithmic factors i.e., $\tilde{\mathcal{O}}(f(n)) = \mathcal{O}(f(n)poly(log(n)))$. $\|\cdot\|_2$ denotes the standard norm in l_2, i.e., $\|x\|_2 = (\sum_i |x_i|^2)^{1/2}$. The $f(n) = \omega(g(n))$ means that f dominates g asymptotically, i.e., $g(n) = o(f(n))$.

4 Algorithm for $c = \Omega(\sqrt{\log n})$

The basic idea is the following: we will introduce a number of linear mappings to transform the d-dimensional problem to a number of problems with dimension reduced to $\mathcal{O}(\log n)$. Then we use the algorithm introduced in [13], to solve these problems in the space with the reduced dimension.

We will introduce d/k^2 linear mappings $A^{(1)}, A^{(2)}, \ldots, A^{(d/k)} : R^d \to R^k$, where $k < d$ and show the following properties:

1. for each point $x \in \mathbb{R}^d$, such that $\|x\|_2 \leq 1$, there exists $1 \leq i \leq d/k$, such that $\|A^{(i)}x\|_2 \leq 1$,
2. for each point $x \in \mathbb{R}^d$, such that $\|x\|_2 \geq c$, where $c > 1$, the probability that there exists $1 \leq i \leq d/k$, such that $\|A^{(i)}x\|_2 \leq 1$ is bounded.

The property 1. states, that for a given 'short' vector (with a length smaller than 1), there is always at least one mapping, which transforms this vector to a vector of length smaller than 1. Moreover, we will show, that there exists at least one mapping $A^{(i)}$, which does not increase the length of the vector, i.e., such that $\|A^{(i)}x\|_2 \leq \|x\|_2$. The property 2. states, that we can bound the probability of a 'long' vector ($\|x\|_2 > c$), being mapped to a 'short' one ($\|A^{(i)}x\|_2 \leq 1$). Using the standard concentration measure arguments, we will prove that this probability decays exponentially in k.

4.1 Linear mappings

In this section, we will introduce linear mappings satisfying properties 1. and 2. Our technique will depend on the concentration bound used to prove the classic Johnson-Lindenstrauss Lemma. In Lemma 2, we take a random vector and project it to the first k vectors of the standard basis of $\mathbb{R}^d$. In our settings, we will project the given vector to a random orthonormal basis which gives the same guaranties. The mapping $A^{(i)}$ consists of k consecutive vectors from the random basis of the $\mathbb{R}^d$ space scaled by $\sqrt{\frac{d}{k}}$. The following reduction describes the basic properties of our construction:

▶ **Lemma 3** (Reduction Lemma). *For any parameter $\alpha \geq 1$ and $k < d$, there exist d/k linear mappings $A^{(1)}, A^{(2)}, \ldots, A^{(d/k)}$, from $\mathbb{R}^d$ to $\mathbb{R}^k$, such that:*

1. *for each point $x \in \mathbb{R}^d$ such that $\|x\|_2 \leq 1$, there exists $1 \leq i \leq d/k$, such that $\|A^{(i)}x\|_2 \leq 1$,*
2. *for each point $x \in \mathbb{R}^d$ such that $\|x\|_2 \geq c$, where $c > 1$, for each i: $1 \leq i \leq d/k$, we have*

$$\mathbb{P}\left[\|A^{(i)}x\|_2 \leq \alpha\right] < e^{-k(\frac{c-\alpha}{2c})^2}.$$

Proof. Let $a_1, a_2, \ldots, a_d$ be a random basis of R^d. Each of the $A^{(i)}$ mappings is represented by a $k \times d$ dimensional matrix. We will use $A^{(i)}$ for denoting both the mapping and the corresponding matrix. The jth row of the matrix $A^{(i)}$ equals $A^{(i)}_j = \sqrt{\frac{d}{k}} a_{(i-1)k+j}$. In other words, the rows of $A^{(i)}$ consist of k consecutive vectors from the random basis of the $\mathbb{R}^d$ space scaled by $\sqrt{\frac{d}{k}}$.

To prove the first property, observe that $A = \sum_{i=1}^{d} \langle a_i, x\rangle^2 \leq 1$, since the distance is independent of the basis. Assume on the contrary, that for each i, $\|A^{(i)}_2 x\| > 1$. It follows that $d \geq dA = k\sum_{i=1}^{d} \|A^{(i)}x\|_2^2 > d$. This contradiction ends the proof of the first property.

For any $x \in \mathbb{R}^d$, such that $\|x\|_2 > c$, the probability:

$$\mathbb{P}\left[\|A^{(i)}x\|_2 \leq \alpha\right] = \mathbb{P}\left[\frac{\|A^{(i)}x\|_2^2}{c^2} \leq (\frac{\alpha}{c})^2\right] \leq \mathbb{P}\left[\frac{\|A^{(i)}x\|_2^2}{\|x\|_2^2} \leq (\frac{\alpha}{c})^2\right].$$

[2] For simplicity, let us assume that k divides d, this can be achieved by padding extra dimensions with 0's.

Using the fact that $\log x < x - 1 - (x-1)^2/2$ for $x < 1$ and Lemma 2, the above is bounded as follows:

$$\mathbb{P}\left[\frac{\|A^{(i)}x\|_2^2}{\|x\|_2^2} \le (\frac{\alpha}{c})^2\right] \le \exp\big(-\frac{k}{4}(1-(\frac{\alpha}{c})^2)^2\big) \le e^{-k(\frac{c-\alpha}{2c})^2},$$

which completes the proof. ◀

4.2 Algorithm

The algorithm works as follows: for each i, we project $\mathbb{R}^d$ to $\mathbb{R}^k$ using A_i and solve the corresponding problem in the smaller space. For each query point, we need to merge the solutions obtained for each subproblem. This results in reducing the $\mathsf{NN}_{\text{wfn}}(c,d)$ to d/k instances of $\mathsf{NN}_{\text{wfn}}(\alpha,k)$.

▶ **Lemma 4.** *For $1 < \alpha < c$ and $k < d$, the $\mathsf{NN}_{\text{wfn}}(c,d)$ can be reduced to d/k instances of the $\mathsf{NN}_{\text{wfn}}(\alpha,k)$. The expected pre-processing time equals $\mathcal{O}(d^2n + d/k\ preproc(\alpha,k))$ and the expected query time equals $\mathcal{O}(d^2 + d/k\ e^{-k(\frac{c-\alpha}{2c})^2}n + d/k\ query(k,\alpha))$.*

Proof. We use the assumption that $k < d < n$ to simplify the complexities. The pre-processing time consists of:

- d^3: the time of computing a random orthonormal basis of $\mathbb{R}^d$.
- d^2n: the time of changing the basis to $a_1, a_2, \ldots, a_d$.
- dnk: the time of computing $A^{(i)}x$ for all $1 \le i \le d$ and for all n points.
- $d/k\ preproc(\alpha,k)$: the expected pre-processing time of all subproblems.

The query time consists of:

- d^2: the time of changing the basis to $a_1, a_2, \ldots, a_d$.
- $d/k\ e^{-k(\frac{c-\alpha}{2c})^2}n$: the expected number of false positives (by Lemma 3).
- $d/k\ query(k,\alpha)$: the expected query time of all subproblems. ◀

The following corollary simplifies the formulas used in Lemma 4 and shows that if $\frac{c}{c-\alpha}$ is bounded, the $\mathsf{NN}_{\text{wfn}}(c,d)$ can be reduced to a number of problems of dimension $\log n$ in an efficient way. Namely, setting $k = \big(\frac{2c}{c-\alpha}\big)^2 \log n$ we get:

▶ **Corollary 5.** *For any $1 \le \alpha < c$ and $\gamma \log n < d$, the $\mathsf{NN}_{\text{wfn}}(c,d)$ can be reduced to $d/\log n$ instances of the $\mathsf{NN}_{\text{wfn}}(\alpha,\gamma\log n)$, where $\gamma = \big(\frac{2c}{c-\alpha}\big)^2$ and:*

$$query(c,d) = \mathcal{O}(d^2 + d/\log(n)\ query(\alpha,\gamma\log n)),$$

$$preproc(c,d) = \mathcal{O}(d^2n + d/\log(n)\ preproc(\alpha,\gamma\log n)).$$

Combining the above corollary with the results introduced in [13], we can achieve the algorithm with the polynomial pre-processing time and the sub-linear query time. Theorem 1 states, that for any $c > 2\sqrt{d}$, the $\mathsf{NN}_{\text{wfn}}(c,d)$ can be solved in the $\mathcal{O}(n^{1+\frac{\log 3}{\log(c/\tilde{\tau})}})$ pre-processing time and the query time equal to $\mathcal{O}(d|P| + d\log n + d^2)$, where P is the size of the result set and $\tilde{\tau} = 2\sqrt{d}$. Altogether, setting $\alpha = c/2$ in Corollary 5, we get[3]:

[3] The author of [13] presented multiple algorithms giving different trade-offs between the pre-processing time and the query time. Particularly, the algorithm with the $\mathcal{O}(n\log n)$ processing time and the sub-linear query time was presented. The same can be done for Theorem 6. We omit this to avoid the unnecessary complexity.

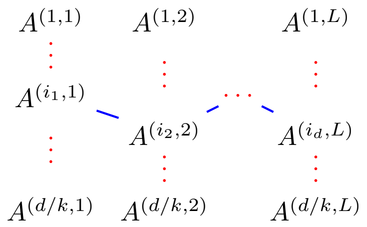


Figure 1 Each column describes one family of linear mappings, constructed based on one random, orthonormal basis. The blue path describes one combination of mappings.

▶ **Theorem 6.** *The* $\mathsf{NN}_{\text{wfn}}(c, d)$ *can be solved for any* $c > \tilde{\kappa} = 16\sqrt{\log n}$ *with:*

$$query(c, d) = \mathcal{O}(d|P| + d\log n + d^2)$$

$$preproc(c, d) = \mathcal{O}(dn^{1+\frac{\ln 3}{\log(c/\tilde{\kappa})}} / \log(n)).$$

The time complexity of the algorithm is the same as for $c = \Omega(\sqrt{d})$, the pre-processing time is larger by a factor of $d/\log(n)$.

5 The algorithm for $c = \omega(\sqrt{\log(\log(n))})$

In this section we give another algorithm which works for $c = \omega(\sqrt{\log(\log(n))})$. Lemma 2 implies that the $\mathsf{NN}_{\text{wfn}}(c, d)$ problem can be reduced to $d/\log(n)$ problems of dimension logarithmic in n. In order to reduce the dimension even more, we will employ L independent families of linear mappings introduced in Section 4. In each of the families, there is at least one mapping, which does not increase the length of the input vector. As a result, there exists a combination of L mappings (each mapping taken from a distinct family) which do not increase the input vector length. Also, for any combination of L mappings, the probability that all the mappings transform a 'long' vectors to a 'short; one can be bounded. The structure of the mappings is presented in Figure 1.

To formalize the above line of thinking, we introduce the following lemma:

▶ **Lemma 7.** *For any natural number* $L > 0$*, there exist* d/k L *linear mappings* $A^{(i,j)} : R^d \to R^k$*, where* $k < d$*,* $1 \le i \le d/k$ *and* $1 \le j \le L$*, such that*

1. *for each point* $x \in R^d$ *which satisfies* $\|x\|_2 \le 1$*, there exist* $1 \le i_1, i_2, \ldots, i_L \le d$ *such that* $\|A^{(i_j,j)}x\|_2 \le 1$*, for each* $1 \le j \le L$*.*
2. *for each point* $x \in R^d$ *which satisfies* $\|x\|_2 \ge c$*, where* $c > 1$*, for each* $i_1, i_2, \ldots, i_L$*:* $1 \le i_1, i_2, \ldots, i_L \le d/k$*, we have*

$$\mathbb{P}\left[\forall_j : \|A^{(i_j,j)}x\|_2 \le \alpha\right] < \exp\big(-\frac{kL}{4}(\frac{c-\alpha}{c})^2\big).$$

Proof. For each j: $1 \le j \le L$ we independently sample the orthonormal basis of $\mathbb{R}^d$: $a_1, a_2, \ldots, a_d$. The $A^{(i,j)}$ will be created in the same way as in Lemma 3, namely, the t–th row of $A^{(i,j)}$ equals $A_t^{(i,j)} = \sqrt{\frac{d}{k}} a_{(i-1)k+t}$. The properties (1) and (2) follow directly from Lemma 3. ◀

In order to employ Lemma 7 for a given query q, we need to be able to find all points in X such that a given combination of mappings transforms these points and the query point to 'close' vectors. In other words, we need to find all c-approximate nearest neighbors for the transformed input set $\tilde{X} \subset (\mathbb{R}^k)^L$ in the space equipped with metric: $max\text{-}l_2(x, y) = \max_{1\le i\le L}(\|x_i - y_i\|)$, which is formally defined as follows:

▶ **Definition 8** (the c-approximate nearest neighbor search in $max\text{-}l_2$)**.** The $max\text{-}l_2\_NN(c, L, k)$ is defined as follows: given a query point $q \in (\mathbb{R}^k)^L$ and a set $\tilde{X} \subset (\mathbb{R}^k)^L$ of n input points, find all input points, such that for each $1 \le i \le L$: $\|q_i - \tilde{x}_i\|_2 \le 1$. Moreover, each $\tilde{x}$ satisfying $\forall_i \|q_i - \tilde{x}_i\|_2 \le c$, might be returned as well. Finally, each x such that $\exists_i \|q_i - \tilde{x}_i\| > c$, must not be returned.

Using the construction from Lemma 7, the $\mathsf{NN}_{\text{wfn}}(c, d)$ problem can be reduced to d^L instances of the $max\text{-}l_2\_NN(\alpha, L, k)$. Each of the instances is represented by indices: $\{i_1, i_2, \dots, i_L\}$ and the corresponding mappings $A^{(i_j,j)}$ for $1 \le j \le L$. Each input point $\tilde{x} \in \tilde{X}$ comes from the point $x \in X$ by applying the mappings: $\tilde{x} = (A^{(i_1,1)}x, \dots, A^{(i_L,L)}x)$. Similarly, the query point q, in $max\text{-}l_2$, is created from the query point q in $\mathsf{NN}_{\text{wfn}}(c, d)$, as $(A^{(i_1,1)}q, \dots, A^{(i_L,L)}q)$.

▶ **Lemma 9.** *The $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to $(d/k)^L$ instances of $max\text{-}l_2\_NN(\alpha, L, k)$. The expected pre-processing time equals:*

$$\mathcal{O}(Ld^2 n + (d/k)^L\ preproc_{max\text{-}l_2}(\alpha, L, k))$$

and the expected query time equals:

$$\mathcal{O}(Ld^2 + (d/k)^L e^{-kL(\frac{c-\alpha}{2c})^2} n + (d/k)^L\ query_{max\text{-}l_2}(\alpha, L, k)).$$

The proof of the Lemma is analogical to the proof of Lemma 4. The following corollary presents the simplified version of Lemma 9. Setting $k = \lceil L^{-1}\big(\frac{2c}{c-\alpha}\big)^2 \log n \rceil$ we get:

▶ **Corollary 10.** *For any $1 \le \alpha < c$, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to d^L instances of $max\text{-}l_2\_NN(\alpha, L, L^{-1}\gamma \log n)$, where $\gamma = \big(\frac{2c}{c-\alpha}\big)^2$ and:*

$$query(c, d) = \mathcal{O}(Ld^2 + (d/\log(n))^L\ query_{max\text{-}l_2}(\alpha, L, \lceil L^{-1}\gamma \log n \rceil)),$$

$$preproc(c, d) = \mathcal{O}(Ld^2 n + (d/\log(n))^L\ preproc_{max\text{-}l_2}(\alpha, L, \lceil L^{-1}\gamma \log n \rceil)).$$

The $max\text{-}l_2\_NN(\alpha, L, k)$ can be trivially solved by dealing with each of the L-dimensional $\mathsf{NN}_{\text{wfn}}(\alpha, k)$ problems separately. Unfortunately, this gives unacceptable complexities. In order to improve complexity of algorithm for the $\mathsf{NN}_{\text{wfn}}(c, d)$ problem, we need to be able to solve the $max\text{-}l_2\_NN$ more efficiently.

5.1 Solving the c-approximate nearest neighbors in $max\text{-}l_2$

In order to solve this problem, we use the standard LSH technique based on the hash functions $\tilde{h}$ introduced in [13] defined as follows:

$$\tilde{h}(x) = \lfloor \langle w, x \rangle \rfloor\,,\ \text{where } w \text{ is a random vector from the unit sphere } \mathbb{S}^{(d-1)}.$$

We consider two hashes to be 'close' if $|\tilde{h}(x) - \tilde{h}(x')| \le 1$. Based on $\tilde{h}$, we introduce a new hash function g. Each of the input points is hashed by g and the reference to this point is kept in a single hash map. For a given query point, we examine all input points which are hashed to the same value as the query point.

Namely, each $\tilde{x} \in (\mathbb{R}^k)^L$ will be hashed by $g(\tilde{x}) := (g_1(\tilde{x}_1), \dots, g_L(\tilde{x}_L))$, where $g_i(x) := (\tilde{h}_1(x), \tilde{h}_2(x), \dots, \tilde{h}_w(x))$ is a hash function defined as a concatenation of w random LSH functions $\tilde{h}$. The function g can be also seen as a concatenation of wL random hash functions $\tilde{h}$. If two points are 'close' in the considered $max\text{-}l_2$ metric, then g transforms these points to hashes $p^{(1)}, p^{(2)} \in \mathbb{Z}^{wL}$, such that $|p_i^{(1)} - p_i^{(2)}| \le 1$ for all $i \in wL$. The pre-processing algorithm is summarized in the following pseudocode:

Algorithm 1: The pre-processing algorithm

```
Data: $X \subset (\mathbb{R}^k)^L$ - the set of $n$ input points
Result: $H : \mathbb{Z}^{wL} \to 2^X$ - the hash map storing for each hash $\alpha \in \mathbb{Z}^{wL}$ the subset of
        input points with hashes close to $\alpha$
$H = \emptyset$;
for $x \in X$ do
    $\alpha = g(x)$;
    for $\alpha'$ such that $\|\alpha - \alpha'\|_\infty \le 1$ do
        $H(\alpha').push(x)$;
    end
end
```

The query algorithm consists of examining the bucket for $g(query\_point)$:

Algorithm 2: The query algorithm

```
Data: $q \in (\mathbb{R}^k)^L$ - the query point
Result: $P \subset X$ - the set of neighbors of $q$
$P = \emptyset$;
for $x \in H(g(q))$ do
    if $x$ is a neighbor of $q$ then
        $P.push(x)$;
    end
end
```

The following theorem describes the above algorithm:

▶ **Theorem 11.** *For $L = o(\log n)$ and $c > 2\sqrt{k}$, the max-l_2_$NN(c,L,k)$ can be solved in the $\mathcal{O}(kL|P| + k \ln n + Lk^2)$ query time and $\mathcal{O}(n^{1+\frac{\ln 3}{\ln(c/\tilde{\kappa})}})$ pre-processing time complexity for $\tilde{\kappa} = 2\sqrt{k}$.*[4]

Proof. Let us start with the key properties of the LSH family.

▶ **Observation 12** ('Close' points have 'close' hashes for $\tilde{h}$ (Observation 5 in [13])). For $x, y \in \mathbb{R}^d$, if $\|x - y\| < 1$ then $\forall_{\tilde{h}} |\tilde{h}(x) - \tilde{h}(y)| \le 1$.

▶ **Lemma 13** (The probability of false positives for $\tilde{h}$ (Lemma 2 in [13])). *For $x, y \in \mathbb{R}^d$ and $c > \tilde{\tau} = 2d^{1/2}$ such that $\|x - y\| > c$, it holds:*

$$\tilde{p}_{\mathsf{fp}} = \mathbb{P}\left[|\tilde{h}(x) - \tilde{h}(y)| \le 1\right] < \tilde{\tau}/c.$$

Since we consider two hashes to be 'close', when they differ at most by one (see Observation 12), for each hash $\alpha \in \mathbb{Z}^{wL}$ we need to store the reference to every point, that satisfies $\|\alpha - g(x)\|_\infty \le 1$. Thus, the hash map size is $\mathcal{O}(n3^{wL})$. Computing a single $\tilde{h}$ function in $\mathbb{R}^k$ takes $\mathcal{O}(k)$, so evaluating the $g(x)$ for $x \in (\mathbb{R}^k)^L$ takes $O(wkL)$. The pre-processing consists of computing the 3^{wL} hashes for each point in the input set. The query consists of computing the hash of the query point, looking up all the points with colliding hashes, filtering out the false positives and returning the neighbors.

[4] Theorem 11 might be generalized, to any $p \in [1, \infty]$. The generalization is done by applying hash functions suited for l_p. Such hash functions where introduced in [13].

It is easy to derive the following complexities:

For any $c > 2k^{1/2}$ and the number of iterations $w \geq 1$, there exists a *max-l_2_NN*(c,L,k) algorithm with the following properties:

- the pre-processing time: $\mathcal{O}(n(wkL + 3^{wL}))$, where wkL is the time needed to compute the $g(input\_point)$ and the $\mathcal{O}(3^{wL})$ is the number of the updated hashes for one input point,
- the expected query time: $\mathcal{O}(kL(|P| + w + n\tilde{\mathsf{p}}_{\mathsf{fp}}{}^{wL}))$, where wkL is the time needed to compute the $g(query\_point)$, $n\tilde{\mathsf{p}}_{\mathsf{fp}}{}^{wL}$ is the number of false positives which need to be ignored, $|P|$ denotes the size of the result set. For each of the candidates, we need to perform a check of complexity $\mathcal{O}(kL)$ to classify the point as a true positive or a false positive.

Above $\tilde{\mathsf{p}}_{\mathsf{fp}} = \tilde{\tau}/c$ (see Lemma 13).

The number of iterations w can be chosen arbitrarily, so we will choose the optimal value. Denote $a = -\ln \tilde{\mathsf{p}}_{\mathsf{fp}}$ and $b = \ln 3$, then set w to be:

$$w = \left\lceil \frac{\ln \frac{na}{k}}{a} L^{-1} \right\rceil.$$

Let us assume that n is large enough so that $w \geq 1$. Then, using the fact that $x^{1/x}$ is bounded for $x > 0$ we have:

$$3^{wL} \leq 3 \cdot (3^{\ln \frac{na}{k}})^{1/a} = 3 \cdot (\frac{na}{k})^{b/a} = 3 \cdot (\frac{n}{k})^{b/a},$$

$$n\tilde{\mathsf{p}}_{\mathsf{fp}}{}^{wL} = ne^{-awL} \leq ne^{-a\frac{\ln(\frac{na}{k})}{a}} = \frac{k}{a}.$$

Hence, for constant c the expected query time is $\mathcal{O}(kL|P| + k\ln n + Lk^2)$. Subsequently, the pre-processing time is: $\mathcal{O}(n3^{wL}) = \mathcal{O}(n^{1+b/a})$. Substituting a, b and p_{fp} values gives the needed complexity guaranties. ◀

5.2 Putting it All Together

In order to achieve an efficient algorithm for $c = \omega(\log(\log(n)))$, we will make a series of reductions. First, using Corollary 5, we reduce our problem to a number of $\mathsf{NN}_{\text{wfn}}(\mathcal{O}(c), \mathcal{O}(\log n))$ problems. Next, these problems are reduced to a number of *max-l_2_NN* problems with dimension k of $O(\log\log n)$. In the end, we use Theorem 11 to solve the *max-l_2_NN*.

▶ **Theorem 14.** *The $\mathsf{NN}_{\text{wfn}}(c, d)$ can be solved with:*

- *the pre-processing time $\tilde{\mathcal{O}}(d^2 n + dn^{1+\frac{\ln 3}{\ln(c/\mu)}+1/f(n)})$,*
- *the query time $\tilde{\mathcal{O}}(d^2 + dn^{1/f(n)}|P|)$,*

for any $c > \mu = D\sqrt{f(n)\log\log n}$, where $f(n)$ is any function, which satisfies $1/f(n) = o(1)$ and D is some constant.

Proof. There are two consecutive reductions:

1. By Corollary 5, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to d instances of the $\mathsf{NN}_{\text{wfn}}(\alpha_1, k_1)$.
2. By Corollary 10, the $\mathsf{NN}_{\text{wfn}}(\alpha_1, k_1)$ can be reduced to k_1^L instances of the *max-l_2_NN*(α_2, k_2, L).

Accordingly, we set:

1. $\alpha_1 = c/2$ and $k_1 = \lceil D_1 \log n \rceil$ in the first reduction
2. $\alpha_2 = c/4$, $k_2 = \lceil D_2 L^{-1} \log n \rceil \leq \lceil D_2 f(n) \log\log n \rceil$ and $L = \lceil \frac{\log n}{f(n)\log\log n} \rceil$ in the second reduction.

The constants D_1 and D_2 are chosen to satisfy Corollaries 5 and 10. k_1^L can be bounded in the following way:

$$k_1^L = \lceil D_1 \log n \rceil^L = \tilde{\mathcal{O}}(n^{1/f(n)}) = \tilde{\mathcal{O}}(n^{o(1)}).$$

Substituting the complexities for subproblems gives the final complexities. ◀

The function $f(n)$ may be chosen arbitrarily. Slowly increasing $f(n)$ will be chosen for small c close to $\Theta(\log \log n)$. For larger c, one should choose the maximal possible $f(n)$, to optimize the query time complexity.

6 Conclusion and Future Work

We have presented the c-approximate nearest neighbor algorithm without false negatives in l_2 for any $c = \omega(\sqrt{\log \log n})$. Such an algorithm might work very well for high entropy datasets, where the distances tend to be relatively large (see [11] for more details). Also, we showed that the c-approximate nearest neighbor search in l_2^d may be reduced to d instances of the problem in $l_2^{\log n}$. Hence, further research might focus on the instances with dimension logarithmic in n.

Another open problems are to reduce the time complexity of the algorithm and relax the restrictions on the approximation factor c or proving that these restrictions are essential. We wish to match the time complexities given in [7] or show that the achieved bounds are optimal.

Acknowledgments. We would like to thank Andrzej Pacuk and Kamila Wygocka for meaningful discussion. Also, I would like to thank the anonymous reviewers for many comments which greatly increased the quality of this work.

References

1 Thomas Dybdahl Ahle. Optimal las vegas locality sensitive data structures. *CoRR*, abs/1704.02054, 2017. URL: http://arxiv.org/abs/1704.02054.

2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008. doi:10.1145/1327452.1327494.

3 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 793–801. ACM, 2015. doi:10.1145/2746539.2746553.

4 Bernard Chazelle, Ding Liu, and Avner Magen. Approximate range searching in higher dimension. *Computational Geometry*, 39(1):24–29, 2008. doi:10.1016/j.comgeo.2007.05.008.

5 Piotr Indyk. On approximate nearest neighbors in non-euclidean spaces. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98, November 8-11, 1998, Palo Alto, California, USA*, pages 148–155, 1998. doi:10.1109/SFCS.1998.743438.

6 Piotr Indyk. Uncertainty principles, extractors, and explicit embeddings of l2 into l1. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC'07, pages 615–620, New York, NY, USA, 2007. ACM. doi:10.1145/1250790.1250881.

7 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on*

Theory of Computing, STOC'98, pages 604–613, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276876.

8 William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

9 Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Trans. Comput. Theory*, 6(1):5:1–5:13, March 2014. doi:10.1145/2578221.

10 Andrzej Pacuk, Piotr Sankowski, Karol Wegrzycki, and Piotr Wygocki. Locality-sensitive hashing without false negatives for l_p. In *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, pages 105–118, 2016. doi:10.1007/978-3-319-42634-1_9.

11 Rasmus Pagh. Locality-sensitive hashing without false negatives. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'16, pages 1–9, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2884435.2884436.

12 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2):357–365, December 2005. doi:10.1016/j.tcs.2005.09.023.

13 Piotr Wygocki. On fast bounded locality sensitive hashing. *ArXiv e-prints*, 2017. URL: http://arxiv.org/abs/1704.05902.

Tight Approximation for Partial Vertex Cover with Hard Capacities

Jia-Yau Shiau[1], Mong-Jen Kao[2], Ching-Chi Lin[3], and D.T. Lee[4]

1 **Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan**
janus7799@gmail.com

2 **Institute of Information Science, Academia Sinica, Taipei, Taiwan**
mong@iis.sinica.edu.tw

3 **Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung City, Taiwan**
lincc@mail.ntou.edu.tw

4 **Institute of Information Science, Academia Sinica, Taipei, Taiwan**
dtlee@ieee.org

Abstract

We consider the partial vertex cover problem with hard capacity constraints (Partial VC-HC) on hypergraphs. In this problem we are given a hypergraph $G = (V, E)$ with a maximum edge size f and a covering requirement $\mathcal{R}$. Each edge is associated with a demand, and each vertex is associated with a capacity and an (integral) available multiplicity. The objective is to compute a minimum vertex multiset such that at least $\mathcal{R}$ units of demand from the edges are covered by the capacities of the vertices in the multiset and the multiplicity of each vertex does not exceed its available multiplicity.

In this paper we present an f-approximation for this problem, improving over a previous result of $(2f + 2)(1 + \epsilon)$ by Cheung et al to the tight extent possible. Our new ingredient of this work is a generalized analysis on the extreme points of the natural LP, developed from previous works, and a strengthened LP lower-bound obtained for the optimal solutions.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Approximation Algorithm, Capacitated Vertex Cover, Hard Capacities

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.64

1 Introduction

The capacitated vertex cover problem with hard capacities (VC-HC) models a demand-to-service assignment scenario generalized from the classical vertex cover problem. In this problem, we are given a hypergraph $G = (V, E \subseteq 2^V)$ where each $e \in E$ is associated with a demand $d_e \in \mathbb{R}^{\geq 0}$ and each $v \in V$ is associated with a capacity $c_v \in \mathbb{R}^{\geq 0}$ and an available multiplicity $m_v \in \mathbb{Z}^{\geq 0}$. The objective is to find a vertex multiset, or, cover, represented by a demand assignment function $h \colon E \times V \to \mathbb{R}^{\geq 0}$, such that the followings are met:

(1) $\sum_{v \in e} h_{e,v} = 1$ for all $e \in E$,

(2) $x_v^{(h)} \leq m_v$ for all $v \in V$, where $x_v^{(h)} := \left\lceil \sum_{e\colon e \in E,\, v \in e} (d_e \cdot h_{e,v}) / c_v \right\rceil$,

and $\sum_{v \in V} x_v^{(h)}$ is minimized.

In this paper, we consider VC-HC with partial covering constraints (Partial VC-HC). Instead of dictating that all the demand be covered, we are specified a covering requirement $\mathcal{R}$ to be fulfilled. In particular, the constraint (1) above is replaced by the following two constraints:

$$\text{(1.a)} \quad \sum_{v \in e} h_{e,v} \le 1 \text{ for all } e \in E, \qquad \text{(1.b)} \quad \sum_{e \in E} \sum_{v \in e} d_e \cdot h_{e,v} \ \ge \ \mathcal{R}.$$

Note that, under this notion, VC-HC is the special case for which we have $\mathcal{R} = \sum_{e \in E} d_e$. Moreover, following the convention used in the literature, it is equivalent to specify the slack of coverage $\mathcal{L}$, i.e., the amount of demand that is allowed to be left uncovered.

Background

The vertex cover problem is among the fundamental problems in the study of graph theory and approximation algorithms. It is known that an f-approximation can be obtained via LP rounding and duality, where f is the size of the largest hyperedge. Khot and Regev [13] showed that, assuming the unique game conjecture (UGC), approximating this problem to a ratio better than $f - \epsilon$ is NP-hard for any $\epsilon > 0$ and $f \ge 2$.

The capacitated vertex cover generalizes vertex cover in that demand-to-supply constraints are introduced in addition to the 0/1-covering model. Chuzhoy and Naor [4] considered VC-HC on simple graphs with unit edge demands, i.e., $f = 2$ and $d_e = 1$ for all $e \in E$. They presented a 3-approximation for this problem. They showed that, when the vertices are weighted, minimizing the weighted cost is at least set-cover-hard. Therefore $O(f)$-approximations towards weighted cost model is unlikely even for this simple setting.

Gandhi et al. [5] gave a 2-approximation for VC-HC with unit edge demand by a refined rounding approach to [4]. Saha and Khuller [15] considered general edge demands and presented an $O(f)$-approximation for f-hypergraphs. Cheung et al. [3] presented an improved approach for this problem. They presented a $\left(1 + 2/\sqrt{3}\right)$-approximation for simple graphs and a $2f$-approximation for f-hypergraphs. The gap of approximation for VC-HC was closed recently by Kao and Wong [9, 16] to a tight f-approximation for any $f \ge 2$.

The Partial VC-HC problem was first considered by Cheung et al al in [3] and a $(2f+2)(1+\epsilon)$-approximation in time $O\left(|V|^{1/\epsilon}|E|\right)$ was presented, based on the rounding technique provided for VC-HC in the same paper and an exhaustive search on potential solutions with cardinality $O(1/\epsilon)$.

Related Work

When the available multiplicity of the vertices is unlimited, this problem is known as soft capacitated vertex cover (CVC). This problem was first considered by Guha et al. [7] and a 2-approximation was presented. Kao et al. [8, 10, 11] studied capacitated dominating set problem and presented a series of results for the complexity and approximability of this problem. Bar-Yehuda et al. [2] considered partial CVC and presented a 3-approximation for simple graphs. A tight approximation for Partial CVC was given by Mestre [14], based on a delicate primal-dual scheme developed for a gradually strengthened LP of this problem.

In additional to partial coverage, VC-HC with relaxed multiplicity constraints have also been addressed in the literature, i.e., the constraint $x_v \le m_v$ for all v is relaxed in exchange of affordable solution quality. Grandoni et al. [6] considered VC-HC with weighted cost model and relaxed multiplicity constraints. For the case $m_v = 1$ for all $v \in V$, they showed that, when augmenting the available multiplicity by a factor of f, a cover with cost guarantee of f^2 to the optimal feasible cover can be obtained.[1], The bi-criteria approximation ratio was recently improved by Kao et al. [12] to $\left(k, (1+\frac{1}{k-1})(f-1)\right)$ for any $k \ge 2$ and arbitrary vertex multiplicities.

Our Contribution and Discussion.

In this paper we consider the partial VC-HC problem and present tight approximation result for this problem. Our main result is the following theorem.

▶ **Theorem 1.** *We can compute an f-approximation for partial VC-HC in polynomial time, where f is the size of the largest hyperedge.*

This improves over the previous ratio of $(2f+2)(1+\epsilon)$ in [3] for Partial VC-HC to the tight extent. Our algorithm builds upon the iterative rounding technique developed for the VC-HC problem in our previous works [9, 16]. In each iteration, our algorithm modifies the current instance based on the optimal extreme point solution of the current working LP. When certain structural property is attained for the gradually modified instance, the algorithm rounds up the solution and terminates.

In contrast to the previous works for VC-HC [9, 16], our new ingredient in this work comes in two-fold. The first one is a strengthened LP lower-bound for the optimal solution. Surprisingly, the natural LP formulation for Partial VC-HC, which performs arbitrarily badly even for very simple settings, can be tuned to give tight lower-bounds for the optimal solutions. This allows us to get rid of the exhaustive search step used in [3], which inevitably brings in an undesirable ϵ-dependency in their result.

Our second ingredient for this work is a new insight on the analysis of extreme point solutions of the extended natural LP. Instead of reducing instances of Partial VC-HC for f-hypergraphs into instances of VC-HC for $(f+1)$-hypergraphs and introducing an extra $O(1)$ constant in the approximation ratio, as was done in [3], we analyze the extreme point solution for the original instance directly. This keeps the possibility of tight approximation result alive. Together this gives our tight f-approximation for Partial VC-HC.

The rest of this paper is organized as follows. In Section 2 we formally define Partial VC-HC and the LP relaxation we will be using throughout this paper. In Section 3 we present our tight approximation algorithm for this problem. We conclude with an extension of our result and discussion for future direction in Section 4.

2 Problem Statement and LP Relaxation

In this section we formally define the Partial VC-HC problem and introduce the LP relaxation we will be using. Throughout this paper, we use $G = (V, E)$ to denote a hypergraph with vertex set V and edge set $E \subseteq 2^V$. Under this context, each hyperedge $e \in E$ is represented by the set of its incident vertices, which is a vertex subset of V. In other words, $e \subseteq V$ for all $e \in E$. We use $f := \max_{e \in E} |e|$ to denote the size of the largest hyperedge in the considered graph.

For any edge subset $\mathcal{E} \subseteq E$, we use $\mathcal{E}[v]$ to denote the set of edges in $\mathcal{E}$ that are incident to the vertex v. Formally, $\mathcal{E}[v] := \left\{ e : e \in \mathcal{E} \text{ such that } v \in e \right\}$.

[1] The bi-approximation ratio of [6] is updated due to the difference between the considered models.

2.1 Partial Vertex Cover with Hard Capacities (Partial VC-HC)

In this problem we are given a hypergraph $G = (V, E)$ and a covering requirement $\mathcal{R}$, where each $e \in E$ is associated with a demand d_e and each $v \in V$ is associated with a capacity c_v and an (integral) available multiplicities m_v.

A solution to this problem consists of a demand assignment function $h\colon E \times V \to \mathbb{R}^+ \cup \{0\}$, where $h_{e,v}$ denotes the fraction of the edge e that is assigned to the vertex v. The multiplicity of each vertex v given by h is denoted $x_v^{(h)} := \left\lceil \sum_{e \in E[v]} (d_e \cdot h_{e,v})/c_v \right\rceil$. The assignment h is *feasible* if (1) $\sum_{v \in e} h_{e,v} \le 1$ for all $e \in E$, (2) $\sum_{e \in E} \sum_{v \in e} d_e \cdot h_{e,v} \ge \mathcal{R}$, and (3) $x_v^{(h)} \le m_v$ for all $v \in V$.

Given an instance $\Pi = (V, E, \mathcal{R}, \mathbf{c}, \mathbf{m}, \mathbf{d})$ as described above, the problem of Partial VC-HC is to find a feasible assignment h such that $\sum_{v \in V} x_v^{(h)}$ is minimized. Without loss of generality, we assume that the input graph G admits a feasible assignment since this condition can be checked via a max-flow computation.

For the ease of presentation, we also use $\mathcal{L} := \sum_{e \in E} d_e - \mathcal{R}$ to denote the amount of demand that can be left unassigned. Furthermore, for each $e \in E$, we use $h_e^\phi := 1 - \sum_{v \in e} h_{e,v}$ to denote the fraction that is left unassigned for edge e.

We remark that, when d_e and c_v are integer-valued for all $e \in E$, $v \in V$, by the integrality of b-matching polytope, any fractional assignment can be turned into an integral assignment, i.e., $d_e \cdot h_{e,v} \in \mathbb{Z}^{\ge 0}$ for all e, v, using a standard integer flow computation.

2.2 LP relaxation for Partial VC-HC

Given an instance $\Pi = (V, E, \mathbf{c}, \mathbf{m}, \mathbf{d})$ of Partial VC-HC, for

(1) a vertex subset $\mathcal{V} \subseteq V$,

(2) an edge subset $\mathcal{E} \subseteq E$,

(3) a residue fraction of the edges $\boldsymbol{r}$ to be covered, where $\mathbf{0} \le \boldsymbol{r} \le \mathbf{1}$, and

(4) an additional lower-bound $\boldsymbol{\ell}$, where $\mathbf{0} \le \boldsymbol{\ell} \le \mathbf{m}$, on the multiplicity of the vertices,

we consider the following relaxation, with $\Psi := (\mathcal{V}, \mathcal{E}, \boldsymbol{r}, \boldsymbol{\ell}, \mathbf{c}, \mathbf{m}, \mathbf{d})$ being a parameter tuple:

$$\min_{(\mathbf{x},\mathbf{h})} \quad \sum_{v \in \mathcal{V}} x_v \qquad \text{LP}(\Psi)$$

s.t.

$$\sum_{v \in e \cap \mathcal{V}} h_{e,v} + h_e^\phi = r_e, \qquad \forall e \in \mathcal{E} \tag{1a}$$

$$\sum_{e \in \mathcal{E}[v]} d_e \cdot h_{e,v} \le c_v \cdot x_v, \qquad \forall v \in \mathcal{V} \tag{1b}$$

$$\sum_{e \in \mathcal{E}} d_e \cdot h_e^\phi \le \mathcal{L}, \tag{1c}$$

$$\ell_v \le x_v \le m_v, \qquad \forall v \in \mathcal{V} \tag{1d}$$

$$0 \le h_{e,v} \le x_v, \qquad \forall e \in \mathcal{E},\ v \in e \cap \mathcal{V} \tag{1e}$$

$$0 \le h_e^\phi, \qquad \forall e \in \mathcal{E} \tag{1f}$$

Since each of the variables $h_{e,v}$ and x_v is bounded from both below and above, the feasible region of LP(Ψ) is a polytope, and the reference to its extreme points is well-defined.

Throughout this paper, for a given instance Π of Partial VC-HC, a number of different parameter tuples will be considered. However, since $\mathbf{m}$ and $\mathbf{d}$ will remain the same in every considered tuple, we will use $(\mathcal{V}, \mathcal{E}, \boldsymbol{r}, \boldsymbol{\ell}, \mathbf{c})$ to denote the parameter tuple Ψ for the LP.

2.3 The Integrality Gap

We have introduced an extended natural LP relaxation for Partial VC-HC in the previous section. It may seem that LP(Ψ) with parameter tuple $\Psi = (V, E, \mathbf{1}, \mathbf{0}, \boldsymbol{c})$ has an unbounded integrality gap for Partial VC-HC.

Consider the following simple example, where we have one vertex v and one edge $e = \{v\}$. The capacity of v is k for some $k \in \mathbb{N}$, the demand of e is also k. The covering requirement $\mathcal{R}$ is 1. (Therefore, $\mathcal{L} = k - 1$.)

The optimal fractional solution to LP(Ψ) has cost at most $1/k$ since we can set $x_v = h_{e,v} = 1/k$ and $h_e^\phi = (k-1)/k$. However, any integral solution has cost at least 1 since the vertex v has to be selected. Therefore the gap between the two solutions can be arbitrarily large, and it seems that we need an LP stronger than LP(Ψ) to manage this problem.

In general, strengthening an LP for improved integrality gap can be a challenging task to accomplish. However, if we use the fact that any integral solution must also have an integral cost, then the lower-bound given by LP(Ψ) for this simple example becomes $\lceil 1/k \rceil = 1$, which matches the integral solution. Although this may seem to be case-dependent, we will show in the rest of this paper that, surprisingly, this strengthened LP lower-bound is sufficient to give a tight f-approximation result for this problem.

3 Tight Approximation for Partial VC-HC

In this section we describe our tight approximation algorithm for Partial VC-HC. In each iteration, the algorithm makes local decisions based on current working LP and modifies the parameter tuple accordingly. When no such decisions are there to be made, it rounds up all vertices unconditionally and stops.

We first introduce notions and operations our algorithm will be using. In Section 3.1 we describe our algorithm in detail. The analysis is provided in Section 3.2.

Basic Notion and Operations

Let $\Psi = (\mathcal{V}, \mathcal{E}, \boldsymbol{r}, \boldsymbol{\ell}, \boldsymbol{c})$ be a parameter tuple and $p = (\boldsymbol{x}, \boldsymbol{h})$ be a feasible solution for LP(Ψ).

▶ **Definition 2** (Supporting edge)**.** For any $e \in \mathcal{E}$ and $v \in e$, we say that edge e *supports* vertex v in the solution p if $0 < h_{e,v} = x_v$.

The idea behind this definition is that, this condition gives information on how the rounding can be done locally. Suppose that $h_{e,v} = x_v$ holds for some e and v. Then from constraint (1b) of LP(Ψ) we know that

$$d_e \cdot h_{e,v} \ \le \ \sum_{e \in \mathcal{E}[v]} d_e \cdot h_{e,v} \ \le \ c_v \cdot x_v$$

Since $h_{e,v} = x_v$, it follows that $d_e \le c_v$. In other words, if we know that the vertex v is to be rounded up eventually, then its capacity will be sufficient for covering the demand of e. This suggests the edge-folding operation for our rounding algorithm.

▶ **Definition 3** (Edge folding). Let e be an edge that supports a vertex v in p. By *folding e into v*, we update Ψ as follows:

(i) Remove e from $\mathcal{E}$ and decrease c_v by d_e.
(ii) Impose the constraint $x_v \ \geq \ 1/f$ to LP(Ψ) by setting $\ell_v = 1/f$.

Notice that, from the discussion above, folding supporting edges into the supported vertices results in no loss in the feasibility and approximation guarantee of the final solution, provided that the supported vertices are to be rounded-up eventually.

▶ **Definition 4** (Vertex down-pinning). We say that a vertex $v \in \mathcal{V}$ is *tight* if $x_v = 1/f$. By pinning down the value of v, we update Ψ as follows:

(i) For each $e \in \mathcal{E}[v]$, decrease r_e by $h_{e,v}$.
(ii) Remove v from $\mathcal{V}$.

Intuitively a vertex is tight when its value hits the minimal extent that still results in no extra loss (in terms of an f-approximation) when rounded up. When a tight vertex is pinned down, we also pin down the assignment from its incident edges. By constraint (1e) it ensures that, for each of its incident edge, say, e, only a small amount of demand, i.e., $\leq 1/f$, is removed from r_e when v is removed from $\mathcal{V}$. This guarantees that e still has a relatively large assignment, i.e., $\geq 1/f$, to one of its remaining incident ends.

3.1 The Algorithm

In the following we describe our tight approximation algorithm for Partial VC-HC in detail. The algorithm begins with the initial parameter tuple $\Psi = (V, E, \mathbf{1}, \mathbf{0}, \boldsymbol{c})$. In each iteration, it proceeds in the four steps described below.

1. Let $\Psi = (\mathcal{V}, \mathcal{E}, \boldsymbol{r}, \boldsymbol{\ell}, \boldsymbol{c})$ denote the current parameter tuple.
   Solve LP(Ψ) for a *basic optimal solution* $p = (\boldsymbol{x}, \boldsymbol{h})$. Let $I := \left\{v \in \mathcal{V} \colon 0 < x_v < \frac{1}{f}\right\}$.
2. If there exists an edge $e \in \mathcal{E}$ that supports some $v \notin I$ in p,
   then fold e into v.
3. If there exists a vertex $v \in \mathcal{V}$ that is tight,
   then pin down the value of v.
4. If any operation is performed in Step 2 or Step 3,
   then restart Step 1.
   Otherwise, round up vertices in V and stop.

Let $\hat{\Psi} = (\hat{\mathcal{V}}, \hat{\mathcal{E}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{\ell}}, \hat{\boldsymbol{c}})$ denote the parameter tuple when the algorithm enters the final rounding step and $\hat{p} = (\hat{x}, \hat{h})$ denote the basic optimal solution computed for $\hat{\Psi}$.

Let $\boldsymbol{h}'$ denote the $\{0,1\}$-assignment function for $E \setminus \hat{\mathcal{E}}$ that indicates the vertex which each edge $e \in E \setminus \hat{\mathcal{E}}$ is folded into. In particular, for each $e \in E \setminus \hat{\mathcal{E}}$ and $v \in e$, the variable $h'_{e,v}$ is 1 if e is folded into v and 0 otherwise. Furthermore, for any $v \in V \setminus \hat{\mathcal{V}}$ and any $e \in \hat{\mathcal{E}}[v]$, let $h''_{e,v}$ denote the assignment value of e to v when v was pinned down.

The final output $(\boldsymbol{x}^*, \boldsymbol{h}^*)$ is defined as follows. For any $v \in V$ and $e \in E[v]$, let

$$x^*_v := \begin{cases} \lceil \hat{x}_v \rceil, & \text{if } v \in \hat{\mathcal{V}}, \\ 1, & \text{otherwise.} \end{cases} \qquad \text{and} \qquad h^*_{e,v} := \begin{cases} \hat{h}_{e,v}, & \text{if } e \in \hat{\mathcal{E}} \text{ and } v \in \hat{\mathcal{V}}, \\ h''_{e,v}, & \text{if } e \in \hat{\mathcal{E}} \text{ but } v \notin \hat{\mathcal{V}}, \\ h'_{e,v}, & \text{otherwise.} \end{cases}$$

Then $(\boldsymbol{x}^*, \boldsymbol{h}^*)$ is output as the approximate solution.

We remark that, since this approach is insensitive to multiple operations, in the actual algorithm we will fold every supporting edge and pin down the value of every tight vertex. Furthermore, ties are broken arbitrarily if an edge supports multiple vertices outside I.

Let Tight-Partial-VC-HC denote the above algorithm. Our tight approximation result is stated in the following theorem:

▶ **Theorem 5.** *On any instance $\Pi = (V, E, \mathcal{R}, \boldsymbol{c}, \boldsymbol{m}, \boldsymbol{d})$ of Partial VC-HC with maximum edge size $f \geq 2$, algorithm Tight-Partial-VC-HC computes an f-approximation $(\boldsymbol{x}^*, \boldsymbol{h}^*)$ in polynomial time.*

3.2 Analysis

In this section we prove Theorem 5. First, since the algorithm iterates only if some edge is folded or some vertex is pinned down, we know that the number of iterations is at most $O(|E| + |V|)$. Let k denote the number of iterations the algorithm repeats before it enters the rounding stage. For $1 \leq i \leq k$, we use the following notations to denote the respective concepts we have in the i^{th} iteration:

- $\Psi^{(i)} = \left(\mathcal{V}^{(i)}, \mathcal{E}^{(i)}, \boldsymbol{r}^{(i)}, \boldsymbol{\ell}^{(i)}, \boldsymbol{c}^{(i)}\right)$: The parameter tuple the algorithm maintains when it enters the i^{th} iteration. Note that $\Psi^{(1)} = (V, E, \mathbf{1}, \mathbf{0}, \boldsymbol{c})$ is the initial tuple and $\Psi^{(k)} = \left(\hat{\mathcal{V}}, \hat{\mathcal{E}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{\ell}}, \hat{\boldsymbol{c}}\right)$ is the final tuple.
- $p^{(i)} = \left(\boldsymbol{x}^{(i)}, \boldsymbol{h}^{(i)}\right)$: The basic optimal solution computed for LP($\Psi^{(i)}$).

From the usage of edge-folding and vertex-pinning operations, it is not difficult to see that Algorithm Tight-Partial-VC-HC indeed produces a feasible solution for Partial VC-HC. We summarize the feasibility of the algorithm in the following lemma.

▶ **Lemma 6.** *Algorithm Tight-Partial-VC-HC outputs a feasible solution $(\boldsymbol{x}^*, \boldsymbol{h}^*)$ for LP($\Psi^{(1)}$). Moreover, $\boldsymbol{x}^*$ is integral.*

To prove Lemma 6, it suffices to show that (1) After each iteration, the modified tuple is feasible for the next iteration, i.e., for any $1 \leq i < k$, the feasible region of LP($\Psi^{(i+1)}$) is not empty. Therefore the access to basic feasible solutions of the LP is always valid. (2) The solution (x^*, h^*) does not violate the constraints of LP($\Psi^{(1)}$).

For statement (1) above, the following lemma shows that, $\left(\boldsymbol{x}^{(i)}, \boldsymbol{h}^{(i)}\right)$ gives a feasible point for LP($\Psi^{(i+1)}$), thereby ensuring that its feasible region will not be empty.

▶ **Lemma 7.** *The solution $\left(\boldsymbol{x}^{(i)}, \boldsymbol{h}^{(i)}\right)$, when restricted to $\mathcal{V}^{(i+1)}$ and $\mathcal{E}^{(i+1)}$, is feasible for LP($\Psi^{(i+1)}$), for any $1 \leq i < k$.*

Lemma 7 and statement (2) are proved by verifying the corresponding LP constraints. In the rest of this section, we establish the approximation guarantee.

The Approximation Guarantee

From Lemma 7 we know that, the sum of fractional values over vertices in V between successive iterations, which is composed of the objective value of the LP and the values of the down-pinned vertices, will form a non-increasing sequence. Therefore the total fractional value we have in each iteration always gives a valid lower-bound to any optimal integral solution. In particular, we have the following lemma.

▶ **Lemma 8.** *For any* $1 \le i \le k$,

$$\left\lceil \frac{1}{f} \cdot \left| V \setminus \mathcal{V}^{(i)} \right| + \sum_{v \in \mathcal{V}^{(i)}} x_v^{(i)} \right\rceil \le OPT,$$

where OPT is the cost of any optimal integral solution.

Proof. Since OPT is integer-valued, it suffices to show that for any $1 \le i \le k$,

$$\frac{1}{f} \cdot \left| V \setminus \mathcal{V}^{(i)} \right| + \sum_{v \in \mathcal{V}^{(i)}} x_v^{(i)} \le OPT.$$

The base case $i = 1$ holds directly since $\mathcal{V}^{(1)} = V$ and since $\boldsymbol{x}^{(1)}$ is optimal for $\mathrm{LP}(\Psi^{(1)})$. For $1 < i \le k$, since $\mathcal{V}^{(i)} \subseteq \mathcal{V}^{(i-1)}$, by Lemma 7 and the optimality of $\boldsymbol{x}^{(i)}$ for $\mathrm{LP}(\Psi^{(i)})$, we have

$$\sum_{v \in \mathcal{V}^{(i)}} x_v^{(i)} \le \sum_{v \in \mathcal{V}^{(i)}} x_v^{(i-1)}.$$

Therefore,

$$\begin{aligned} \frac{1}{f} \cdot \left| V \setminus \mathcal{V}^{(i)} \right| + \sum_{v \in \mathcal{V}^{(i)}} x_v^{(i)} &\le \frac{1}{f} \cdot \left| V \setminus \mathcal{V}^{(i-1)} \right| + \left(\frac{1}{f} \cdot \left| \mathcal{V}^{(i-1)} \setminus \mathcal{V}^{(i)} \right| + \sum_{v \in \mathcal{V}^{(i)}} x_v^{(i-1)} \right) \\ &= \frac{1}{f} \cdot \left| V \setminus \mathcal{V}^{(i-1)} \right| + \sum_{v \in \mathcal{V}^{(i-1)}} x_v^{(i-1)}. \end{aligned}$$

Therefore this lemma follows by an induction on i. ◀

Note that, the rounding cost of vertices in $\left| V \setminus \mathcal{V}^{(k)} \right|$ is properly bounded by its own fractional value since the fractional value of each down-pinned vertex is exactly $1/f$. Therefore it suffices to show that the rounding cost of vertices in $\mathcal{V}^{(k)}$ can be properly bounded as well.

Let $I^{(k)} := \left\{ v \in \mathcal{V}^{(k)} \colon 0 < x_v^{(k)} < \frac{1}{f} \right\}$ and $M^{(k)} := \left\{ v \in \mathcal{V}^{(k)} \colon 0 < x_v^{(k)} = m_v \right\}$ denote the set of vertices with small and large fractional values in $p^{(k)}$, respectively. The following lemma upper-bounds the cardinality of $I^{(k)}$ in terms of the cardinality of $M^{(k)}$.

▶ **Lemma 9.** $\left| I^{(k)} \right| \le \left| M^{(k)} \right| + 1$.

Proof. Recall that $p^{(k)} = (\boldsymbol{x}^{(k)}, \boldsymbol{h}^{(k)})$ is an extreme point solution of $\mathrm{LP}(\Psi^{(k)})$. Therefore, it follows that the rank of the tight constraints we have, i.e., the maximum number of linearly independent constraints we can pick among the inequalities that hold with equality, w.r.t. $p^{(k)}$ equals the total number of variables.

Since there exist no tight vertices in $\mathcal{V}^{(k)}$ nor edges in $\mathcal{E}^{(k)}$ that supports vertices outside $I^{(k)}$ by the design of the algorithm, it follows that, each tight constraint we have in $\mathrm{LP}(\Psi^{(k)})$ w.r.t. $p^{(k)}$ must belong to one of the forms listed in Figure 1.

Modify the tight constraints as follows: Remove all zero-valued varaibles from the constraints of types (2a), (2b), and (2c). Plug the constraints of types (2d) and (2e) back into that of (2a) and (2b) by replacing corresponding x_v with m_v and $h_{e,v}$ with x_v.

Note that, by doing so, we literally remove equal number of variables and linearly independent tight constraints from the above. Therefore, the number of remaining variable still equals the rank of remaining tight constraints.

$$\sum_{v \in e \cap \mathcal{V}^{(k)}} h_{e,v} + h_e^{\phi} = r_e^{(k)}, \qquad e \in \mathcal{E}^{(k)} \tag{2a}$$

$$\sum_{e \in \mathcal{E}^{(k)}[v]} d_e \cdot h_{e,v} = c_v^{(k)} \cdot x_v, \qquad v \in \mathcal{V}^{(k)} \tag{2b}$$

$$\sum_{e \in \mathcal{E}^{(k)}} d_e \cdot h_e^{\phi} = \mathcal{L}, \tag{2c}$$

$$x_v = m_v, \qquad v \in M^{(k)} \tag{2d}$$

$$h_{e,v} = x_v, \qquad e \in \mathcal{E}^{(k)}, \; v \in e \cap I^{(k)} \tag{2e}$$

$$x_v = 0, \qquad v \in \mathcal{V}^{(k)} \tag{2f}$$

$$h_{e,v} = 0, \qquad e \in \mathcal{E}^{(k)}, \; v \in e \cap \mathcal{V}^{(k)} \tag{2g}$$

$$h_e^{\phi} = 0, \qquad e \in \mathcal{E}^{(k)} \tag{2h}$$

Figure 1 Possible tight constraints we have in LP($\Psi^{(k)}$) with reference to the solution $p^{(k)}$.

Define the following sets:

- $H^* := \{ (e,v) : \; e \in \mathcal{E}^{(k)}, \; v \in e \cap I^{(k)}, \; 0 < h_{e,v} = x_v \}$. This corresponds to the non-zero $h_{e,v}$ variables that have been replaced by x_v due to constraints of type (2e).
- $H^+ := \{ (e,v) : \; e \in \mathcal{E}^{(k)}, \; v \in e \cap \mathcal{V}^{(k)}, \; 0 < h_{e,v} < x_v \}$, which corresponds to the set of $h_{e,v}$ variables that remains in the constraints.
- $\mathcal{V}^+ := \{ v : \; v \in \mathcal{V}^{(k)} \setminus M^{(k)}, \; x_v > 0 \}$ and $E^{\phi} := \{ e : \; e \in \mathcal{E}^{(k)}, \; h_e^{\phi} > 0 \}$.

The remaining tight constraints we have, after modification, are of the following forms, where $\tilde{c}_v^{(k)} = c_v^{(k)} - \sum_{e \colon (e,v) \in H^*} d_e$ is the resulting constant after the constraints (2e) are plugged in.

$$\sum_{v \colon (e,v) \in H^+} h_{e,v} + \sum_{v \colon (e,v) \in H^*} x_v + \left[e \in E^{\phi} \right] \cdot h_e^{\phi} = r_e^{(k)}, \qquad e \in \mathcal{E}^{(k)} \tag{3a}$$

$$\sum_{e \colon (e,v) \in H^+} d_e \cdot h_{e,v} = \tilde{c}_v^{(k)} \cdot x_v, \qquad v \in \mathcal{V}^+ \tag{3b}$$

$$\sum_{e \in E^{\phi}} d_e \cdot h_e^{\phi} = \mathcal{L}, \tag{3c}$$

$$\sum_{e \colon (e,v) \in H^+} d_e \cdot h_{e,v} = c_v^{(k)} \cdot m_v, \qquad v \in M^{(k)} \tag{3d}$$

In the following we work with constraints (3a) through (3d). We show that, the cardinality of $I^{(k)}$ is at most the number of constraints of types (3c) and (3d).

Let $\mathcal{C}$ be a maximal collection of linearly independent constraints from (3a) through (3d) and Vars denote the set of remaining variables. It follows that $|\mathcal{C}| = |\text{Vars}| = |H^+| + |\mathcal{V}^+| + |E^{\phi}|$ and the coefficient matrix of $\mathcal{C}$ is of full-rank. Therefore, by identifying a set of pivots of the coefficient matrix, we obtain a bijection $\sigma\colon \text{Vars} \mapsto \mathcal{C}$ such that any variable, say, $a \in \text{Var}$, appears in the constraint $\sigma(a)$.

For the brevity of notations, in the following for any $e \in \mathcal{E}^{(k)}$ we will use e to denote the corresponding type (3a) constraint it represents. Similarly, for any $v \in \mathcal{V}^{(k)}$, we use v to denote the type (3b) or (3d) constraint it represents. We use ϕ to denote the only one type (3c) constraint.

We have the following two properties, obtained from the fact that σ is a bijection.

▶ **Claim 10.** *We have $\sigma(x_v) = v$ for any $v \in \mathcal{V}^+ \setminus I^{(k)}$.*

Proof. Consider any $v \in \mathcal{V}^+ \setminus I^{(k)}$. Since $(e, v) \notin H^*$ for all e, the only constraint that contains x_v is v itself. Therefore σ must map x_v to v. ◀

▶ **Claim 11.** *For any $e \in \mathcal{E}^{(k)}$, either $h_e^\phi > 0$ or $h_{e,v} > 0$ for some $v \in \mathcal{V}^{(k)} \setminus I^{(k)}$.*

Proof. Suppose that $h_e^\phi = 0$. We will argue that $h_{e,v} > 0$ for some $v \in \mathcal{V}^{(k)} \setminus I^{(k)}$.

Since the value of each down-pinned vertex equals $1/f$, the residue fraction of e decreases by at most $1/f$ each time when one of its incident vertices is pinned down. Therefore it follows that

$$r_e^{(k)} \;\ge\; 1 - \frac{1}{f} \cdot \left| e \cap \left(V \setminus \mathcal{V}^{(k)} \right) \right| \;\ge\; \frac{1}{f} \cdot \left| e \cap \mathcal{V}^{(k)} \right|,$$

where the last inequality follows from the fact that $|e| \le f$.
This says,

$$\sum_{v\colon (e,v)\in H^+} h_{e,v} \;+\; \sum_{v\colon (e,v)\in H^*} x_v \;\ge\; \frac{1}{f} \cdot \left| e \cap \mathcal{V}^{(k)} \right|.$$

Therefore, one of the variables in the L.H.S. has value at least $1/f$, and it must be $h_{e,v'}$ for some v' with $(e, v') \in H^+$ since $x_v < 1/f$ for all v with $(e, v) \in H^*$. Since $h_{e,v'} \ge 1/f$, we know that $v' \notin I^{(k)}$ and this claim follows. ◀

Consider any $v \in I^{(k)}$ and the constraint $\sigma(x_v)$. Since the variable x_v appears only in (3a) and (3b), we have the following two cases.

- If $\sigma(x_v) = e$ for some $e \in \mathcal{E}^{(k)}$, then according to Claim 11, either $h_e^\phi > 0$ or there exists $v' \in \mathcal{V}^{(k)} \setminus I^{(k)}$ such that the variable $h_{e,v'}$ appears in e.
  If $h_e^\phi > 0$, then σ must map h_e^ϕ to ϕ, i.e., $\sigma(h_e^\phi) = \phi$, since the variable h_e^ϕ appears only in e and ϕ but e is already mapped to by x_v.
  Similarly, if $h_{e,v'} > 0$ for some $v' \in \mathcal{V}^{(k)} \setminus I^{(k)}$, then it follows that $\sigma(h_{e,v'}) = v'$. Therefore, by Claim 10, v' does not belong to $\mathcal{V}^+ \setminus I^{(k)}$ and v' must belong to $I^{(k)} \cup M^{(k)}$. Hence we know that $v' \in M^{(k)}$.
  Therefore, if $\sigma(x_v) = e$ for some $e \in \mathcal{E}^{(k)}$, then x_v corresponds exclusively to either ϕ or a vertex $v' \in M^{(k)}$.
- If $\sigma(x_v) = v$, then there exists some e such that $(e, v) \in H^+$ since the constraint v is linearly independent from the others, implying that constraint v is non-degenerating. Since $h_{e,v}$ appears only in constraints e and v, it follows that $\sigma(h_{e,v}) = e$. By the same argument as the previous case, we can again associate x_v exclusively with either ϕ or some vertex $v' \in M^{(k)}$.

From the two cases above, we can associate each $v \in I^{(k)}$ exclusively to either ϕ or some vertex in $M^{(k)}$. It follows that $\left|I^{(k)}\right| \le \left|M^{(k)}\right| + 1$ and this lemma is proved. ◀

The following lemma bounds the cost of $\boldsymbol{x}^*$, which is exactly $\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in\mathcal{V}^{(k)}} \left\lceil x_v^{(k)} \right\rceil$, and establishes the approximation guarantee.

▶ **Lemma 12.** *We have*

$$\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in\mathcal{V}^{(k)}} \left\lceil x_v^{(k)} \right\rceil \le f \cdot OPT.$$

Proof. By Lemma 8 it suffices to show that

$$\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in\mathcal{V}^{(k)}} \left\lceil x_v^{(k)} \right\rceil \;\le\; f \cdot \left\lceil \frac{1}{f} \cdot \left|V \setminus \mathcal{V}^{(k)}\right| \;+\; \sum_{v\in\mathcal{V}^{(k)}} x_v^{(k)} \right\rceil. \tag{4}$$

Note that, since $\left\lceil x_v^{(k)} \right\rceil \le f \cdot x_v^{(k)}$ for any $v \in \mathcal{V}^{(k)} \setminus I^{(k)}$, this inequality holds trivially if $I^{(k)} = \emptyset$. In the following we assume that $I^{(k)} \neq \emptyset$.

Pair up the vertices in $I^{(k)}$ with that in $M^{(k)}$ by associating each $v \in I^{(k)}$ a distinct vertex from $M^{(k)}$. By Lemma 9, at most one vertex in $I^{(k)}$ could be unpaired. Without loss of generality, we assume that exactly one vertex, say, u, from $I^{(k)}$ is unpaired. It follows that

$$\sum_{v\in I^{(k)}\cup M^{(k)}} \left\lceil x_v^{(k)} \right\rceil \;\le\; \left\lceil x_u^{(k)} \right\rceil + \sum_{v\in M^{(k)}} (m_v + 1) \;\le\; \left\lceil x_u^{(k)} \right\rceil + f \cdot \sum_{v\in M^{(k)}} m_v, \tag{5}$$

where the last inequality holds since $f \ge 2$ and since $m_v \ge 1$ for all $v \in M^{(k)}$.

Let $W := \mathcal{V}^{(k)} \setminus \left(I^{(k)} \cup M^{(k)}\right)$. Plugging Inequality (5) into the L.H.S. of Inequality (4), since $\sum_{v\in M^{(k)}} m_v$ is integral and can be moved freely outside the ceilings, we can conclude that, Inequality (4) holds if the following statement holds.

$$\frac{1}{f}\cdot\left(\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in W} \left\lceil x_v^{(k)} \right\rceil + \left\lceil x_u^{(k)} \right\rceil \right) \;\le\; \left\lceil \frac{1}{f} \cdot \left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in W} x_v^{(k)} + x_u^{(k)} \right\rceil. \tag{6}$$

In the following we establish Inequality (6). Let p and q denote the integer multiple and the remainder of $\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in W} \left\lceil x_v^{(k)} \right\rceil$ with the modulus f, i.e., we write

$$\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in W} \left\lceil x_v^{(k)} \right\rceil \;=\; p \cdot f + q,$$

where $p, q \in \mathbb{Z}^{\ge 0}$, $0 \le q < f$. The L.H.S. of Inequality (6) simplifies to

$$\begin{aligned} \frac{1}{f}\cdot\left(pf + q + 1\right) \;\le\; p + 1 \;&=\; \left\lceil \frac{1}{f} \cdot \left(\left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in W} \left\lceil x_v^{(k)} \right\rceil \right) + x_u^{(k)} \right\rceil \\ &\le\; \left\lceil \frac{1}{f} \cdot \left|V \setminus \mathcal{V}^{(k)}\right| + \sum_{v\in W} x_v^{(k)} + x_u^{(k)} \right\rceil, \end{aligned}$$

where the second equality holds since $0 < x_u^{(k)} < 1/f$ and the last inequality follows from the fact that $1/f < x_v^{(k)} < m_v$ for each $v \in W$, which implies that $(1/f) \cdot \left\lceil x_v^{(k)} \right\rceil \le x_v^{(k)}$. This proves Inequality (6) and this lemma follows. ◀

4 Conclusion

We conclude with an interesting generalization of Partial VC-HC where we are given, instead of one global coverage requirement, a set of partial coverage constraints to be satisfied. In particular, we are given a function $R\colon 2^E \rightarrow [0,1]$, where for any $A \subseteq E$, $R(A)$ denotes the fraction of demand coverage among the edges in A to be fulfilled.[2]

Let $k = \left| \left\{ A : \; A \subseteq E, \; R(A) > 0 \right\} \right|$ denote the number of partial coverage constraints. By a direct generalization of Lemma 9 combined with the exhaustive search technique used in [3], we can obtain an $(f+k)(1+\epsilon)$-approximation in *poly* $\left(|V|^{k/\epsilon}|E|\right)$ time.

However, it is not yet clear how better guarantees can be obtained nor how the dependency of k could be removed, and this would be an interesting direction to explore. The MFN-type LP relaxations, originally developed for capacitated facility location [1], could be a powerful tool to manage the approximation guarantee.

References

1 Hyung-Chan An, Mohit Singh, and Ola Svensson. Lp-based algorithms for capacitated facility location. *SIAM J. Comput.*, 46(1):272–306, 2017. doi:10.1137/151002320.

2 R. Bar-Yehuda, G. Flysher, J. Mestre, and D. Rawitz. Approximation of partial capacitated vertex cover. *SIAM Journal on Discrete Mathematics*, 24(4):1441–1469, 2010. doi:10.1137/080728044.

3 W.-C. Cheung, M. Goemans, and S. Wong. Improved algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of SODA'14*, 2014. doi:10.1137/1.9781611973402.124.

4 Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM Journal on Computing*, 36(2):498–515, August 2006. doi:10.1137/S0097539703422479.

5 R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approx algorithm for vertex cover with hard capacities. *J. Comput. Syst. Sci.*, 72:16–33, 2006. doi:10.1016/j.jcss.2005.06.004.

6 F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. A primal-dual bicriteria distributed algorithm for capacitated vertex cover. *SIAM J. Comput.*, 38(3), 2008. doi:10.1137/06065310X.

7 Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering. *Journal of Algorithms*, 48(1):257–270, August 2003. doi:10.1016/S0196-6774(03)00053-1.

8 Mong-Jen Kao. *An Algorithmic Approach to Local and Global Resource Allocations.* PhD thesis, National Taiwan University, 2012.

9 Mong-Jen Kao. Iterative partial rounding for vertex cover with hard capacities. In *Proceedings of SODA'17*, pages 2638–2653, 2017. URL: http://dl.acm.org/citation.cfm?id=3039686.3039860.

10 Mong-Jen Kao, Han-Lin Chen, and D.T. Lee. Capacitated domination: Problem complexity and approximation algorithms. *Algorithmica*, November 2013. doi:10.1007/s00453-013-9844-6.

11 Mong-Jen Kao, Chung-Shou Liao, and D. T. Lee. Capacitated domination problem. *Algorithmica*, 60(2):274–300, June 2011. doi:10.1007/s00453-009-9336-x.

[2] Under this notion, Partial VC-HC is the special case for which we have $R(E) := \mathcal{R} / \sum_{e \in E} d_e$ and $R(A) := 0$ for any $A \subset E$.

12 Mong-Jen Kao, Hai-Lun Tu, and D. T. Lee. O(f) bi-approximation for capacitated covering with hard capacities. In *ISAAC'16*, pages 40:1–40:12, 2016. doi:10.4230/LIPIcs.ISAAC.2016.40.

13 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, May 2008. doi:10.1016/j.jcss.2007.06.019.

14 Julián Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. *Algorithmica*, 55(1):227–239, 2009. doi:10.1007/s00453-007-9003-z.

15 B. Saha and S. Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *Proceedings of ICALP'12*, pages 762–773, 2012. doi:10.1007/978-3-642-31594-7_64.

16 Sam Chiu-wai Wong. Tight algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of SODA'17*, pages 2626–2637, 2017. URL: http://dl.acm.org/citation.cfm?id=3039686.3039859.

Hybrid VCSPs with Crisp and Valued Conservative Templates

Rustem Takhanov

Nazarbayev University, Astana, Kazakhstan
rustem.takhanov@nu.edu.kz

Abstract

A constraint satisfaction problem (CSP) is a problem of computing a homomorphism $\mathbf{R} \to \mathbf{\Gamma}$ between two relational structures, e.g. between two directed graphs. Analyzing its complexity has been a very fruitful research direction, especially for *fixed template CSPs* (or, *non-uniform CSPs*), denoted $\mathrm{CSP}(\mathbf{\Gamma})$, in which the right side structure $\mathbf{\Gamma}$ is fixed and the left side structure $\mathbf{R}$ is unconstrained.

Recently, the *hybrid* setting, written $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$, where both sides are restricted simultaneously, attracted some attention. It assumes that $\mathbf{R}$ is taken from a class of relational structures $\mathcal{H}$ (called the *structural restriction*) that additionally is *closed under inverse homomorphisms*. The last property allows to exploit an algebraic machinery that has been developed for fixed template CSPs. The key concept that connects hybrid CSPs with fixed-template CSPs is the so called "lifted language". Namely, this is a constraint language $\mathbf{\Gamma}_{\mathbf{R}}$ that can be constructed from an input $\mathbf{R}$. The tractability of the language $\mathbf{\Gamma}_{\mathbf{R}}$ for any input $\mathbf{R} \in \mathcal{H}$ is a necessary condition for the tractability of the hybrid problem.

In the first part we investigate templates $\mathbf{\Gamma}$ for which the latter condition is not only necessary, but also is sufficient. We call such templates $\mathbf{\Gamma}$ *widely tractable*. For this purpose, we construct from $\mathbf{\Gamma}$ a new finite relational structure $\mathbf{\Gamma}'$ and define a "maximal" structural restriction $\mathcal{H}_0$ as a class of structures homomorphic to $\mathbf{\Gamma}'$. For the so called strongly BJK templates that probably captures all templates, we prove that wide tractability is equivalent to the tractability of $\mathrm{CSP}_{\mathcal{H}_0}(\mathbf{\Gamma})$. Our proof is based on the key observation that $\mathbf{R}$ is homomorphic to $\mathbf{\Gamma}'$ if and only if the core of $\mathbf{\Gamma}_{\mathbf{R}}$ is preserved by a Siggers polymorphism. Analogous result is shown for conservative valued CSPs.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases constraint satisfaction problem, polymorphisms, algebraic approach, lifted language, hybrid CSPs, closed under inverse homomorphisms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.65

1 Introduction

The *constraint satisfaction problems (CSPs)* and the *valued constraint satisfaction problems (VCSPs)* provide a powerful framework for the analysis of a large set of computational problems arising in propositional logic, combinatorial optimization, graph theory, artificial intelligence, scheduling, biology (protein folding), computer vision etc. CSP can be formalized either as a problem of (a) finding an assignment of values to a given set of variables, subject to constraints on the values that can be assigned simultaneously to specified subsets of variables, or as a problem of (b) finding a homomorphism between two finite relational structures A and B (e.g., two oriented graphs). These two formulations are polynomially equivalent under the condition that input constraints in the first case or input relations in the second case are given by lists of their elements. A soft version of CSP, the Valued CSP, generalizes the

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 65; pp. 65:1–65:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

CSP by changing crisp constraints to cost functions applied to tuples of variables. In the VCSP we are asked to find a minimum (or maximum) of a sum of cost functions applied to corresponding variables.

The CSPs have been a very active research field since 70s. One of the topics that revealed the rich logical and algebraic structure of the CSPs was the problem's computational complexity when constraint relations are restricted to a given set of relations or, alternatively, when the second relational structure is some fixed $\mathbf{\Gamma}$. Thus, this problem is parameterized by $\mathbf{\Gamma}$, denoted as CSP($\mathbf{\Gamma}$) and called a fixed template CSP with a template $\mathbf{\Gamma}$ (another name is a non-uniform CSP). E.g., if the domain set is boolean and $\mathbf{\Gamma}$ is a structure with four ternary relations $x \vee y \vee z$, $\overline{x} \vee y \vee z$, $\overline{x} \vee \overline{y} \vee z$, $\overline{x} \vee \overline{y} \vee \overline{z}$, CSP($\mathbf{\Gamma}$) models 3-SAT which is historically one of the first NP-complete problems [8]. At the same time, if we restrict that all relations in $\mathbf{\Gamma}$ are binary, then we obtain tractable 2-SAT. Schaeffer proved [25] that for any template $\mathbf{\Gamma}$ over the boolean set, CSP($\mathbf{\Gamma}$) is either in P or NP-complete. For the case when $\mathbf{\Gamma}$ is a graph (without loops) Hell and Nešetřil [14] proved an analogous statement, by showing that only for bipartite graphs the problem is tractable. Feder and Vardi [11] found that all fixed template CSPs can be expressed as problems in a fragment of SNP, called the Monotone Monadic SNP (MM SNP), and showed that for any problem in MM SNP there is a polynomial-time Turing reduction to a fixed template CSP. Thus, non-uniform CSPs' complexity classification would yield a classification for MM SNP problems. This result placed fixed-template CSPs into a broad logical context which naturally lead to a conjecture that such CSPs are either tractable or NP-hard, the so called dichotomy conjecture.

In [16] Jeavons showed that the complexity of CSP($\mathbf{\Gamma}$) is determined by the polymorphisms of $\mathbf{\Gamma}$. Research in this direction lead to a conjectured description of tractable templates through properties of their polymorphisms. The key formulation was given by Bulatov, Jeavons, and Krokhin [5], with subsequent reformulations of this conjecture by Maroti and McKenzie [23]. Later, it was shown by Siggers [26] that if the Bulatov-Jeavons-Krokhin formulation is true, then for a relational structure to be tractable it is necessary and sufficient that its core is preserved by a single 6-ary polymorphism that satisfies a certain term identity. Further, an arity of a polymorphism in the latter formulation was decreased to 4 [18]. We will use the last fact as a key ingredient for our results. Very recently, several independent proofs of the Bulatov-Jeavons-Krokhin formulation were announced [24, 6, 30]. Since the papers have not yet been thoroughly verified and widely accepted by the CSP community, in this paper we refer to the formulation as a hypothesis.

Related work. A meta-problem of the VCSP topic is to establish the complexity of VCSP given that an input is restricted to an arbitrary subset of all input pairs $(\mathbf{R}, \mathbf{\Gamma})$. A natural approach to this problem is to construct a new structure for any input $(\mathbf{R}, \mathbf{\Gamma})$, $G_{\mathbf{R},\mathbf{\Gamma}}$, and shift the analysis to $G_{\mathbf{R},\mathbf{\Gamma}}$. In case of binary CSPs (i.e. when all relations of an input are binary) it is natural to define $G_{\mathbf{R},\mathbf{\Gamma}}$ as a microstructure graph [17] of an instance $(\mathbf{R}, \mathbf{\Gamma})$. Thereby, a set of inputs, in which certain local substructures in $G_{\mathbf{R},\mathbf{\Gamma}}$ are forbidden, forms a parametrized problem. Cooper and Živný [9] investigated this formulation and found examples of specific forbidden substructures that result in tractable hybrid CSPs. Microstructure graphs also naturally appear in the context of fixed template CSPs. Specifically, if a template $\mathbf{\Gamma}$ with binary relations is such that the arc and path consistency preprocessing of an instance of CSP($\mathbf{\Gamma}$) always results in a perfect microstructure graph, then additionally to satisfying all constraints (by finding a maximum clique) one can also optimize arbitrary sums of unary terms over a set of solutions (by assigning weights to vertices of the microstructure graph). The latter optimization problem is called *the minimum cost homomorphism problem* and all such templates were completely classified in [28].

Recently, a hybrid framework for VCSP has attracted some attention [21], that is when left structures are restricted to some set $\mathcal{H}$ and a right structure $\mathbf{\Gamma}$ is fixed (the corresponding CSP is denoted as $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$) and $\mathcal{H}$ is closed under inverse homomorphisms. The specific feature of this case is that for any input $\mathbf{R} \in \mathcal{H}$ one can construct a new language $\mathbf{\Gamma}_{\mathbf{R}}$, called a lifted language, so that tractability of this language is a necessary condition for the tractability of $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$.

Our results. The first question that we address is a characterization of those templates $\mathbf{\Gamma}$ for which the tractability of $\mathbf{\Gamma}_{\mathbf{R}}$ for any $\mathbf{R} \in \mathcal{H}$ is not only necessary, but also is sufficient for the tractability of $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$. We call $\mathbf{\Gamma}$ that possesses this property for any $\mathcal{H}$ (closed under inverse homomorphisms) *widely tractable*. It turns out that the statement that the core of $\mathbf{\Gamma}_{\mathbf{R}}$ is preserved by a Siggers polymorphism (i.e. satisfies the Bulatov-Jeavons-Krokhin test for non-NP-hardness) is equivalent to the statement that $\mathbf{R}$ is homomorphic to a certain structure $\mathbf{\Gamma}'$ (constructed from $\mathbf{\Gamma}$). Based on this observation we prove that, for a class of templates (that is likely to capture all templates), wide tractability is equivalent to the tractability of $\mathrm{CSP}_{\mathcal{H}_0}(\mathbf{\Gamma})$, where $\mathcal{H}_0$ is equal to a set of structures homomorphic to $\mathbf{\Gamma}'$. Moreover, we prove that $\mathrm{CSP}(\mathbf{\Gamma})$ can be in polynomial-time Turing reduced to $\mathrm{CSP}(\mathbf{\Gamma}')$ and, therefore, $\mathbf{\Gamma}'$ is at least as hard as $\mathbf{\Gamma}$. We develop an analogous theory for conservative valued CSPs.

2 Preliminaries

Throughout the paper it is assumed that $P \neq NP$. A problem is called *tractable* if it can be solved in polynomial time. Let $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ denote the set of rational numbers with (positive) infinity and $[k] = \{1, ..., k\}$. Also, D and V are finite sets, D^V is a set of mappings from V to D. We denote the tuples in lowercase boldface such as $\mathbf{a} = (a_1, \dots, a_k)$. Also for mappings $h\colon A \to B$ and tuples $\mathbf{a} = (a_1, \dots, a_k)$, where $a_j \in A$ for $j = 1, \dots, k$, we will write $\mathbf{b} = (h(a_1), \dots, h(a_k))$ simply as $\mathbf{b} = h(\mathbf{a})$. Relational structures are denoted in uppercase boldface as $\mathbf{R} = (R, r_1, \dots, r_k)$. Finally let $\mathrm{ar}(\varrho)$, $\mathrm{ar}(\mathbf{a})$, and $\mathrm{ar}(f)$ stand for the arity of a relation ϱ, the size of a tuple $\mathbf{a}$, and the arity of a function f, respectively.

2.1 Fixed template VCSPs

Let us formulate the general CSP as a homomorphism problem.

▶ **Definition 1.** Let $\mathbf{R} = (R, r_1, \dots, r_k)$ and $\mathbf{R}' = (R', r'_1, \dots, r'_k)$ be relational structures with a common signature (that is $\mathrm{ar}(r_i) = \mathrm{ar}(r'_i)$ for every $i = 1, \dots, k$). A mapping $h\colon R \to R'$ is called a *homomorphism* from $\mathbf{R}$ to $\mathbf{R}'$ if for every $i = 1, \dots, k$ and for any $(x_1, \dots, x_{\mathrm{ar}(r_i)}) \in r_i$ we have that $\big((h(x_1), \dots, h(x_{\mathrm{ar}(r'_i)})\big) \in r'_i$. In that case, we write $\mathbf{R} \xrightarrow{h} \mathbf{R}'$ or sometimes just $\mathbf{R} \to \mathbf{R}'$.

▶ **Definition 2.** **The general CSP** is the following problem. Given a pair of relational structures with a common signature $\mathbf{R} = (V, r_1, \dots, r_k)$ and $\mathbf{\Gamma} = (D, \varrho_1, \dots, \varrho_k)$, the question is whether there is a homomorphism $h : \mathbf{R} \to \mathbf{\Gamma}$. The second structure $\mathbf{\Gamma}$ is called *a template*.

▶ **Definition 3.** Let D be a finite set and $\mathbf{\Gamma}$ be a finite relational structure over D. Then the **fixed template CSP** for template $\mathbf{\Gamma}$, denoted $\mathrm{CSP}(\mathbf{\Gamma})$, is defined as follows: given a relational structure $\mathbf{R} = (V, r_1, \dots, r_k)$ of the same signature as $\mathbf{\Gamma}$, the question is whether there is a homorphism $h : \mathbf{R} \to \mathbf{\Gamma}$.

A more general framework operates with *cost functions* $f : D^n \to \overline{\mathbb{Q}}$ instead of relations $\varrho \subseteq D^n$.

▶ **Definition 4.** Let us denote the set of all functions $f : D^n \to \overline{\mathbb{Q}}$ by $\mathbf{\Phi}_D^{(n)}$ and let $\mathbf{\Phi}_D = \bigcup_{n\geq 1} \mathbf{\Phi}_D^{(n)}$. We call the functions in $\mathbf{\Phi}_D$ *cost functions* over D. For every cost function $f \in \mathbf{\Phi}_D^{(n)}$, let $\operatorname{dom} f = \{x \mid f(x) < \infty\}$.

▶ **Definition 5.** An instance of the **valued constraint satisfaction problem** (VCSP) is a triple $(\mathbf{R}, \mathbf{\Gamma}, \{w_i(\mathbf{v})\}_{i\in[k],\mathbf{v}\in r_i})$ where $\mathbf{R} = (V, r_1, \ldots, r_k)$ is a relational structure, $\mathbf{\Gamma} = (D, f_1, \ldots, f_k)$ is a tuple where D is finite and $f_i \in \mathbf{\Phi}_D^{(\mathrm{ar}(r_i))}$, $\{w_i(\mathbf{v})\}_{i\in[k],\mathbf{v}\in r_i}$ are positive rationals, and the goal is to find an *assignment* $h \in D^V$ that minimizes a function from D^V to $\overline{\mathbb{Q}}$ given by

$$f_{\mathcal{I}}(h) = \sum_{i=1}^{k} \sum_{\mathbf{v}\in r_i} w_i(\mathbf{v}) f_i(h(\mathbf{v})), \tag{1}$$

A tuple $\mathbf{\Gamma} = (D, f_1, \ldots, f_k)$ is called a *valued template.*

▶ **Definition 6.** We will denote by VCSP($\mathbf{\Gamma}$) a class of all VCSP instances in which the valued template is $\mathbf{\Gamma}$.

For such $\mathbf{\Gamma}$ we will denote by Γ (without boldface) the set of cost functions $\{f_1, \ldots, f_k\}$. A set Γ is called a constraint language. The complexity of VCSP($\mathbf{\Gamma}$) does not depend on the order of cost functions, therefore, we will use VCSP(Γ) and VCSP($\mathbf{\Gamma}$) interchangeably.

This framework captures many specific well-known problems, including k-Sat, Graph k-Colouring, Minimum Cost Homomorphism Problem and others (see [15]).

A function $f \in \mathbf{\Phi}_D^{(n)}$ that takes values in $\{0, \infty\}$ is called *crisp.* We will often view it as a relation in D^n, and vice versa (this should be clear from the context). If a language Γ is crisp (i.e. it contains only crisp functions) then VCSP(Γ) is a search problem corresponding to CSP(Γ).

▶ Remark. Note that we formulated CSP as a decision problem, whereas VCSP as a search optimizational problem. This convention is followed throughout the text and further it becomes more important because decision and search problems are not computationally equivalent for hybrid CSPs (see after definition 20).

▶ **Definition 7.** A constraint language Γ (or, a template $\mathbf{\Gamma}$) is said to be tractable, if VCSP(Γ_0) is tractable for each finite $\Gamma_0 \subseteq \Gamma$. Also, Γ (or, $\mathbf{\Gamma}$) is NP-hard if there is a finite $\Gamma_0 \subseteq \Gamma$ such that VCSP(Γ_0) is NP-hard.

An important problem in the CSP research is to characterize all tractable languages.

2.2 Polymorphisms and fractional polymorphisms

Let $\mathcal{O}_D^{(m)}$ denote a set of all operations $g : D^m \to D$ and let $\mathcal{O}_D = \bigcup_{m\geq 1} \mathcal{O}_D^{(m)}$.

Any language Γ over a domain D can be associated with a set of operations on D, known as the polymorphisms of Γ, defined as follows.

▶ **Definition 8.** An operation $g \in \mathcal{O}_D^{(m)}$ is a *polymorphism* of a relation $\rho \subseteq D^n$ (or, g *preserves* ρ) if, for any $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \rho$, we have that $g(\mathbf{x}^1, \ldots, \mathbf{x}^m) \in \rho$ where g is applied component-wise. For any crisp constraint language Γ over a set D, we denote by Pol(Γ) a set of all operations on D which are polymorphisms of every $\rho \in \Gamma$.

Polymorphisms play a key role in the algebraic approach to the CSP, but, for VCSPs, more general constructs are necessary, which we now define.

▶ **Definition 9.** An m-ary *fractional operation* ω on D is a probability distribution on $\mathcal{O}_D^{(m)}$. *The support of* ω is defined as $\mathrm{supp}(\omega) = \{g \in \mathcal{O}_D^{(m)} \mid \omega(g) > 0\}$.

▶ **Definition 10.** An m-ary fractional operation ω on D is said to be a *fractional polymorphism* of a cost function $f \in \mathbf{\Phi}_D$ if, for any $\mathbf{x}^1, \dots, \mathbf{x}^m \in \mathrm{dom}\, f$, we have

$$\sum_{g \in \mathrm{supp}(\omega)} \omega(g) f(g(\mathbf{x}^1, \dots, \mathbf{x}^m)) \leq \frac{1}{m}(f(\mathbf{x}^1) + \dots + f(\mathbf{x}^m)). \tag{2}$$

For a constraint language Γ, $\mathrm{fPol}(\Gamma)$ will denote a set of all fractional operations that are fractional polymorphisms of each function in Γ.

We will also use symbols $\mathrm{Pol}(\mathbf{\Gamma})$, $\mathrm{fPol}(\mathbf{\Gamma})$ meaning $\mathrm{Pol}(\Gamma)$, $\mathrm{fPol}(\Gamma)$ respectively.

2.3 Algebraic dichotomy conjecture

An algebraic characterization for tractable templates was first conjectured by Bulatov, Krokhin and Jeavons [5], and a number of equivalent formulations were later given in [23, 1, 26, 18]. We will use the formulation from [18] that followed a discovery by M. Siggers [26]; it is crucial for our purposes that in the next definition an operation has a fixed arity (namely, 4) and, therefore, there is only a finite number of them on a finite domain D.

▶ **Definition 11.** An operation $s\colon D^4 \to D$ is called **a Siggers operation on** $D' \subseteq D$ if $s(x, y, z, t) \in D'$ whenever $x, y, z, t \in D'$ and for each $x, y, z \in D'$ we have:

$$s(x, y, x, z) = s(y, x, z, y)$$
$$s(x, x, x, x) = x$$

▶ **Definition 12.** Let g be a unary and s be a 4-ary operations on D and $g(D) = \{g(x) | x \in D\}$. A pair (g, s) is called *a Siggers pair on* D if s is a Siggers operation on $g(D)$. A crisp constraint language Γ is said to admit a Siggers pair (g, s) if g and s are polymorphisms of Γ.

▶ **Theorem 13** ([18]). *A crisp constraint language* Γ *that does not admit a Siggers pair is NP-Hard.*

▶ **Definition 14.** A crisp language Γ is called a *BJK* language if it satisfies one of the following:
- $CSP(\Gamma)$ is tractable
- Γ does not admit a Siggers pair.

Algebraic dichotomy conjecture: Every crisp language Γ is a BJK language.

This theorem first has been verified for domains of size 2 [25], 3 [3], or for languages containing all unary relations on D [4]. It has also been shown that it is equivalent to its restriction for directed graphs (that is when Γ contains a single binary relation ϱ) [7]. Just recently, a number of authors [24, 6, 30] independently claimed the proof of the conjecture.

3 Hybrid VCSP setting

▶ **Definition 15.** Let us call a family $\mathcal{H}$ of relational structures with a common signature a **structural restriction**.

▶ **Definition 16** (Hybrid CSP). Let D be a finite domain, $\mathbf{\Gamma}$ a template over D, and $\mathcal{H}$ a structural restriction of the same signature as $\mathbf{\Gamma}$. We define $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ as the following problem: given a relational structure $\mathbf{R} \in \mathcal{H}$ as input, decide whether there is a homomorphism $h : \mathbf{R} \to \mathbf{\Gamma}$.

▶ **Definition 17** (Hybrid VCSP). Let D be a finite domain, $\mathbf{\Gamma} = (D, f_1, \dots, f_k)$ a valued template over D, and $\mathcal{H}$ a structural restriction of the same signature as $\mathbf{\Gamma}$. We define $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$ as a class of instances of the following form.

An instance is a function from D^V to $\overline{\mathbb{Q}}$ given by

$$f_{\mathcal{I}}(h) = \sum_{i=1}^{k} \sum_{\mathbf{v} \in r_i} w_i(\mathbf{v}) f_i(h(\mathbf{v})), \tag{3}$$

where $\mathbf{R} = (V, r_1, \dots, r_k) \in \mathcal{H}$ is a relational structure, $\{w_i(\mathbf{v})\}_{i \in [k], \mathbf{v} \in r_i}$ are positive rationals. The goal is to find an *assignment* $h \in D^V$ that minimizes $f_{\mathcal{I}}$.

For certain classes of structural restrictions the tractability/intractability can be explained by algebraic means, and of special interest is the case when $\mathcal{H}$ is *up-closed.*

▶ **Definition 18.** A family of relational structures $\mathcal{H}$ is called **closed under inverse homomorphisms** (or **up-closed** for short) if whenever $\mathbf{R}' \to \mathbf{R}$ and $\mathbf{R} \in \mathcal{H}$, then also $\mathbf{R}' \in \mathcal{H}$.

Examples of hybrid CSPs with up-closed structural restrictions include such studied problems as a digraph H-coloring for an acyclic input digraph [27] or for an input digraph with odd girth at least k [21], renamable Horn Boolean CSPs [12] etc. The key tool in their analysis is a construction of the so called lifted language that appeared first in [21]. In this construction, given arbitrary $\mathbf{R} \in \mathcal{H}$ one constructs a language $\Gamma_{\mathbf{R}}$ over a finite domain, such that for tractability of $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$, the tractability of $\mathrm{VCSP}(\Gamma_{\mathbf{R}})$ is *necessary.*

Let us give a detailed description of $\Gamma_{\mathbf{R}}$. Given $\mathbf{R} = (V, r_1, \dots, r_k)$ and $\mathbf{\Gamma} = (D, f_1, ..., f_k)$ we define $D_{\mathbf{R}} = V \times D$ and $D_v = \{(v,a) | a \in D\}, v \in V$.

For tuples $\mathbf{a} = (a_1, \dots, a_p) \in D^p$ and $\mathbf{v} = (v_1, \dots, v_p) \in V^p$ denote $d(\mathbf{v}, \mathbf{a}) = ((v_1, a_1), ..., (v_p, a_p))$.

Now for a cost function $f \in \mathbf{\Phi}_D$ and $\mathbf{v} \in V^{\mathrm{ar}(f)}$ we will define a cost function on $D_{\mathbf{R}}$ of the same arity as f via

$$f^{\mathbf{v}}(\mathbf{x}) = \begin{cases} f(\mathbf{y}) & \text{if } \mathbf{x} = d(\mathbf{v}, \mathbf{y}) \text{ for some } \mathbf{y} \in D^{\mathrm{ar}(f)} \\ \infty & \text{otherwise} \end{cases} \qquad \forall \mathbf{x} \in D_{\mathbf{R}}^{\mathrm{ar}(f)} \tag{4}$$

Finally, we construct the sought language $\Gamma_{\mathbf{R}}$ on domain $D_{\mathbf{R}}$ as follows:

$$\Gamma_{\mathbf{R}} = \{f_i^{\mathbf{v}} \ : \ i \in [k], \mathbf{v} \in r_i\} \cup \{D_v \ : \ v \in V\}$$

where relation $D_v \subseteq D_{\mathbf{R}}$ is treated as a unary function $D_v : D_{\mathbf{R}} \to \{0, \infty\}$.

After ordering of its relations $\Gamma_{\mathbf{R}}$ becomes a template $\mathbf{\Gamma_R}$. The following is true [21]:

▶ **Theorem 19.** *Suppose that $\mathcal{H}$ is up-closed, $\mathbf{R} \in \mathcal{H}$ and $\mathbf{\Gamma}$ is a (valued) template. Then there is a polynomial-time reduction from (V)CSP($\mathbf{\Gamma_R}$) to (V)CSP$_{\mathcal{H}}(\mathbf{\Gamma})$. Consequently,*
(a) if (V)CSP$_{\mathcal{H}}(\mathbf{\Gamma})$ is tractable then so is (V)CSP($\mathbf{\Gamma_R}$);
(b) if (V)CSP($\mathbf{\Gamma_R}$) is NP-hard then so is (V)CSP$_{\mathcal{H}}(\mathbf{\Gamma})$.

Let us give a proof of the latter theorem that slightly differs from the original one. For this purpose we will need a special case of hybrid VCSP, called *the VCSP with input prototype.* Given a finite relational structure $\mathbf{R}$, denote $\mathbf{Up}\,(\mathbf{R}) = \{\mathbf{I} | \mathbf{I} \to \mathbf{R}\}$.

▶ **Definition 20.** For a given valued template $\mathbf{\Gamma}$ and a relational structure $\mathbf{R}$ a problem $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$ where $\mathcal{H} = \mathbf{Up}(\mathbf{R})$ is called **the VCSP with input prototype R** and is denoted as $\mathrm{VCSP}_{\mathbf{R}}(\mathbf{\Gamma})$. If $\mathbf{\Gamma}$ is crisp, then the decision version of $\mathrm{VCSP}_{\mathbf{R}}(\mathbf{\Gamma})$ is denoted as $\mathrm{CSP}_{\mathbf{R}}(\mathbf{\Gamma})$.

It is easy to see that $\mathcal{H} = \mathbf{Up}(\mathbf{R})$ is up-closed. Note that an input of $(\mathrm{V})\mathrm{CSP}_{\mathbf{R}}(\mathbf{\Gamma})$ is a relational structure $\mathbf{I}$ that is homomorphic to $\mathbf{R}$ but this homomorphism itself is not a part of the input. If we also assume that together with a structure $\mathbf{I}$ we are given a homomorphism $h : \mathbf{I} \to \mathbf{R}$, then the latter problem is denoted as $(\mathrm{V})\mathrm{CSP}^{+}_{\mathbf{R}}(\mathbf{\Gamma})$.

▶ Remark. Note that the complexities of $\mathrm{VCSP}_{\mathbf{R}}(\mathbf{\Gamma})$ and $\mathrm{VCSP}^{+}_{\mathbf{R}}(\mathbf{\Gamma})$ can be sharply different. For example, consider $\mathbf{\Gamma} = ([4]; \mathbf{neq}_4)$ and $\mathbf{R} = ([3]; \mathbf{neq}_3)$ where $\mathbf{neq}_k = \{(i,j)|i,j \in [k], i \neq j\}$. While $\mathrm{VCSP}_{\mathbf{R}}(\mathbf{\Gamma})$, a problem of 4-coloring of a 3-colorable graph, is known to be NP-hard [19], $\mathrm{VCSP}^{+}_{\mathbf{R}}(\mathbf{\Gamma})$ is a trivial one. This example also demonstrates the distinction between decision and search in the hybrid framework: the decision problem $\mathrm{CSP}_{\mathbf{R}}(\mathbf{\Gamma})$ is also trivial, whereas its search version is NP-hard.

▶ **Lemma 21.** $(\mathrm{V})\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{R}})$ *is polynomially equivalent to* $(\mathrm{V})\mathrm{CSP}^{+}_{\mathbf{R}}(\mathbf{\Gamma})$

Theorem 19 (a). Since $\mathcal{H}$ is up-closed, then for any $\mathbf{R} \in \mathcal{H}$, $\{\mathbf{I}|\mathbf{I} \to \mathbf{R}\} \subseteq \mathcal{H}$. I.e. a problem $\mathrm{VCSP}_{\mathbf{R}}(\mathbf{\Gamma})$ is a restriction of $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$ to certain inputs. Therefore, $\mathrm{VCSP}^{+}_{\mathbf{R}}(\mathbf{\Gamma})$ is polynomially reducible to $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$. Using the previous lemma, we conclude that for the tractability of $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$ it is necessary that $\mathrm{VCSP}^{+}_{\mathbf{R}}(\mathbf{\Gamma})$ and $\mathrm{VCSP}(\mathbf{\Gamma}_{\mathbf{R}})$ are tractable. Part (b) can be proved analogously. ◀

4 Wide tractability of a crisp language

Throughout this section we will assume that $\mathbf{\Gamma}$ is crisp.

4.1 Widely tractable languages

For up-closed structural restrictions $\mathcal{H}$, the construction of a lifted language gives us the necessary conditions for the tractability of $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ (Theorem 19 (a)). Let us now define *widely tractable templates* $\mathbf{\Gamma}$ as those for which the necessary conditions for the tractability of $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ are, in fact, sufficient:

▶ **Definition 22.** A template $\mathbf{\Gamma}$ is called **widely tractable** if for any up-closed $\mathcal{H}$, $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ is tractable if and only if $\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{R}})$ is tractable for any $\mathbf{R} \in \mathcal{H}$.

The concept of wide tractability is important in the hybrid CSPs setting due to the following theorem:

▶ **Theorem 23.** *If a template* $\mathbf{\Gamma}$ *is widely tractable, then there is an up-closed* $\mathcal{H}^{\mathbf{\Gamma}}$ *such that for any up-closed* $\mathcal{H}$, $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ *is tractable if and only if* $\mathcal{H} \subseteq \mathcal{H}^{\mathbf{\Gamma}}$.

Proof. Let us define

$$\mathcal{H}^{\mathbf{\Gamma}} = \{\mathbf{R}|\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{R}}) \text{ is tractable}\} \tag{5}$$

It is easy to see that $\mathcal{H}^{\mathbf{\Gamma}}$ is up-closed itself. By definition, $\mathcal{H}^{\mathbf{\Gamma}}$ contains only such $\mathbf{R}$ for which $\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{R}})$ is tractable, and this together with wide tractability of $\mathbf{\Gamma}$, implies that $\mathrm{CSP}_{\mathcal{H}^{\mathbf{\Gamma}}}(\mathbf{\Gamma})$ is tractable.

Suppose that for some up-closed $\mathcal{H}$, $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ is tractable. From the wide tractability of $\mathbf{\Gamma}$ we obtain that it is equivalent to stating that $\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{R}})$ is tractable for any $\mathbf{R} \in \mathcal{H}$. But the last is equivalent to $\mathcal{H} \subseteq \mathcal{H}^{\mathbf{\Gamma}}$. ◀

4.2 Wide tractability in case of strongly BJK languages

In this section we will give necessary and sufficient conditions of wide tractability in a very important case of crisp languages, namely, *strongly BJK* languages.

▶ **Definition 24.** A crisp language $\mathbf{\Gamma}$ is called strongly BJK language if for any $\mathbf{R}$ the lifted $\mathbf{\Gamma_R}$ is BJK.

▶ Remark. As we have already noted it is likely that this class includes all crisp languages [24, 6, 30].

Before introducing the main theorem of this section, let us describe one construction. Let ρ be some m-ary relation over a domain D. It induces a new relation ρ' over a set of Siggers pairs on a set D, denoted D', by the following rule: a tuple of Siggers pairs $\big((g_1, s_1), \cdots, (g_m, s_m)\big) \in \rho'$ if and only if for any $(x_1, ..., x_m) \in \rho$ we have that $(g_1(x_1), ..., g_m(x_m)) \in \rho$ and for any tuples $(a_1, ..., a_m)$, $(b_1, ..., b_m)$, $(c_1, ..., c_m)$, $(d_1, ..., d_m)$ from ρ we have that $\big(s_1(a_1, b_1, c_1, d_1), \ ... \ , s_m(a_m, b_m, c_m, d_m)\big) \in \rho$. Note that elements of D' are Siggers pairs, but not necessarily polymorphisms of ρ.

Given a relational structure $\mathbf{\Gamma} = (D, \rho_1, ..., \rho_s)$, we define $\mathbf{\Gamma}' = (D', \rho_1', ..., \rho_s')$.

▶ **Theorem 25.** *Let $\mathbf{\Gamma}$ be a strongly BJK language. Then $\mathbf{\Gamma}$ is widely tractable if and only if $\mathrm{CSP}_{\mathbf{\Gamma}'}(\mathbf{\Gamma})$ is tractable.*

A proof of theorem 25 is mainly based on the following lemma:

▶ **Lemma 26.** *For an arbitrary $\mathbf{R}$, $\mathbf{\Gamma_R}$ admits a Siggers pair if and only if there is a homomorphism $h : \mathbf{R} \to \mathbf{\Gamma}'$.*

▶ Remark. If $\mathbf{\Gamma}' \to \mathbf{\Gamma}$ then $\mathrm{CSP}_{\mathbf{\Gamma}'}(\mathbf{\Gamma})$ is a trivial problem and theorem 25 gives us that $\mathbf{\Gamma}$ is a widely tractable template. Such templates are quite common. E.g. our computational experiment showed (see section 6) that if $D = \{0, 1\}$ and $\rho \subseteq \{0, 1\}^3$ is such that $\Gamma = \{\rho\}$ is NP-hard, then $\mathbf{\Gamma}' \to \mathbf{\Gamma}$. Example of a widely tractable and NP-hard $\mathbf{\Gamma}$ for which $\mathbf{\Gamma}' \not\to \mathbf{\Gamma}$ will be given in the next section (example 29).

4.3 Relationship between $\mathbf{\Gamma}$ and $\mathbf{\Gamma}'$

The binary relation $\to$ is transitive, reflexive, but not antisymmetric. It also induces the equivalence relation $\sim$ on a set of all finite structures:

$$\mathbf{R}_1 \sim \mathbf{R}_2 \Leftrightarrow \mathbf{R}_1 \to \mathbf{R}_2, \mathbf{R}_2 \to \mathbf{R}_1$$

▶ **Theorem 27.** *For any $\mathbf{\Gamma}$, $\mathbf{\Gamma} \to \mathbf{\Gamma}'$.*

Thus, we can view $\mathrm{CSP}(\mathbf{\Gamma}')$ as a relaxation of $\mathrm{CSP}(\mathbf{\Gamma})$. Moreover, theorem 27 has the following interesting consequence.

▶ **Theorem 28.** *If $\mathbf{\Gamma}$ is strongly BJK, then there is a polynomial-time Turing reduction from $\mathrm{CSP}(\mathbf{\Gamma})$ to $\mathrm{CSP}(\mathbf{\Gamma}')$*

If $\mathbf{\Gamma}$ is tractable, then $\mathbf{\Gamma}'$ is preserved by a nullary constant operation $o = (g, s)$, where $(g, s) \in D'$ is a Siggers pair that is admitted by $\mathbf{\Gamma}$. I.e., $\mathbf{Up}(\mathbf{\Gamma}')$ is a set of all finite structures with the same vocabulary as $\mathbf{\Gamma}$. We can take any tractable $\mathbf{\Gamma}$ that is not constant-preserving (e.g. $\mathbf{\Gamma} = ([3]; \mathbf{neq}_3)$) as an example of a template for which $\mathbf{\Gamma} \not\sim \mathbf{\Gamma}'$, i.e. $\mathbf{\Gamma}' \not\to \mathbf{\Gamma}$.

The following example demonstrates an NP-hard $\mathbf{\Gamma}$ for which $\mathbf{\Gamma} \not\sim \mathbf{\Gamma}'$.

▶ **Example 29.** Define $\mathbf{\Gamma} = (\{0,1\}; \{0\}, \{1\}, \rho)$, where $\rho = \{0,1\}^3 \setminus \{(0,1,0),(1,0,1)\}$. A fixed-template CSP with this $\mathbf{\Gamma}$ is called *the boolean betweennes*, and it is NP-hard because $\mathbf{\Gamma}$ does not fall into any of Schaefer's classes [25].

The boolean betweennes can be popularly reformulated in the following way. Suppose that we have a number of n towns $v_1, ..., v_n$ and a system of roads (each consisting of 3 consecutive towns) $(v_{\alpha_1}, v_{\alpha_2}, v_{\alpha_3}), ..., (v_{\omega_1}, v_{\omega_2}, v_{\omega_3})$. Our goal is to divide those towns between 2 states (assign 0 or 1 to n variables) in such a way that unary constraints are satisfied, i.e. certain towns should be given to prespecified states, and every road should not cross administrative barriers twice.

Let $\mathbf{\Gamma}_\alpha = (\{0,1,\alpha\}; \{0,\alpha\}, \{1,\alpha\}, \rho_\alpha)$, where $\rho_\alpha = \rho \cup \{(1,1,\alpha),(\alpha,1,1),(0,0,\alpha),(\alpha,0,0),(0,\alpha,1),(1,\alpha,0)\}$. A symbol α can be interpreted as a "dual attachment" status that can be given to towns, for which we can freely change α-status to both 0 and 1 without violating ternary constraints.

It is easy to see that $\mathbf{\Gamma}_\alpha \not\to \mathbf{\Gamma}$ (image of α cannot be both 0 and 1). If we prove that $\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{\Gamma}_\alpha})$ is tractable (and, therefore, $\mathbf{\Gamma}_{\mathbf{\Gamma}_\alpha}$ admits a Siggers pair), this will lead to a conclusion that $\mathbf{\Gamma}_\alpha \to \mathbf{\Gamma}'$ by lemma 26, and consequently, $\mathbf{\Gamma}' \not\to \mathbf{\Gamma}$.

According to lemma 21, $\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{\Gamma}_\alpha})$ is equivalent to a problem of deciding whether there is a homomorphism $h : \mathbf{R} \to \mathbf{\Gamma}$ for a relational structure $\mathbf{R} = (V, \Omega_0, \Omega_1, \Omega)$ and a homomorphism $g : \mathbf{R} \to \mathbf{\Gamma}_\alpha$ given as inputs. If $\Omega_0 \cap \Omega_1 \neq \emptyset$ we claim the nonexistence of h. Otherwise, h is defined in the following way: $h(x) = g(x)$, if $g(x) \neq \alpha$; $h(x) = 0$, if $x \in \Omega_0$ and $g(x) = \alpha$; $h(x) = 1$, if $x \in \Omega_1$ and $g(x) = \alpha$; and $h(x) = 0$, if otherwise. It can be checked that this algorithm solves $\mathrm{CSP}^+_{\mathbf{\Gamma}_\alpha}(\mathbf{\Gamma})$.

Our computational experiment showed (see section 6) that, in fact, $\mathbf{\Gamma}' \sim \mathbf{\Gamma}_\alpha$. It is easy to see that in the latter algorithm for $\mathrm{CSP}(\mathbf{\Gamma}_{\mathbf{\Gamma}_\alpha})$ we used a homomorphism $g : \mathbf{R} \to \mathbf{\Gamma}_\alpha$ only at the stage of the construction of h, i.e. we did not need it at the decision stage. The latter means that $\mathrm{CSP}_{\mathbf{\Gamma}_\alpha}(\mathbf{\Gamma})$ as a decision problem is also tractable and from theorem 25 we obtain that $\mathbf{\Gamma}$ is widely tractable (under condition that it is strongly BJK).

Theorem 28 gives us the idea that we can reduce $\mathrm{CSP}(\mathbf{\Gamma})$ to $\mathrm{CSP}(\mathbf{\Gamma}')$, $\mathrm{CSP}(\mathbf{\Gamma}')$ to $\mathrm{CSP}(\mathbf{\Gamma}'')$ etc. It turns out that this sequence of reductions collapses very soon:

▶ **Theorem 30.** *If $\mathbf{\Gamma}, \mathbf{\Gamma}'$ are both strongly BJK, then $\mathbf{\Gamma}' \sim \mathbf{\Gamma}''$.*

5 Valued templates: conservative case

So far, the most applicable class of fixed-template valued VCSPs was *the submodular function minimization* problems [22]. Also, *minimum cost homomorphism problems (MinHom)* appeared in such different contexts as Defense Logistics [13] and Computer Vision [10]. These two examples make the framework of *conservative valued CSPs* of special interest, since it includes both MinHom and submodular function minimization. The structure of tractable conservative languages is very clearly understood both in crisp [4] and valued cases [29]. Let us now give the definition.

▶ **Definition 31.** A valued constraint language Γ is called *conservative* if it contains $\mathbf{Un}_D$, where $\mathbf{Un}_D$ is a set of all unary $\{0,1\}$-valued cost functions over D.

In the hybrid VCSPs setting, if the right structure $\mathbf{\Gamma}$ is conservative, we have to make a certain supplementary assumption on structural restrictions, so that we do not loose the desirable property that optimized function can have an arbitrary unary part.

▶ **Definition 32.** We say that a relational structure $\mathcal{H}$ **does not restrict unaries** if for each $\mathbf{R} \in \mathcal{H}$ of the form $\mathbf{R} = (V, r_1, \dots, r_{i-1}, r_i, r_{i+1}, \dots, r_k)$ with $\mathrm{ar}(r_i) = 1$ and for each unary relation $r'_i \subseteq V$, we have $\mathbf{R}' \in \mathcal{H}$, where $\mathbf{R}' = (V, r_1, \dots, r_{i-1}, r'_i, r_{i+1}, \dots, r_k)$.

A generalization of the wide tractability for conservative languages will be the following definition.

▶ **Definition 33.** A valued conservative language $\mathbf{\Gamma}$ is called **widely c-tractable** if for any up-closed $\mathcal{H}$ that does not restrict unaries, $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$ is tractable if and only if $\mathrm{VCSP}(\mathbf{\Gamma_R})$ is tractable for any $\mathbf{R} \in \mathcal{H}$.

▶ **Theorem 34.** *Any conservative valued language is widely c-tractable.*

An analog of theorem 23 is the following statement.

▶ **Theorem 35.** *For any conservative valued language $\mathbf{\Gamma}$ there is an up-closed $\mathcal{H}_c^{\mathbf{\Gamma}}$ that does not restrict unaries and such that for any up-closed $\mathcal{H}$ that does not restrict unaries,* $\mathrm{VCSP}_{\mathcal{H}}(\mathbf{\Gamma})$ *is tractable if and only if $\mathcal{H} \subseteq \mathcal{H}_c^{\mathbf{\Gamma}}$.*

Our next goal will be to prove that $\mathcal{H}_c^{\mathbf{\Gamma}} = \mathbf{Up}(\mathbf{\Gamma}'_c)$ for a certain template $\mathbf{\Gamma}'_c$. If in a case of $\mathrm{CSP}_{\mathcal{H}}(\mathbf{\Gamma})$ we used a description of tractable templates in terms of polymorphisms, in the current case we will need a description via fractional polymorphisms.

▶ **Definition 36.** Let $(\sqcup, \sqcap)$ be a pair of binary operations and (Mj_1, Mj_2, Mn_3) be a triple of ternary operations defined on a domain D, and $M \subseteq \{\{a,b\}\,|a,b \in D, a \neq b\}$.

The pair $(\sqcup, \sqcap)$, is a **symmetric tournament polymorphism (STP)** on M if $\forall x, y$, $\{x \sqcup y, x \sqcap y\} = \{x, y\}$ and for any $\{a,b\} \in M$, $a \sqcup b = b \sqcup a$, $a \sqcap b = b \sqcap a$.

The triple (Mj_1, Mj_2, Mn_3) is **an MJN** on M if $\forall x, y, z, \{Mj_1(x,y,z), Mj_2(x,y,z), Mn_3(x,y,z)\} = \{x,y,z\}$ and for each triple $(a,b,c) \in D^3$ with $\{a,b,c\} = \{x,y\} \in M$ operations $Mj_1(a,b,c)$, $Mj_2(a,b,c)$ return the unique majority element among a, b, c (that occurs twice) and $Mn_3(a,b,c)$ returns the remaining minority element.

The following theorem was established in [20].

▶ **Theorem 37.** *A conservative valued language $\mathbf{\Gamma}$ is tractable if and only if there is a symmetric tournament polymorphism $(\sqcup, \sqcap)$ on M, an MJN (Mj_1, Mj_2, Mn_3) on $\overline{M} = \{\{a,b\}\,|a,b \in D, a \neq b\} \setminus M$, such that $(\sqcup, \sqcap)$, $(Mj_1, Mj_2, Mn_3) \in \mathrm{fPol}(\Gamma)$.*

Given $\mathbf{\Gamma} = (D, f_1, ..., f_s)$, let us construct a relational structure $\mathbf{\Gamma}'_c = (D'_c, f'_1, ..., f'_s)$. Its domain, D'_c, is defined as a set of all triples $\big(M, (\sqcup, \sqcap), (Mj_1, Mj_2, Mn_3)\big)$ such that $(\sqcup, \sqcap)$ is a symmetric tournament polymorphism on M and (Mj_1, Mj_2, Mn_3) is an MJN on $\overline{M}$. All f'_i will be relations, i.e. crisp cost functions.

A tuple

$$\big(\big(M^1, (\sqcup^1, \sqcap^1), (Mj_1^1, Mj_2^1, Mn_3^1)\big), \cdots, \big(M^p, (\sqcup^p, \sqcap^p), (Mj_1^p, Mj_2^p, Mn_3^p)\big)\big)$$

is in f'_i if and only if $\big(\sqcup^1, \cdots, \sqcup^p\big)$, $\big(\sqcap^1, \cdots, \sqcap^p\big)$ and $\big(Mj_1^1, \cdots, Mj_1^p\big)$, $\big(Mj_2^1, \cdots, Mj_2^p\big)$, $\big(Mn_3^1, \cdots, Mn_3^p\big)$ are component-wise fractional polymorphisms of f_i, i.e. for any $\mathbf{x} = (x_1, \cdots, x_p)$, $\mathbf{y} = (y_1, \cdots, y_p)$, $\mathbf{z} = (z_1, \cdots, z_p)$ the following inequalities are satisfied:

$$f_i(\mathbf{x} \sqcup \mathbf{y}) + f_i(\mathbf{x} \sqcap \mathbf{y}) \leq f_i(\mathbf{x}) + f_i(\mathbf{y})$$

$$f_i(Mj_1(\mathbf{x}, \mathbf{y}, \mathbf{z})) + f_i(Mj_2(\mathbf{x}, \mathbf{y}, \mathbf{z})) + f_i(Mn_3(\mathbf{x}, \mathbf{y}, \mathbf{z})) \leq$$
$$f_i(\mathbf{x}) + f_i(\mathbf{y}) + f_i(\mathbf{z})$$

where $\mathbf{x} \sqcup \mathbf{y} = \left(x_1 \sqcup^1 y_1, ..., x_p \sqcup^p y_p\right)$ and $\mathbf{x} \sqcap \mathbf{y} = \left(x_1 \sqcap^1 y_1, ..., x_p \sqcap^p y_p\right)$. Analogously, $M(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \left(M^1(x_1, y_1, z_1), ..., M^p(x_p, y_p, z_p)\right)$, where instead of M we can paste Mj_1, Mj_2, or Mn_3.

The structure $\mathbf{\Gamma}'_c$ is an analog of $\mathbf{\Gamma}'$. Its domain consists of fractional polymorphisms, that play the same role for valued CSPs as polymorphisms for the crisp case.

▶ **Theorem 38.** *For conservative* $\mathbf{\Gamma}$, $\mathcal{H}_c^{\mathbf{\Gamma}} = \mathbf{Up}(\mathbf{\Gamma}'_c)$.

6 Some experiments and open problems

We list here some experimental results and open problems

- In the case when $D = \{0, 1\}$, it can be shown that in the definition of $\mathbf{\Gamma}'$ Siggers pairs can be replaced with pairs (g, w) where g is unary and w is a ternary weak near unanimity operation on $g(D)$ (the number of such pairs on $\{0, 1\}$ is moderate). This allows a practical computation of $\mathbf{\Gamma}'$s core. We experimented with random structures over the boolean domain ($\Gamma = \{\rho_1, \rho_2, \rho_3\}, \mathrm{ar}(\rho_i) \leq 3$) and found that the domain size of $\mathbf{\Gamma}'$s core is never greater than 5.
- Since CSP($\mathbf{\Gamma}$) is reducible to CSP($\mathbf{\Gamma}'$), an interesting problem is to find necessary and sufficient conditions for $\mathbf{\Gamma} \sim \mathbf{\Gamma}'$ (i.e. for the case when such reduction is trivial). Experiments showed that if $\Gamma = \{\rho\}, \rho \subseteq \{0, 1\}^3$ is NP-hard, then $\mathbf{\Gamma} \sim \mathbf{\Gamma}'$. At the same time, if $\Gamma = \{\rho, \{0\}, \{1\}\}, \rho \subseteq \{0, 1\}^3$ is NP-hard, then $\mathbf{\Gamma} \not\sim \mathbf{\Gamma}'$.
- The number of Siggers pairs on D grows as $O(|D|^{|D|^4})$ which does not allow the calculation of $\mathbf{\Gamma}'$ even in the case when $|D| = 3$. Upper bounds on the domain size of $\mathbf{\Gamma}'$s core is an open problem.
- The problem of classifying all conservative $\mathbf{\Gamma}$ for which CSP($\mathbf{\Gamma}'_c$) is tractable (modification: is solvable in Datalog [2]) is also open.
- Are all crisp templates widely tractable, or is $\mathrm{CSP}_{\mathbf{\Gamma}'}(\mathbf{\Gamma})$ always tractable?

References

1 L. Barto and M. Kozik. New conditions for Taylor varieties and CSP. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 100–109, 2010.

2 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. In *Proceedings of the 23rd Annual Conference on Theoretical Aspects of Computer Science*, STACS'06, pages 646–659, Berlin, Heidelberg, 2006. Springer-Verlag.

3 A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

4 A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, 12(4), 2011. Article 24.

5 A. Bulatov, A. Krokhin, and A. Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

6 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. *CoRR*, abs/1703.03021, 2017. URL: http://arxiv.org/abs/1703.03021.

7 J. Bulín, D. Delic, M. Jackson, and T. Niven. On the reduction of the CSP dichotomy conjecture to digraphs. In Christian Schulte, editor, *CP*, volume 8124 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2013.

8 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

9 Martin C. Cooper and Stanislav Živný. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, 175(9-10):1555–1569, 2011.

10 Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 48–64, 2014.

11 Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

12 Martin J. Green and David A. Cohen. Domain permutation reduction for constraint satisfaction problems. *Artificial Intelligence*, 172(8):1094 – 1118, 2008.

13 Gregory Gutin, Arash Rafiey, Anders Yeo, and Michael Tso. Level of repair analysis and minimum cost homomorphisms of graphs. *Discrete Applied Mathematics*, 154(6):881–889, 2006.

14 Pavol Hell and Jaroslav Nešetřil. On the complexity of h-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92 – 110, 1990.

15 P. Jeavons, A. Krokhin, and S. Živný. The complexity of valued constraint satisfaction. *Bulletin of the EATCS*, 113:21–55, 2014.

16 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, June 1998.

17 Philippe Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In Richard Fikes and Wendy G. Lehnert, editors, *AAAI*, pages 731–736. AAAI Press / The MIT Press, 1993.

18 Keith Kearnes, Petar Marković, and Ralph McKenzie. Optimal strong Mal'cev conditions for omitting type 1 in locally finite varieties. *Algebra Univers.*, 72(1):91–100, 2014.

19 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.

20 V. Kolmogorov and S. Živný. The complexity of conservative valued CSPs. *Journal of the ACM*, 60(2), 2013. Article 10.

21 Vladimir Kolmogorov, Michal Rolinek, and Rustem Takhanov. Effectiveness of structural restrictions for hybrid csps. In *Proceedings of 26th International Symposium, (ISAAC 2015)*, pages 566–577. Springer Berlin Heidelberg, 2015.

22 Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? In *Proceedings of the 7th European Conference on Computer Vision-Part III*, ECCV '02, pages 65–81, London, UK, UK, 2002. Springer-Verlag.

23 M. Maróti and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra universalis*, 59(3–4):463–489, October 2008.

24 Arash Rafiey, Jeff Kinne, and Tomás Feder. Dichotomy for digraph homomorphism problems. *CoRR*, abs/1701.02409, 2017. URL: http://arxiv.org/abs/1701.02409.

25 T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226. ACM, 1978.

26 M. H. Siggers. A strong Mal'cev condition for locally finite varieties omitting the unary type. *Algebra universalis*, 64(1–2):15–20, October 2010.

27 Jacobus Stephanus Swarts. *The complexity of digraph homomorphisms: Local tournaments, injective homomorphisms and polymorphisms.* PhD thesis, University of Victoria, Canada, 2008.

28 Rustem S. Takhanov. A dichotomy theorem for the general minimum cost homomorphism problem. In *In Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 657–668, 2010.

29 Johan Thapper and Stanislav Zivny. Sherali-adams relaxations for valued csps. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 1058–1069, 2015.

30 Dmitriy Zhuk. The proof of CSP dichotomy conjecture. *CoRR*, abs/1704.01914, 2017. URL: `http://arxiv.org/abs/1704.01914`.

A $(1.4 + \epsilon)$-Approximation Algorithm for the 2-Max-Duo Problem[*][†][‡]

Yao Xu[1], Yong Chen[§][2], Guohui Lin[¶][3], Tian Liu[‖][4], Taibo Luo[**][5], and Peng Zhang[††][6]

1 **Department of Computing Science, University of Alberta, Edmonton, Canada**
xu2@ualberta.ca

2 **Department of Computing Science, University of Alberta, Edmonton, Canada and Department of Mathematics, Hangzhou Dianzi University, Zhejiang, China**
chenyong@hdu.edu.cn

3 **Department of Computing Science, University of Alberta, Edmonton, Canada**
guohui@ualberta.ca

4 **Key Laboratory of High Confidence Software Technologies (MOE), Institute of Software, School of Electronic Engineering and Computer Science, Peking University, Beijing, China**
lt@pku.edu.cn

5 **Business School, Sichuan University, Chengdu, China and Department of Computing Science, University of Alberta, Edmonton, Canada**
taibo@ualberta.ca

6 **School of Computer Science and Technology, Shandong University, Shandong, China**
algzhang@sdu.edu.cn

Abstract

The *maximum duo-preservation string mapping* (Max-Duo) problem is the complement of the well studied *minimum common string partition* (MCSP) problem, both of which have applications in many fields including text compression and bioinformatics. k-Max-Duo is the restricted version of Max-Duo, where every letter of the alphabet occurs at most k times in each of the strings, which is readily reduced into the well known *maximum independent set* (MIS) problem on a graph of maximum degree $\Delta \leq 6(k-1)$. In particular, 2-Max-Duo can then be approximated arbitrarily close to 1.8 using the state-of-the-art approximation algorithm for the MIS problem. 2-Max-Duo was proved APX-hard and very recently a $(1.6 + \epsilon)$-approximation was claimed, for any $\epsilon > 0$. In this paper, we present a vertex-degree reduction technique, based on which, we show that 2-Max-Duo can be approximated arbitrarily close to 1.4.

1998 ACM Subject Classification F.2.2 Pattern matching, G.2.1 Combinatorial algorithms, G.4 Algorithm design and analysis

Keywords and phrases Approximation algorithm, duo-preservation string mapping, string partition, independent set

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.66

[*] This work was partially supported by NSERC Canada and NSF China.

[†] A full version of this paper can be found at https://arxiv.org/abs/1702.0625606256.

[‡] All authors are supported by NSERC Canada.

[§] Chen is supported by the NSFC Grants No. 11401149, 11571252 and 11571087, and the China Scholarship Council Grant No. 201508330054.

[¶] Correspondence authors. Lin is supported by the NSFC Grant No. 61672323.

[‖] Liu is supported by the NSFC Grant Nos. 61370052 and 61370156.

[**] Luo is supported by the NSFC Grant No. 71371129 and the PSF China Grant No. 2016M592680.

[††] Zhang is supported by the NSFC Grant No. 61672323.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 66; pp. 66:1–66:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *minimum common string partition* (MCSP) problem is a well-studied string comparison problem in computer science, with applications in fields such as text compression and bioinformatics. MCSP was first introduced by Goldstein *et al.* [16], and can be defined as follows: Consider two length-n strings $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ over some alphabet Σ, such that B is a permutation of A. Let $\mathcal{P}_A$ be a *partition* of A, which is a multi-set of substrings whose concatenation in a certain order becomes A. The *cardinality* of $\mathcal{P}_A$ is the number of substrings in $\mathcal{P}_A$. The MCSP problem asks to find a minimum cardinality partition $\mathcal{P}_A$ of A which is also a partition of B. k-MCSP denotes the restricted version of MCSP where every letter of the alphabet Σ occurs at most k times in each of the two strings.

Goldstein *et al.* [16] have shown that the MCSP problem is NP-hard and APX-hard, even when $k = 2$. There have been several approximation algorithms [10, 11, 12, 16, 18, 19] presented since 2004, among which the current best result is an $O(\log n \log^* n)$-approximation for the general MCSP and an $O(k)$-approximation for k-MCSP. On the other hand, MCSP is proved to be *fixed parameter tractable* (FPT), with respect to k and/or the cardinality of the optimal partition [13, 17, 7, 8].

An ordered pair of consecutive letters in a string is called a *duo* of the string [16], which is said to be *preserved* by a partition if the pair resides inside a substring of the partition. Therefore, a length-ℓ substring in the partition *preserves* $\ell - 1$ duos of the string. With the complementary objective to that of MCSP, the problem of maximizing the number of duos preserved in the common partition is referred to as the *maximum duo-preservation string mapping* problem by Chen *et al.* [9], denoted as Max-Duo. Analogously, k-Max-Duo is the restricted version of Max-Duo where every letter of the alphabet Σ occurs at most k times in each string. In this paper, we focus on 2-Max-Duo, to design an improved approximation algorithm.

Along with Max-Duo, Chen *et al.* [9] introduced the *constrained maximum induced subgraph* (CMIS) problem, in which one is given an m-partite graph $G = (V_1, V_2, \ldots, V_m, E)$ with each V_i having n_i^2 vertices arranged in an $n_i \times n_i$ matrix, and the goal is to find n_i vertices in each V_i from different rows and different columns such that the number of edges in the induced subgraph is maximized. k-CMIS is the restricted version of CMIS where $n_i \le k$ for all i. Given an instance of Max-Duo, we may construct an instance of CMIS by setting m to be the number of distinct letters in the string A, and n_i to be the number of occurrences of the i-th distinct letter; the vertex in the (s,t)-entry of the $n_i \times n_i$ matrix "means" mapping the s-th occurrence of the i-th distinct letter in the string A to its t-th occurrence in the string B; and there is an edge between a vertex of V_i and a vertex of V_j if the two corresponding mappings together preserve a duo. Therefore, Max-Duo is a special case of CMIS, and furthermore k-Max-Duo is a special case of k-CMIS. Chen *et al.* [9] presented a k^2-approximation for k-CMIS and a 2-approximation for 2-CMIS, based on a linear programming and randomized rounding techniques. These imply that k-Max-Duo can also be approximated within a ratio of k^2 and 2-Max-Duo can be approximated within a ratio of 2.

Alternatively, an instance of the k-Max-Duo problem with the two strings $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ can be viewed as a bipartite graph $H = (A, B, F)$, constructed as follows: The vertices in A and B are $a_1, a_2, \ldots, a_n$ in order and $b_1, b_2, \ldots, b_n$ in order, respectively, and there is an edge between a_i and b_j if they are the same letter. The two edges $(a_i, b_j), (a_{i+1}, b_{j+1}) \in F$ are called a pair of *parallel* edges. This way, a common partition of the strings A and B corresponds one-to-one to a perfect matching in H, and the number of duos preserved by the partition is exactly the number of pairs of parallel edges in the matching.

Moreover, from the bipartite graph $H = (A, B, F)$, we can construct another graph $G = (V, E)$ in which every vertex of V corresponds to a pair of parallel edges of F, and there is an edge between two vertices of V if the two corresponding pairs of parallel edges of F *cannot* co-exist in any perfect matching of H (called *conflicting*, which can be determined in constant time; see Section 2 for more details). This way, one easily sees that a set of duos that can be preserved together, by a perfect matching of H, corresponds one-to-one to an independent set of G [16, 5]. Therefore, the Max-Duo problem can be cast as a special case of the well-known *maximum independent set* (MIS) problem [15]; furthermore, Boria *et al.* [5] showed that in such a reduction, an instance of k-Max-Duo gives rise to a graph with a maximum degree $\Delta \le 6(k-1)$. It follows that the state-of-the-art $\big((\Delta+3)/5+\epsilon\big)$-approximation algorithm for MIS [2], for any $\epsilon > 0$, is a $\big((6k-3)/5+\epsilon\big)$-approximation algorithm for k-Max-Duo. Especially, 2-Max-Duo can now be better approximated within a ratio of $1.8+\epsilon$. Boria *et al.* [5] proved that 2-Max-Duo is APX-hard, similar to 2-MCSP [16], via a linear reduction from MIS on cubic graphs. For MIS on cubic graphs, it is NP-hard to approximate within 1.00719 [3]. Besides, Boria *et al.* [5] claimed that 2-Max-Duo can be approximated within $1.6+\epsilon$, for any $\epsilon > 0$.

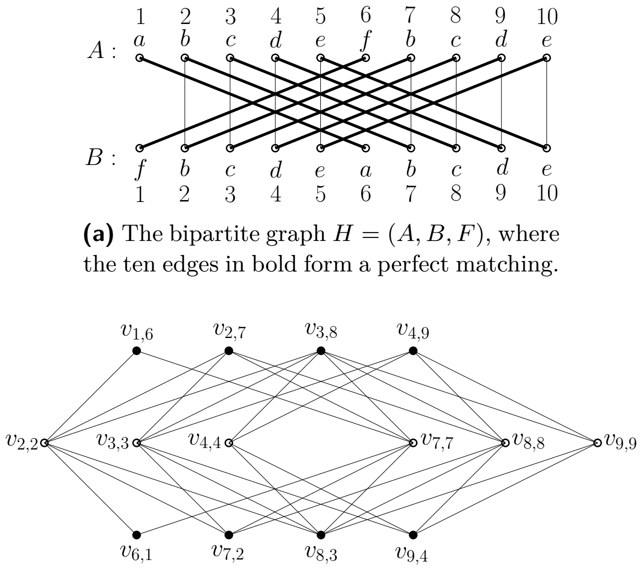
Recently, Boria *et al.* [4] presented a local search 3.5-approximation for the general Max-Duo problem. In the meantime, Brubach [6] presented a 3.25-approximation using a novel *combinatorial triplet matching*. Max-Duo has also been proved to be FPT by Beretta *et al.* [1], with respect to the number of preserved duos in the optimal partition. Most recently, two local search algorithms were independently designed for the general Max-Duo problem at the same time, achieving approximation ratios of 2.917 [20] and $2+\epsilon$ [14] for any $\epsilon > 0$, respectively. They both exceed the previously the best $\big((6k-3)/5+\epsilon\big)$-approximation algorithm for k-Max-Duo, when $k \ge 3$. In this paper, we focus on the 2-Max-Duo problem; using the above reduction to the MIS problem, we present a vertex-degree reduction scheme and design an improved $(1.4+\epsilon)$-approximation, for any $\epsilon > 0$.

The rest of the paper is organized as follows. We provide some preliminaries in Section 2, including several important structural properties of the graph constructed from the two given strings. The vertex-degree reduction scheme is also presented as a separate subsection in Section 2. The new approximation algorithm, denoted as Approx, is presented in Section 3, where we show that it is a $(1.4+\epsilon)$-approximation for 2-Max-Duo. We conclude the paper in Section 4.

2 Preliminaries

Consider an instance of the k-Max-Duo problem with two length-n strings $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ such that B is a permutation of A. Recall that we can view the instance as a bipartite graph $H = (A, B, F)$, where the vertices in A and B are $a_1, a_2, \ldots, a_n$ in order and $b_1, b_2, \ldots, b_n$ in order, respectively, and there is an edge between $a_i \in A$ and $b_j \in B$ if they are the same letter, denoted as $e_{i,j}$. See Figure 2.1a for an example, where $A = (a, b, c, d, e, f, b, c, d, e)$ and $B = (f, b, c, d, e, a, b, c, d, e)$. Note that $|F| \le kn$, and so H can be constructed in $O(n^2)$ time.

The two edges $e_{i,j}, e_{i+1,j+1} \in F$ are called a pair of *parallel* edges (and they are said to be parallel to each other); when both are included in a perfect matching of H, the corresponding duo (a_i, a_{i+1}) of A is preserved. Two pairs of parallel edges are *conflicting* if they cannot co-exist in any perfect matching of H. This motivates the following reduction from the


1 2 3 4 5 6 7 8 9 10
a b c d e f b c d e
A :
B :
f b c d e a b c d e
1 2 3 4 5 6 7 8 9 10
$v_{1,6}$ $v_{2,7}$ $v_{3,8}$ $v_{4,9}$
$v_{2,2}$ $v_{3,3}$ $v_{4,4}$ $v_{7,7}$ $v_{8,8}$ $v_{9,9}$
$v_{6,1}$ $v_{7,2}$ $v_{8,3}$ $v_{9,4}$

(a) The bipartite graph $H = (A, B, F)$, where the ten edges in bold form a perfect matching.

(b) The instance graph $G = (V, E)$ of MIS, where the eight filled vertices form an independent set.

Figure 2.1 An instance of the k-Max-Duo problem with $A = (a, b, c, d, e, f, b, c, d, e)$ and $B = (f, b, c, d, e, a, b, c, d, e)$. Figure 2.1a is the graphical view as a bipartite graph $H = (A, B, F)$, where a perfect matching consisting of the ten bold edges form into eight pairs of parallel edges, corresponding to the eight preserved duos $(a, b), (b, c), (c, d), (d, e), (f, b), (b, c), (c, d)$ and (d, e). Figure 2.1b shows the instance graph $G = (V, E)$ of MIS constructed from H, where the independent set $\{v_{1,6}, v_{2,7}, v_{3,8}, v_{4,9}, v_{6,1}, v_{7,2}, v_{8,3}, v_{9,4}\}$ corresponds to the eight pairs of parallel edges shown in Figure 2.1a, and consequently also corresponds to the eight preserved duos. In this instance, we have $k = 2$. Any maximum independent set of G must contain some of the degree-6 vertices, invalidating the $(1.6 + \epsilon)$-approximation for 2-Max-Duo proposed in [5].

k-Max-Duo problem to the MIS problem: From the bipartite graph $H = (A, B, F)$, we construct another graph $G = (V, E)$ in which a vertex $v_{i,j}$ of V corresponds to the pair of parallel edges $(e_{i,j}, e_{i+1,j+1})$ of F; two vertices of V are *conflicting* if and only if the two corresponding pairs of parallel edges are conflicting, and two conflicting vertices of V are adjacent in G. One can see that a set of duos of A that can be preserved all together, a set of pairwise non-conflicting pairs of parallel edges of F, and an independent set in G, are equivalent to each other. See Figure 2.1b for an example of the graph $G = (V, E)$ constructed from the bipartite graph H shown in Figure 2.1a. We note that $|V| \leq k(n - 1)$ and thus G can be constructed in $O(k^2 n^2)$ time from the instance of the k-Max-Duo problem.

In the graph G, for any $v \in V$, we use $N(v)$ to denote the set of its neighbors, that is, the vertices adjacent to v. The two ordered letters in the duo corresponding to the vertex v is referred to as the *letter content* of v. For example, in Figure 2.1b, the letter content of $v_{1,6}$ is "ab" and the letter content of $v_{6,1}$ is "fb".

Recall from the construction that there is an edge $e_{i,j}$ in the graph $H = (A, B, F)$ if $a_i = b_j$, and there is a vertex $v_{i,j}$ in the graph $G = (V, E)$ if the parallel edges $e_{i,j}$ and $e_{i+1,j+1}$ are in $H = (A, B, F)$.

▶ **Lemma 2.1.** *The graph $G = (V, E)$ has the following properties.*

1. *If $v_{i,j}$, $v_{i+2,j+2} \in V$, then $v_{i+1,j+1} \in V$.*
2. *Given any subset of vertices $V' \subset V$, let $F' = \{e_{i,j} | v_{i,j} \in V'\}$, $A' = \{a_i | e_{i,j} \in F'\}$,*

and $B' = \{b_j | e_{i,j} \in F'\}$. If the subgraph $H' = (A', B', F')$ in H is connected, then all the vertices of V' have the same letter content; and consequently for any two vertices $v_{i,j}, v_{h,\ell} \in V'$, we have both $v_{h,j}, v_{i,\ell} \in V$.

3. *For any $v_{i,j} \in V$, we have*

$$N(v_{i,j}) = \bigcup_{p=-1,0,1} \{v_{i'+p,j+p} \in V \mid i' \neq i\} \cup \bigcup_{p=-1,0,1} \{v_{i+p,j'+p} \in V \mid j' \neq j\}. \qquad (1)$$

Proof. The proof is mostly based on the definitions, or how the graphs are constructed from the instance of the 2-Max-Duo problem. Due to the space limit, the interested reader can find the detailed proofs of several earlier lemmas and corollaries in the full version. ◀

From Lemma 2.1 and its proof (in the full version), we see that for any vertex of V there are at most $k - 1$ conflicting vertices of each kind (corresponding to a set in Equation (1)). We thus have the following corollary.

▶ **Corollary 2.2.** *The maximum degree of the vertices in $G = (V, E)$ is $\Delta \leq 6(k - 1)$.*

2.1 When $k = 2$

We examine more properties for the graph $G = (V, E)$ when $k = 2$. First, from Corollary 2.2 we have $\Delta \leq 6$.

Berman and Fujito [2] have presented an approximation algorithm with a performance ratio arbitrarily close to $(\Delta + 3)/5$ for the MIS problem, on graphs with maximum degree Δ. This immediately implies a $(1.8 + \epsilon)$-approximation for 2-Max-Duo. Our goal is to reduce the maximum degree of the graph $G = (V, E)$ to achieve a better approximation algorithm. To this purpose, we examine all the degree-6 and degree-5 vertices in the graph G, and show a scheme to safely remove them from consideration when computing an independent set. This gives rise to a new graph G_2 with maximum degree at most 4, leading to a desired $(1.4 + \epsilon)$-approximation for 2-Max-Duo.

We remark that, in our scheme we first remove the degree-6 vertices from G to compute an independent set, and later we add half of these degree-6 vertices to the computed independent set to become the final solution. Contrary to the claim that there always exists a maximum independent set in G containing no degree-6 vertices [5, Lemma 1], the instance in Figure 2.1 shows that any maximum independent set for the instance must contain some degree-6 vertices, thus invalidating the $(1.6 + \epsilon)$-approximation for 2-Max-Duo proposed in [5].

In more details, the instance of 2-Max-Duo, illustrated in Figure 2.1, consists of two length-10 strings $A = (a, b, c, d, e, f, b, c, d, e)$ and $B = (f, b, c, d, e, a, b, c, d, e)$. The bipartite graph $H = (A, B, F)$ is shown in Figure 2.1a and the instance graph $G = (V, E)$ of the MIS problem is shown in Figure 2.1b. In the graph G, we have six degree-6 vertices: $v_{2,2}, v_{7,7}, v_{3,3}, v_{3,8}, v_{8,3}$ and $v_{8,8}$. One can check that $\{v_{1,6}, v_{2,7}, v_{3,8}, v_{4,9}, v_{6,1}, v_{7,2}, v_{8,3}, v_{9,4}\}$ is an independent set in G, of size 8. On the other hand, if none of these degree-6 vertices is included in an independent set, then because the four vertices $v_{4,4}, v_{4,9}, v_{9,4}, v_{9,9}$ form a square implying that at most two of them can be included in the independent set, the independent set would be of size at most 6, and thus can never be a maximum independent set in G.

Consider a duo (a_i, a_{i+1}) of the string A and for ease of presentation assume its letter content is "ab". If no duo of the string B has the same letter content "ab", then this duo of the string A can never be preserved; in fact this duo does not even become (a part of) a vertex of V of the graph G. If there is exactly one duo (b_j, b_{j+1}) of the string B having the same letter content "ab", then these two duos make up a vertex $v_{i,j} \in V$, and from Lemma 2.1 we know that the degree of the vertex $v_{i,j} \in V$ is at most 5, since there is no such vertex $v_{i,j'}$ with $j' \neq j$ sharing exactly the two letters a_i and a_{i+1} with $v_{i,j}$. Therefore, if the degree of the vertex $v_{i,j} \in V$ is six, then there must be two duos of the string A and two duos of the string B having the same letter content "*ab*". Assume the other duo of the string A and the other duo of the string B having the same letter content "*ab*" are $(a_{i'}, a_{i'+1})$ and $(b_{j'}, b_{j'+1})$, respectively. Then all four vertices $v_{i,j}, v_{i,j'}, v_{i',j}, v_{i',j'}$ exist in V. We call the subgraph of G induced on these four vertices a *square*, and denote it as $S(i,i';j,j') = (V(i,i';j,j'), E(i,i';j,j'))$, where $V(i,i';j,j') = \{v_{i,j}, v_{i,j'}, v_{i',j}, v_{i',j'}\}$ and $E(i,i';j,j') = \{(v_{i,j}, v_{i,j'}), (v_{i,j}, v_{i',j}), (v_{i',j'}, v_{i,j'}), (v_{i',j'}, v_{i',j})\}$ due to their conflicting relationships. One clearly sees that every square has a unique letter content, which is the letter content of its four member vertices.

In Figure 2.1b, there are three squares $S(2,7;2,7)$, $S(3,8;3,8)$ and $S(4,9;4,9)$, with their letter contents "*bc*", "*cd*" and "*de*", respectively. The above argument says that every degree-6 vertex of V must belong to a square, but the converse is not necessarily true, for example, all vertices of the square $S(4,9;4,9)$ have degree 4. We next characterize several properties of a square.

The following lemma is a direct consequence of how the graph G is constructed and the fact that $k = 2$.

▶ **Lemma 2.3.** *In the graph $G = (V,E)$ constructed from an instance of* 2-Max-Duo*,*

1. *for each index i, there are at most two distinct j and j' such that $v_{i,j}, v_{i,j'} \in V$;*
2. *if $v_{i,j}, v_{i,j'} \in V$ where $j' \neq j$, and $v_{i+1,j''+1} \in V$ (or symmetrically, $v_{i-1,j''-1} \in V$), then either $j'' = j$ or $j'' = j'$.*

▶ **Lemma 2.4.** *For any square $S(i,i';j,j')$ in the graph $G = (V,E)$, $N(v_{i,j}) = N(v_{i',j'})$, $N(v_{i,j'}) = N(v_{i',j})$, and $N(v_{i,j}) \cap N(v_{i,j'}) = \emptyset$. (Together, these imply that every vertex of V is adjacent to either none or exactly two of the four member vertices of a square.)*

▶ **Corollary 2.5.** *In the graph $G = (V,E)$, the degree-6 vertices can be partitioned into pairs, where each pair of degree-6 vertices belong to a square in G and they are adjacent to the same six other vertices, two inside the square and four outside of the square.*

Proof. We have seen that every degree-6 vertex in the graph G must be in a square. The above Lemma 2.4 states that the four vertices of a square $S(i,i';j,j')$ can be partitioned into two pairs, $\{v_{i,j}, v_{i',j'}\}$ and $\{v_{i,j'}, v_{i',j}\}$, and the two vertices inside each pair are non-adjacent to each other and have the same neighbors. In particular, if the vertex $v_{i,j}$ in the square $S(i,i';j,j')$ has degree 6, then Lemma 2.1 states that it is adjacent to the six vertices $v_{i-1,j'-1}, v_{i,j'}, v_{i+1,j'+1}, v_{i'-1,j-1}, v_{i',j}, v_{i'+1,j+1}$ (see an illustration in Figure 2.2). ◀

▶ **Corollary 2.6.** *If there is no square in the graph $G = (V,E)$, then every degree-5 vertex is adjacent to a degree-1 vertex.*

We say the two vertices $v_{i,j}$ and $v_{i+1,j+1}$ of V are *consecutive*; and we say the two squares $S(i,i';j,j')$ and $S(i+1,i'+1;j+1,j'+1)$ in G are *consecutive*. Clearly, two consecutive squares contain four pairs of consecutive vertices. The following Lemma 2.7 summarizes the fact that when two consecutive vertices belong to two different squares, then these two squares are also consecutive (and thus contain the other three pairs of consecutive vertices).

▶ **Lemma 2.7.** *In the graph G, if there are two consecutive vertices $v_{i,j}$ and $v_{i+1,j+1}$ belonging to two different squares $S(i_1,i'_1;j_1,j'_1)$ and $S(i_2,i'_2;j_2,j'_2)$ respectively, then $i_2 = i_1+1, i'_2 = i'_1+1, j_2 = j_1+1, j'_2 = j'_1+1$, i.e., these two squares are consecutive.*

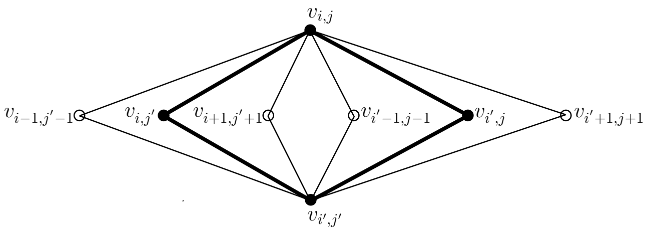


Figure 2.2 The square $S(i,i';j,j')$ shown in bold lines. The two non-adjacent vertices $v_{i,j}$ and $v_{i',j'}$ of the square form a pair stated in Corollary 2.5; they have 6 common neighbors, of which two inside the square and four outside of the square.

A series of p consecutive squares $\{S(i+q,i'+q;j+q,j'+q), q=0,1,\ldots,p-1\}$ in the graph G, where $p \geq 1$, is *maximal* if none of the square $S(i-1,i'-1;j-1,j'-1)$ and the square $S(i+p,i'+p;j+p,j'+p)$ exists in the graph G. Note that the non-existence of the square $S(i-1,i'-1;j-1,j'-1)$ in G does not rule out the existence of some of the four vertices $v_{i-1,j-1}, v_{i'-1,j'-1}, v_{i-1,j'-1}, v_{i'-1,j-1}$ in V; in fact by Lemma 2.1 there can be as many as two of these four vertices existing in V (however, more than two would imply the existence of the square). Similarly, there can be as many as two of the four vertices $v_{i+p,j+p}, v_{i'+p,j'+p}, v_{i+p,j'+p}, v_{i'+p,j+p}$ existing in V. In the sequel, a maximal series of p consecutive squares starting with $S(i,i';j,j')$ is denoted as $\mathcal{S}^p(i,i';j,j')$, where $p \geq 1$. See for an example in Figure 2.3b where there is a maximal series of 2 consecutive squares $\mathcal{S}^2(2,8;2,8)$, where the instance of the 2-Max-Duo is expanded slightly from the instance shown in Figure 2.1.

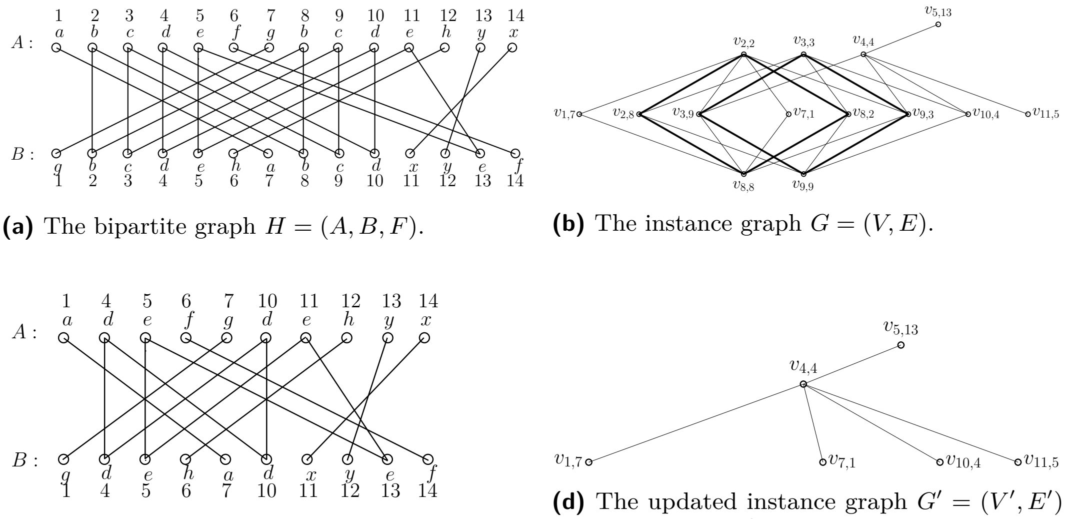
▶ **Lemma 2.8.** *Suppose $\mathcal{S}^p(i,i';j,j')$, where $p \geq 1$, exists in the graph G. Then,*

1. *the two substrings $(a_i, a_{i+1}, \ldots, a_{i+p})$ and $(a_{i'}, a_{i'+1}, \ldots, a_{i'+p})$ of the string A and the two substrings $(b_j, b_{j+1}, \ldots, b_{j+p})$ and $(b_{j'}, b_{j'+1}, \ldots, b_{j'+p})$ of the string B are identical and do not overlap;*
2. *if a maximum independent set of G contains less than $2p$ vertices from $\mathcal{S}^p(i,i';j,j')$, then it must contain either the four vertices $v_{i-1,j-1}, v_{i'-1,j'-1}, v_{i'+p,j+p}, v_{i+p,j'+p}$ or the four vertices $v_{i'-1,j-1}, v_{i-1,j'-1}, v_{i+p,j+p}, v_{i'+p,j'+p}$.*

Proof. By the definition of the square $S(i+q,i'+q;j+q,j'+q)$, we have $a_{i+q} = a_{i'+q}$ and $a_{i+q+1} = a_{i'+q+1}$; we thus conclude that the two substrings $(a_i, a_{i+1}, \ldots, a_{i+p})$ and $(a_{i'}, a_{i'+1}, \ldots, a_{i'+p})$ are identical. In Figure 2.3b, for $\mathcal{S}^2(2,8;2,8)$ the two substrings are "*bcd*". If these two substrings overlapped, then there would be three occurrences of at least one letter, contradicting the fact that $k = 2$. This proves the first item.

Note that the square $S(i-1,i'-1;j-1,j'-1)$ does not exist in the graph G, and thus at most two of its four vertices (which are $v_{i-1,j-1}, v_{i'-1,j-1}, v_{i-1,j'-1}$ and $v_{i'-1,j'-1}$) exist in V. We claim that if no vertex of the square $S(i,i';j,j')$ is in I^*, then there are exactly two of the four vertices $v_{i-1,j-1}, v_{i'-1,j-1}, v_{i-1,j'-1}$ and $v_{i'-1,j'-1}$ exist in V and they both are in I^*. Suppose otherwise there is at most one of the four vertices in I^*, say $v_{i-1,j-1}$; we may increase the size of I^* by removing $v_{i-1,j-1}$ while adding either the two vertices $v_{i,j}$ and $v_{i',j'}$ or the two vertices $v_{i',j}$ and $v_{i,j'}$ (depending on which vertices of the square $S(i+1,i'+1;j+1,j'+1)$ are in I^*), a contradiction.

Assume next that a vertex of the square $S(i,i';j,j')$ is in I^*, say $v_{i,j}$; then due to maximality of I^* and Lemma 2.4 both $v_{i,j}$ and $v_{i',j'}$ are in I^*. We claim and prove similarly as in the last paragraph that if no vertex of the square $S(i+1,i'+1;j+1,j'+1)$ is in I^*, then there are exactly two of the four vertices $v_{i-1,j-1}, v_{i'-1,j-1}, v_{i-1,j'-1}$ and $v_{i'-1,j'-1}$ exist in V


A :
B :
1 2 3 4 5 6 7 8 9 10 11 12 13 14
a b c d e f g b c d e h y x
g b c d e h a b c d x y e f
1 2 3 4 5 6 7 8 9 10 11 12 13 14
$v_{5,13}$
$v_{2,2}$ $v_{3,3}$ $v_{4,4}$
$v_{1,7}$ $v_{2,8}$ $v_{3,9}$ $v_{7,1}$ $v_{8,2}$ $v_{9,3}$ $v_{10,4}$ $v_{11,5}$
$v_{8,8}$ $v_{9,9}$
1 4 5 6 7 10 11 12 13 14
a d e f g d e h y x
g d e h a d x y e f
1 4 5 6 7 10 11 12 13 14
$v_{5,13}$
$v_{4,4}$
$v_{1,7}$ $v_{7,1}$ $v_{10,4}$ $v_{11,5}$

(a) The bipartite graph $H=(A,B,F)$.

(b) The instance graph $G=(V,E)$.

(c) The bipartite graph $H'=(A',B',F')$ after removal of $\mathcal{S}^2(2,8;2,8)$.

(d) The updated instance graph $G'=(V',E')$ after removal of $\mathcal{S}^2(2,8;2,8)$.

Figure 2.3 An instance of the 2-MAX-DUO problem with $A=(a,b,c,d,e,f,g,b,c,d,e,h,y,x)$ and $B=(g,b,c,d,e,h,a,b,c,d,x,y,e,f)$. The bipartite graph $H=(A,B,F)$ is shown in Figure 2.3a and the instance graph $G=(V,E)$ of the MIS problem is shown in Figure 2.3b. There is a maximal series of 2 squares $\mathcal{S}^2(2,8;2,8)$ in the graph G, with the four substrings "bcd". The bipartite graph $H'=(A',B',F')$ is shown in Figure 2.3c and the graph $G'=(V',E')$ is shown in Figure 2.3d, on $A'=(a,d,e,f,g,d,e,h,y,x)$ and $B'=(g,d,e,h,a,d,x,y,e,f)$. Applying the vertex contracting process on G also gives the graph G'.

and they both are in I^*. If there is a vertex of the square $S(i+1,i'+1;j+1,j'+1)$ in I^*, then it must be one of $v_{i+1,j+1}$ and $v_{i'+1,j'+1}$; and due to maximality and Lemma 2.4 both $v_{i+1,j+1}$ and $v_{i'+1,j'+1}$ are in I^*. And so on; repeatedly applying this argument, we claim and prove similarly that if no vertex of the square $S(i+p-1,i'+p-1;j+p-1,j'+p-1)$ is in I^*, then there are exactly two of the four vertices $v_{i-1,j-1}, v_{i'-1,j-1}, v_{i-1,j'-1}$ and $v_{i'-1,j'-1}$ exist in V and they both are in I^*. If there is a vertex of the square $S(i+p-1,i'+p-1;j+p-1,j'+p-1)$ in I^*, then it must be one of $v_{i+p-1,j+p-1}$ and $v_{i'+p-1,j'+p-1}$; and due to maximality and Lemma 2.4 both $v_{i+p-1,j+p-1}$ and $v_{i'+p-1,j'+p-1}$ are in I^*.

To summarize, we proved in the above two paragraphs that if I^* contains less than $2p$ vertices from $\mathcal{S}^p(i,i';j,j')$, then there are exactly two of the four vertices $v_{i-1,j-1}, v_{i'-1,j-1}$, $v_{i-1,j'-1}$ and $v_{i'-1,j'-1}$ exist in V and they both are in I^*; and these two vertices are either $v_{i-1,j-1}$ and $v_{i'-1,j'-1}$ or $v_{i'-1,j-1}$ and $v_{i-1,j'-1}$. Symmetrically, there are exactly two of the four vertices $v_{i+p,j+p}, v_{i'+p,j+p}, v_{i+p,j'+p}$ and $v_{i'+p,j'+p}$ exist in V and they both are in I^*; and these two vertices are either $v_{i+p,j+p}$ and $v_{i'+p,j'+p}$ or $v_{i'+p,j+p}$ and $v_{i+p,j'+p}$. Clearly from the above, when the combination is $v_{i-1,j-1}$ and $v_{i'-1,j'-1}$ versus $v_{i+p,j+p}$ and $v_{i'+p,j'+p}$, we may increase the size of I^* to contain exactly $2p$ vertices from $\mathcal{S}^p(i,i';j,j')$ without affecting any vertex outside of $\mathcal{S}^p(i,i';j,j')$, a contradiction. Therefore, the only possible combinations are $v_{i-1,j-1}$ and $v_{i'-1,j'-1}$ versus $v_{i'+p,j+p}$ and $v_{i+p,j'+p}$, and $v_{i'-1,j-1}$ and $v_{i-1,j'-1}$ versus $v_{i+p,j+p}$ and $v_{i'+p,j'+p}$. This proves the second item of the lemma. ◀

Suppose $\mathcal{S}^p(i,i';j,j')$, where $p\geq 1$, exists in the graph G. Let A' denote the string obtained from A by removing the two substrings $(a_i,a_{i+1},\ldots,a_{i+p-1})$ and $(a_{i'},a_{i'+1},\ldots,a_{i'+p-1})$ and concatenating the remainder together, and B' denote the string obtained from B by removing the two substrings $(b_j,b_{j+1},\ldots,b_{j+p-1})$ and

Algorithm 3.1 Approx – A high-level description of the approximation algorithm for 2-Max-Duo.

```
 1: Construct the graph $G = (V, E)$ from two input strings $A$ and $B$;
 2: while (there is a square in the graph) do
 3:     find a maximal series of squares;
 4:     locate the four identical substrings of $A$ and $B$ as in Lemma 2.8;
 5:     remove the corresponding substrings and accordingly update the graph;
 6: end while
 7: denote the resultant graph as $G_1 = (V_1, E_1)$;
 8: set $L_1$ to contain all degree-0 and degree-1 vertices of $G_1$;
 9: set $N[L_1]$ to be the closed neighborhood of $L_1$ in $G_1$, i.e. $N[L_1] = L_1 \cup N(L_1)$;
10: set $G_2 = G_1[V_1 - N[L_1]]$, the subgraph of $G_1$ induced on $V_1 - N[L_1]$;
11: compute an independent set $I_2$ in $G_2$ by the $((\Delta + 3)/5 + \epsilon)$-approximation in [2];
12: set $I_1 = I_2 \cup L_1$, an independent set in $G_1$;
13: return an independent set $I$ in $G$ using $I_1$ and Corollary 2.9.
```

$(b_{j'}, b_{j'+1}, \ldots, b_{j'+p-1})$ and concatenating the remainder. Let the graph $G' = (V', E')$ denote the instance graph of the MIS problem constructed from the two strings A' and B'. See for an example G' in Figure 2.3d, where there is a maximal series of 2 consecutive squares $\mathcal{S}^2(2, 8; 2, 8)$ in the graph G.

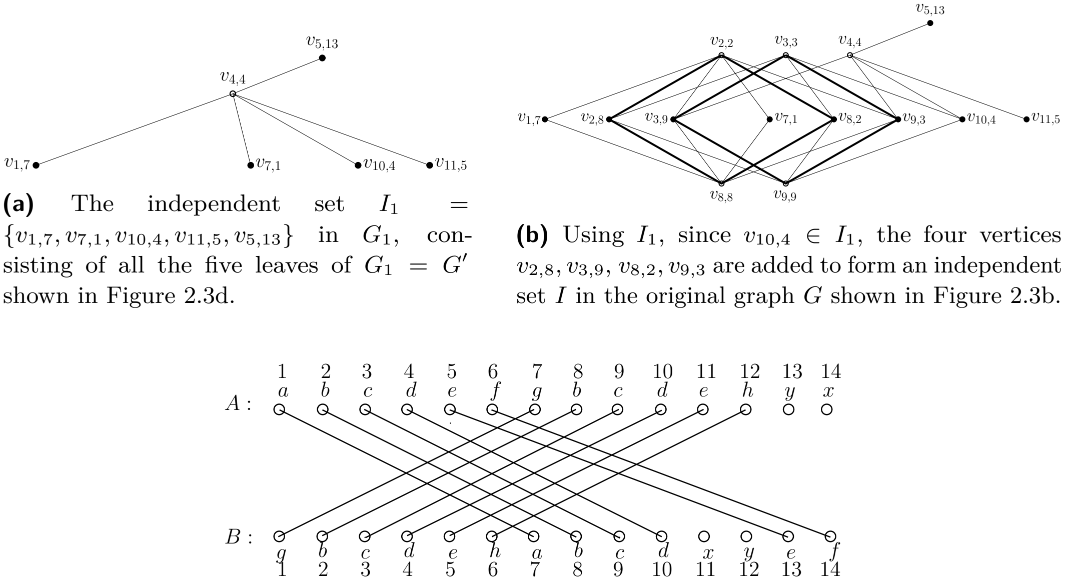
▶ **Corollary 2.9.** *Suppose $\mathcal{S}^p(i, i'; j, j')$, where $p \geq 1$, exists in the graph G. Then, the union of a maximum independent set in the graph $G' = (V', E')$ and certain $2p$ vertices from $\mathcal{S}^p(i, i'; j, j')$ becomes a maximum independent set in the graph $G = (V, E)$, where these certain $2p$ vertices are $v_{i,j}, v_{i+1,j+1}, \ldots, v_{i+p-1,j+p-1}$ and $v_{i',j'}, v_{i'+1,j'+1}, \ldots, v_{i'+p-1,j'+p-1}$ if $v_{i-1,j-1}$ or $v_{i+p,j+p}$ is in the maximum independent set in G', or they are $v_{i',j}, v_{i'+1,j+1}, \ldots$, $v_{i'+p-1,j+p-1}$ and $v_{i,j'}, v_{i+1,j'+1}, \ldots, v_{i+p-1,j'+p-1}$ if $v_{i'-1,j-1}$ or $v_{i'+p,j+p}$ is in the maximum independent set in G'.*

Iteratively applying the above string shrinkage process, or equivalently the vertex contracting process, associated with the elimination of a maximal series of consecutive squares. In $O(n)$ iterations, we achieve the final graph containing no squares, which we denote as $G_1 = (V_1, E_1)$.

3 An approximation algorithm for 2-Max-Duo

A high-level description of the approximation algorithm, denoted as Approx, for the 2-Max-Duo problem is depicted in Algorithm 3.1.

In more details, given an instance of the 2-Max-Duo problem with two length-n strings A and B, the first step of our algorithm is to construct the graph $G = (V, E)$, which is done in $O(n^2)$ time. In the second step (Lines 2–7 in Algorithm 3.1), it iteratively applies the vertex contracting process presented in Section 2 at the existence of a maximal series of consecutive squares, and at the end it achieves the final graph $G_1 = (V_1, E_1)$ which does not contain any square. This second step can be done in $O(n^2)$ time too since each iteration of vertex contracting process is done in $O(n)$ time and there are $O(n)$ iterations. In the third step (Lines 8–10 in Algorithm 3.1), let L_1 denote the set of singletons (degree-0 vertices) and leaves (degree-1 vertices) in the graph G_1; our algorithm removes all the vertices of L_1 and their neighbors from the graph G_1 to obtain the remainder graph $G_2 = (V_2, E_2)$. This step can be done in $O(n^2)$ time too due to $|V_1| \leq |V| \leq 2n$, and the resultant graph G_2 has maximum degree $\Delta \leq 4$ by Corollaries 2.5 and 2.6. (See for an example illustrated in Figure 3.1a.) In the fourth step (Lines 11–12 in Algorithm 3.1), our algorithm calls the



(a) The independent set $I_1 = \{v_{1,7}, v_{7,1}, v_{10,4}, v_{11,5}, v_{5,13}\}$ in G_1, consisting of all the five leaves of $G_1 = G'$ shown in Figure 2.3d.

(b) Using I_1, since $v_{10,4} \in I_1$, the four vertices $v_{2,8}, v_{3,9}, v_{8,2}, v_{9,3}$ are added to form an independent set I in the original graph G shown in Figure 2.3b.

(c) The parallel edges of H corresponding to the independent set I shown in Figure 3.1b, also correspond to the 9 preserved duos $(a,b), (b,c), (c,d), (e,f),$ $(g,b), (b,c), (c,d), (d,e), (e,h)$ for the instance shown in Figure 2.3a.

Figure 3.1 Illustration of the execution of our algorithm Approx on the instance shown in Figure 2.3. The independent set I_1 in the graph G_1 is shown in Figure 3.1a in filled circles, for which we did not apply the state-of-the-art approximation algorithm for the MIS problem. The independent set I in the graph G is shown in Figure 3.1b in filled circles, according to Corollary 2.9 the four vertices $v_{2,8}, v_{3,9}$, $v_{8,2}, v_{9,3}$ are added due to $v_{10,4} \in I_1$. The parallel edges of H corresponding to the vertices of I are shown in Figure 3.1c, representing a feasible solution to the 2-Max-Duo instance shown in Figure 2.3.

state-of-the-art approximation algorithm for the MIS problem [2] on the graph G_2 to obtain an independent set I_2 in G_2; and returns $I_1 = L_1 \cup I_2$ as an independent set in the graph G_1. The running time of this step is dominated by the running time of the state-of-the-art approximation algorithm for the MIS problem, which is a high polynomial in n and $1/\epsilon$. In the last step (Line 13 in Algorithm 3.1), using the independent set I_1 in G_1, our algorithm adds $2p$ vertices from each maximal series of p consecutive squares according to Corollary 2.9, to produce an independent set I in the graph G. (For an illustrated example see Figure 3.1b.) The last step can be done in $O(n)$ time.

The state-of-the-art approximation algorithm for the MIS problem on a graph with maximum degree Δ has a performance ratio of $(\Delta+3)/5+\epsilon$, for any $\epsilon > 0$ [2].

▶ **Lemma 3.1.** *In the graph $G_1 = (V_1, E_1)$, let* OPT_1 *denote the cardinality of a maximum independent set in G_1, and let* SOL_1 *denote the cardinality of the independent set I_1 returned by the algorithm* Approx. *Then,* $\mathrm{OPT}_1 \leq (1.4+\epsilon)\mathrm{SOL}_1$, *for any $\epsilon > 0$.*

▶ **Theorem 3.2.** *The* 2-Max-Duo *problem can be approximated within a ratio arbitrarily close to* 1.4, *by a linear reduction to the* MIS *problem.*

Proof. We prove by induction. At the presence of maximal series of p consecutive squares, we perform the vertex contracting process iteratively. In each iteration to handle one maximal series of p consecutive squares, let G and G' denote the graph before and after the contracting step, respectively. Let OPT' denote the cardinality of a maximum independent set in G', and let SOL' denote the cardinality of the independent set I' returned by the algorithm Approx. Given any $\epsilon > 0$, from Lemma 3.1, we may assume that $\mathrm{OPT}' \leq (1.4 + \epsilon)\mathrm{SOL}'$.

Let OPT denote the cardinality of a maximum independent set in G, and let SOL denote the cardinality of the independent set returned by the algorithm Approx, which adds $2p$ vertices from the maximal series of p consecutive squares to the independent set I' in G', according to Corollary 2.9, to produce an independent set I in the graph G. Lemma 2.8 states that $\mathrm{OPT} = \mathrm{OPT}' + 2p$. Therefore,

$$\mathrm{OPT} = \mathrm{OPT}' + 2p \leq (1.4 + \epsilon)\mathrm{SOL}' + 2p \leq (1.4 + \epsilon)(\mathrm{SOL}' + 2p) = (1.4 + \epsilon)\mathrm{SOL}.$$

This proves that for the original graph $G = (V, E)$ we also have $\mathrm{OPT} \leq (1.4 + \epsilon)\mathrm{SOL}$ accordingly. That is, the worst-case performance ratio of our algorithm Approx is $1.4 + \epsilon$, for any $\epsilon > 0$. The time complexity of the algorithm Approx has been determined to be polynomial at the beginning of the section, and it is dominated by the time complexity of the state-of-the-art approximation algorithm for the MIS problem. The theorem is thus proved. ◀

4 Conclusion

In this paper, we examined the Max-Duo problem, the complement of the well studied *minimum common string partition* problem. Based on an existing linear reduction to the *maximum independent set* (MIS) problem [16, 5], we presented a vertex-degree reduction technique for the 2-Max-Duo to reduce the maximum degree of the constructed instance graph to 4. Along the way, we uncovered many interesting structural properties of the constructed instance graph. This degree reduction enables us to adopt the state-of-the-art approximation algorithm for the MIS problem on low degree graphs [2] to achieve a $(1.4 + \epsilon)$-approximation for 2-Max-Duo, for any $\epsilon > 0$.

It is worth mentioning that our vertex-degree reduction technique can be applied for k-Max-Duo with $k \geq 3$. In fact, we had worked out the details for $k = 3$, to reduce the maximum degree of the constructed instance graph from 12 to 10, leading to a $(2.6 + \epsilon)$-approximation for 3-Max-Duo, for any $\epsilon > 0$. Nevertheless, the $(2.6 + \epsilon)$-approximation is superseded by the $(2 + \epsilon)$-approximation for the general Max-Duo [14].

It would be worthwhile to investigate whether the maximum degree can be further reduced to 3, by examining the structural properties associated with the degree-4 vertices. On the other hand, it is also interesting to examine whether a better-than-1.4 approximation algorithm can be designed directly for the MIS problem on those degree-4 graphs obtained at the end of the vertex contracting process.

References

1 S. Beretta, M. Castelli, and R. Dondi. Parameterized tractability of the maximum-duo preservation string mapping problem. *Theoretical Computer Science*, 646:16–25, 2016.

2 P. Berman and T. Fujito. On approximation properties of the independent set problem for low degree graphs. *Theory of Computing Systems*, 32:115–132, 1999.

3 P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proceedings of the of 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, pages 200–209, 1999.

4 N. Boria, G. Cabodi, P. Camurati, M. Palena, P. Pasini, and S. Quer. A 7/2-approximation algorithm for the maximum duo-preservation string mapping problem. In *Proceedings of*

the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016), volume 54 of *LIPIcs*, pages 11:1–11:8, 2016.

5 N. Boria, A. Kurpisz, S. Leppänen, and M. Mastrolilli. Improved approximation for the maximum duo-preservation string mapping problem. In *Proceedings of the 14th International Workshop on Algorithms in Bioinformatics (WABI 2014)*, volume 8701 of *LNBI*, pages 14–25, 2014.

6 B. Brubach. Further improvement in approximating the maximum duo-preservation string mapping problem. In *Proceedings of the 16th International Workshop on Algorithms in Bioinformatics (WABI 2016)*, volume 9838 of *LNBI*, pages 52–64, 2016.

7 L. Bulteau, G. Fertin, C. Komusiewicz, and I. Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In *Proceedings of the 13th International Workshop on Algorithms in Bioinformatics (WABI 2013)*, volume 8126 of *LNBI*, pages 244–258, 2013.

8 L. Bulteau and C. Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 102–121, 2014.

9 W. Chen, Z. Chen, N. F. Samatova, L. Peng, J. Wang, and M. Tang. Solving the maximum duo-preservation string mapping problem with linear programming. *Theoretical Computer Science*, 530:1–11, 2014.

10 X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2:302–315, 2005.

11 M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2004) and the 8th International Workshop on Randomization and Computation (RANDOM 2004)*, volume 3122 of *LNCS*, pages 84–95, 2004.

12 G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3:2:1–2:19, 2007.

13 P. Damaschke. Minimum common string partition parameterized. In *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics (WABI 2008)*, volume 5251 of *LNBI*, pages 87–98, 2008.

14 B. Dudek, P. Gawrychowski, and P. Ostropolski-Nalewaja. A family of approximation algorithms for the maximum duo-preservation string mapping problem. *arXiv*, 1702.02405, 2017.

15 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

16 A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC 2004)*, volume 3341 of *LNCS*, pages 484–495, 2004.

17 H. Jiang, B. Zhu, D. Zhu, and H. Zhu. Minimum common string partition revisited. *Journal of Combinatorial Optimization*, 23:519–527, 2012.

18 P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. In *Proceedings of the 4th International Workshop on Approximation and Online Algorithms (WAOA 2006)*, volume 4368 of *LNCS*, pages 279–289, 2006.

19 P. Kolman and T. Waleń. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, 155:327–336, 2007.

20 Y. Xu, Y. Chen, T. Luo, and G. Lin. A local search 2.917-approximation algorithm for duo-preservation string mapping. *arXiv*, 1702.01877, 2017.

Envy-free Matchings with Lower Quotas[*][†]

Yu Yokoi

National Institute of Informatics, Tokyo, Japan
yokoi@nii.ac.jp

Abstract

While every instance of the Hospitals/Residents problem admits a stable matching, the problem with lower quotas (HR-LQ) has instances with no stable matching. For such an instance, we expect the existence of an envy-free matching, which is a relaxation of a stable matching preserving a kind of fairness property.

In this paper, we investigate the existence of an envy-free matching in several settings, in which hospitals have lower quotas. We first provide an algorithm that decides whether a given HR-LQ instance has an envy-free matching or not. Then, we consider envy-freeness in the Classified Stable Matching model due to Huang (2010), i.e., each hospital has lower and upper quotas on subsets of doctors. We show that, for this model, deciding the existence of an envy-free matching is NP-hard in general, but solvable in polynomial time if quotas are paramodular.

1998 ACM Subject Classification F.2.2 Computations on discrete structures, G.2.1 Combinatorial algorithms

Keywords and phrases stable matchings, envy-free matchings, lower quotas, polynomial time algorithm, paramodular functions

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.67

1 Introduction

Since the seminal work of Gale and Shapley [11], the *Hospitals/Residents problem* (HR, for short), or the *College Admission problem*, has been studied extensively [14, 20, 27]. They proposed an algorithm that finds a stable matching in linear time for every instance. In this problem, each hospital has an upper quota for the number of doctors assigned to it. In some applications, each hospital also has a lower quota for the number of doctors it receives. That is, we want to consider the Hospitals/Residents problem with lower quotas (HR-LQ, for short). Unfortunately, for HR-LQ, we cannot ensure the existence of a stable matching. However, it is easy to decide whether there is a stable matching or not for a given HR-LQ instance, because the number of doctors assigned to each hospital is identical for any stable matching (according to the well-known Rural Hospitals Theorem [12, 24, 25, 26]).

When a given HR-LQ instance has no stable matching, one natural approach is to weaken stability concept while preserving some kind of fairness. *Envy-freeness* [30] (also called *fairness* in the school choice literature [8, 13]) of matchings is a relaxation of stability obtained by giving up efficiency. Similarly to stability, envy-freeness forbids the existence of a doctor who has justified envy toward some other doctor, but it tolerates the existence of a doctor who claims a hospital's vacant seat. The importance of envy-freeness and its variants has recently been recognized in the context of constrained matching [8, 13, 18, 19, 4], and structural properties of envy-free matchings were investigated in [30].

[*] The full version is available at http://arxiv.org/abs/1704.04888.

[†] This work was supported by JST CREST, Grant Number JPMJCR14D2, Japan.

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 67; pp. 67:1–67:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Envy-free matchings naturally arise when we find a matching in the following ad hoc manner. For an HR-LQ instance, suppose that we find a stable matching while disregarding the lower quotas, and that the obtained matching does not meet the lower quotas. Let us reduce the upper quotas of hospitals that receive many doctors, and again find a stable matching while disregarding the lower quotas, and repeat. If we find a stable matching that meets the lower quotas after repeating such adjustments, then the obtained matching is an envy-free matching of the original instance (see Proposition 4).

Because an envy-free matching is a relaxation of a stable matching, it is more likely to exist. Indeed, if all doctor-hospital pairs are acceptable and the sum of lower quotas of all hospitals does not exceed the number of doctors, then we can ensure the existence of an envy-free matching. (This follows from the results of Fragiadakis et al. [8]). However, if not all pairs are acceptable, then even an envy-free matching may fail to exist. Moreover, deciding the existence of an envy-free matching is not so simple because envy-free matchings have different sizes unlike stable matchings.

Our Contribution

In this paper, we study envy-free matchings for the HR-LQ model and its generalizations. In our models, not all doctor-hospital pairs are acceptable (i.e., preference lists are incomplete).

We first investigate envy-free matchings in the setting of HR-LQ. We provide the following characterization of the existence of an envy-free matching. Let I be a given HR-LQ instance and let I' be an HR instance obtained from I by removing lower quotas and replacing upper quotas with the original lower quotas. We prove that I has an envy-free matching if and only if every hospital is full in a stable matching of I' (Theorem 6). Combined with the rural hospitals theorem, this characterization yields an efficient algorithm to decide the existence of an envy-free matching for an HR-LQ instance. That is, we can decide it by finding a stable matching for the HR instance whose upper quotas are the original lower quotas, and checking whether all hospitals are full or not.

Next, we move to a generalized model, in which each hospital imposes an upper and a lower quota on each subset of doctors. That is, we consider an envy-free matching version of Huang's *Classified Stable Matching* [17] (CSM, for short). (See "Related Works" below for results on stable matchings of CSM and its generalizations.) In Huang's original model, each hospital has a family of sets of doctors, called *classes*, and each class has an upper and a lower quota. We formulate this setting by letting each hospital have a pair of set functions defined on the set of acceptable doctors. These two functions respectively represent upper quotas and lower quotas. For this model, we show that it is NP-hard to decide the existence of an envy-free matching, even if the number of non-trivial quotas is linear (Theorem 6). The proof is by a reduction from the NP-complete problem (3,B2)-SAT [2].

Then, we provide a tractable special case of CSM. We show that if the pair of lower and upper quota functions of each hospital is *paramodular* [9] (see Section 4 for the definition), then we can decide the existence of an envy-free matching in polynomial time. This means that the problem is tractable if the family of acceptable doctor sets forms a generalized matroid for each hospital. A *generalized matroid* [28] (also called an *$M^\natural$-convex family* [22]) is a family of subsets satisfying a certain axiom called the exchange axiom. It is known that a paramodular function pair defines a generalized matroid and vice versa. Because constraints defined on a laminar (or hierarchical) family yield a generalized matroid, our tractable special case includes a case in which each hospital defines quotas on a laminar family of doctors.

Related Works

Recently, the study of matching models with lower quotas has developed substantially [1, 7, 13, 15, 16, 17, 20, 21]. The Hospitals/Residents problem with lower quotas (HR-LQ) was first studied by Hamada et al. [15, 16], who considered the minimization of the number of blocking pairs subject to upper and lower quotas. They showed the NP-hardness of the problem, gave an inapproximability result, and provided an exponential-time exact algorithm. Motivated by the matching scheme used in Hungary's higher education sector, Biró et al. [3] considered a version of HR-LQ in which hospitals (i.e., colleges) are allowed to be closed, i.e., each hospital is assigned enough doctors or no doctor. They showed the NP-completeness to decide the existence of a stable matching.

The Classified Stable Matching problem (CSM), proposed by Huang [17], is a generalization of HR-LQ without hospital closures. In this model, each hospital (or institute, in Huang's terminology) has a classification of doctors (i.e., applicants) based on their expertise and gives an upper and lower quota for each class. Huang showed that it is NP-complete in general to decide the existence of a stable matching, and proved that it is solvable in polynomial time if classes form a laminar family. For this tractable special case, Fleiner and Kamiyama [7] gave a concise explanation in terms of matroids, and their framework is generalized by Yokoi [31] to a framework with generalized matroids.

To cope with the nonexistence of a stable matching in constrained matching models (not only models with lower quotas but also with other types of constraints such as regional constraints), several relaxations of stability have been proposed. See, e.g., Kamada and Kojima [18, 19], Fragiadakis et al. [8], and Goto et al. [13]. Envy-freeness is one of them that places emphasis on fairness rather than efficiency. Fragiadakis et al. [8] provided a strategy-proof algorithm that always finds an envy-free matching (or fair matching, in their terminology) of HR-LQ under the assumption that all doctor-hospital pairs are acceptable. The outcome of their mechanism also fulfills a second-best efficiency (i.e., nonwastefulness) property. Their framework is generalized in Goto et al. [13] so that regional quotas can be handled.

Here we compare our models with the above models. Unlike the models of Goto et al. [13] and Kamada and Kojima [18, 19], our models cannot handle regional quotas. Instead, our CSM model (in Sections 3 and 4) allows each hospital to have quotas on classes of doctors, which are not dealt with in their models. The setting of a tractable special case of CSM described in Section 4 is equivalent to a many-to-one case of Yokoi's model [31], which studied stable matchings. Neither [31] nor the study in this paper relies on the results of the other, while both of them utilize the matroid framework of Fleiner [5, 6].

The remainder of this paper is organized as follows. Section 2 investigates envy-free matchings in the Hospitals/Residents problem with lower quotas (HR-LQ). In Section 3, we define an envy-free matching in the classified stable matching (CSM) model, and show the NP-hardness of its existence test. As its tractable special case, Section 4 presents results on CSM with paramodular quota functions. Due to space constraints, we defer the proofs for the theorems and corollary in Section 4 to the full version.

2 Envy-freeness in HR with lower quotas

In this section, we investigate envy-free matchings in the Hospitals/Residents problem with lower quotas (HR-LQ).

There are two disjoint sets D and H, which represent doctors and hospitals, respectively. A set of acceptable doctor-hospital pairs is denoted by $E \subseteq D \times H$.

For each doctor $d \in D$, its acceptable hospital set is denoted by

$$A(d) := \{\, h \in H \mid (d,h) \in E \,\} \subseteq H,$$

and d has a preference list (strict order) $\succ_d$ on $A(d)$. Similarly, for each hospital $h \in H$,

$$A(h) := \{\, d \in D \mid (d,h) \in E \,\} \subseteq D,$$

and h has a preference $\succ_h$ on $A(h)$. Each hospital h has a lower quota $l_h \in \mathbf{Z}$ and an upper quota $u_h \in \mathbf{Z}$ with $0 \leq l_h \leq u_h \leq |A(h)|$.

We call a tuple $I = (D, H, E, \succ_{DH}, \{(l_h, u_h)\}_{h \in H})$ an **HR-LQ instance**, where $\succ_{DH}$ is an abbreviated notation for the union of $\{\succ_d\}_{d \in D}$ and $\{\succ_h\}_{h \in H}$. In particular, if $l_h = 0$ for all $h \in H$, we call it an **HR instance**. An arbitrary subset M of E is called an **assignment**. For any assignment M, we denote $M(d) = \{\, h \in A(d) \mid (d,h) \in M \,\}$ for each $d \in D$ and $M(h) = \{\, d \in A(h) \mid (d,h) \in M \,\}$ for each $h \in H$. If $|M(d)| = 1$, the notation $M(d)$ is also used to refer its single element.

An assignment M is called a **matching** (or, said to be **feasible**) if $|M(d)| \leq 1$ for each $d \in D$ and $l_h \leq |M(h)| \leq u_h$ for each $h \in H$. In a matching M, a doctor d is **unassigned** (resp., **assigned**) if $M(d) = \emptyset$ (resp., $|M(d)| = 1$), and h is **undersubscribed** (resp., **full**) if $|M(h)| < u_h$ (resp., $|M(h)| = u_h$).

▶ **Definition 1.** For a matching M, an unassigned pair $(d,h) \in E \setminus M$ is a **blocking pair** if (i) d is unassigned or $h \succ_d M(d)$, and (ii) h is undersubscribed or there is $d' \in M(h)$ with $d \succ_h d'$. A matching M is **stable** if there is no blocking pair.

For an HR instance, it is known that the algorithm of Gale and Shapley [11] always finds a stable matching. The set of stable matchings has the following property.

▶ **Proposition 2** ("Rural Hospitals" Theorem [12, 24, 26])**.** *For a given HR instance, any two stable matchings M, M' satisfy $|M(h)| = |M'(h)|$ for every $h \in H$. Moreover $M(h) = M'(h)$ if h is undersubscribed in M or M'.*

As mentioned in the Introduction, if hospitals have lower quotas, then we cannot guarantee the existence of a stable matching anymore. By Proposition 2, however, we can easily check the existence by finding a stable matching while disregarding lower quotas, and checking whether the obtained matching meets lower quotas.

For an instance that has no stable matching, we want to obtain some matching that still has a kind of fairness. As a relaxation of the concept of stability, envy-freeness (also called fairness) of matchings has been proposed [8, 30].

▶ **Definition 3.** For a matching M, a doctor d has **justified envy** toward d' with $M(d') = h$ if (i) d is unassigned or $h \succ_d M(d)$ and (ii) $d \succ_h d'$. A matching M is **envy-free** if no doctor has justified envy.

Note that, if d has justified envy toward d' with $M(d) = h$, then it means that (d,h) is a blocking pair. Thus, stability implies envy-freeness. The envy-freeness of a matching is also regarded as the stability with reduced upper quotas, as follows.

▶ **Proposition 4.** *For $I = (D, H, E, \succ_{DH}, \{(l_h, u_h)\}_{h \in H})$, an assignment M is an envy-free matching if and only if M is a stable matching of $I' = (D, H, E, \succ_{DH}, \{(l_h, u'_h)\}_{h \in H})$ for some $\{u'_h\}_{h \in H}$ with $u'_h \leq u_h$ $(h \in H)$.*

Doctor's preferences	Hospitals' preferences	
$d_1 : h_1$	$h_1 : d_2 \quad d_1$	$(l_{h_1} = 1,\ u_{h_1} = 2)$
$d_2 : h_1 \quad h_2$	$h_2 : d_2$	$(l_{h_2} = 1,\ u_{h_2} = 2)$

Figure 1 An instance of HR-LQ with no envy-free matching.

Proof. The "if" part is clear because feasibility in I' implies that in I, and stability implies envy-freeness. For the "only if" part, suppose that M is envy-free in I and set $u'_h := |M(h)|$ for each $h \in H$. Then, M is feasible for I' and all hospitals are full, and hence there is no doctor who claims a hospital's vacant seat. Because M is envy-free, it is stable in I'. ◀

By Proposition 4, to check whether we can obtain a stable matching by reducing upper quotas, it suffices to check for the existence of an envy-free matching.

Under the assumption that all doctor-hospital pairs are acceptable and the sum of lower quotas does not exceed the number of doctors, Fragiadakis et al. [8] provided a strategy-proof mechanism that always finds an envy-free matching. As a corollary, we have the following.

▶ **Proposition 5.** *For an instance $I = (D, H, E, \succ_{DH}, \{(l_h, u_h)\}_{h \in H})$ such that $E = D \times H$ and $|D| \geq \sum_{h \in H} l_h$, there exists an envy-free matching.*

However, if not all pairs are acceptable, then even an envy-free matching may not exist. Figure 1 shows an instance with $D = \{d_1, d_2\}$, $H = \{h_1, h_2\}$, $E = \{(d_1, h_1), (d_2, h_1), (d_2, h_2)\}$, $l_{h_1} = l_{h_2} = 1$, and $u_{h_1} = u_{h_2} = 2$. For this instance, $M = \{(d_1, h_1), (d_2, h_2)\}$ is the unique feasible matching, but it is not envy-free because d_2 has justified envy toward d_1. Hence, there is no envy-free matching.

Note that an envy-free matching does exist if there is no lower quota, because empty matching is clearly envy-free. Therefore, the existence test of an envy-free matching is non-trivial when incomplete lists and lower quotas are introduced simultaneously. Here we provide a characterization.

▶ **Theorem 6.** *$I = (D, H, E, \succ_{DH}, \{(l_h, u_h)\}_{h \in H})$ has an envy-free matching if and only if some stable matching M' of the HR instance $I' = (D, H, E, \succ_{DH}, \{(0, l_h)\}_{h \in H})$ satisfies $|M'(h)| = l_h$ for all $h \in H$.*

Proof. For the "if" part, let M' be a stable matching of I' satisfying $|M'(h)| = l_h$ for all $h \in H$. Then, M' is feasible for I' and no doctor has justified envy because M' has no blocking pair. Thus, M' is an envy-free matching of I.

For the "only if" part, assume that I has an envy-free matching M. Suppose, to the contrary, a stable matching M' of I' satisfies $|M'(h^*)| < l_{h^*}$ for some $h^* \in H$. Let us denote $N = M \setminus M'$ and $N' = M' \setminus M$. For every $h \in H$, because $|M'(h)| \leq l_h \leq |M(h)|$, we have $|N'(h)| \leq |N(h)|$. In particular, $|N'(h^*)| < |N(h^*)|$ follows from $|M'(h^*)| < l_{h^*}$.

Consider a bipartite graph $G = (D, H; N \cup N')$, i.e., a graph between doctors and hospitals with edge set $N \cup N' = M \triangle M'$. Let G^* be a connected component of G including h^*, and denote by D^* and H^* the sets of doctors and hospitals in G^*, respectively. Because there is no edge connecting G^* and the outside, $\sum_{d \in D^*} |N(h)| = \sum_{h \in H^*} |N(h)|$ and $\sum_{d \in D^*} |N'(h)| = \sum_{h \in H^*} |N'(h)|$. As $|N'(h^*)| < |N(h^*)|$ and $|N'(h)| \leq |N(h)|$ for any $h \in H^*$, we obtain

$$\textstyle\sum_{d \in D^*} |N'(h)| = \sum_{h \in H^*} |N'(h)| < \sum_{h \in H^*} |N(h)| = \sum_{d \in D^*} |N(h)|.$$

Then, there exists $d^* \in D^*$ with $|N'(d^*)| < |N(d^*)|$, which implies $N'(d^*) = \emptyset$ and $|N(d^*)| = 1$ because $N' = M' \setminus M$ and $N = M \setminus M'$ are subsets of matchings. As G^* is a connected bipartite graph, there is a path $d_0\, h_0\, d_1\, h_1 \dots d_k\, h_k$ with $d_0 = d^*$ and $h_k = h^*$. Also, as $|N(d_i)| \leq 1$ and $|N'(d_i)| \leq 1$ for $i = 0, 1, \dots k$, this path alternately uses edges in $N = M \setminus M'$ and $N' = M' \setminus M$. Because $N'(d^*) = \emptyset$ and $|N(d^*)| = 1$, we have

$$\begin{aligned} M'(d_0) &= \emptyset, \\ (d_i, h_i) &\in M \setminus M' \qquad (i = 0, 1, \dots, k), \\ (d_{i+1}, h_i) &\in M' \setminus M \qquad (i = 0, 1, \dots, k-1). \end{aligned}$$

The doctor d_0 is unassigned in M' and finds h_0 acceptable because $(d_0, h_0) \in M$. Hence, the stability of M' implies that h_0 prefers $d_1 \in M'(h_0)$ to d_0. Then, the envy-freeness of M implies that d_1 prefers $h_1 = M(d_1)$ to h_0. In this way, we obtain

$$\begin{aligned} &d_{i+1} \succ_{h_i} d_i \qquad (i = 0, 1, \dots, k-1), \\ &h_{i+1} \succ_{d_{i+1}} h_i \qquad (i = 0, 1, \dots, k-1). \end{aligned}$$

Thus, $M(d_k) = h_k \succ_{d_k} h_{k-1} = M'(d_k)$. Because $h_k = h^*$ satisfies $|M'(h_k)| < l_{h_k}$, then (d_k, h_k) is a blocking pair in I', which contradicts the stability of M'. ◀

Theorem 6 ensures that the following algorithm decides the existence of an envy-free matching of an HR-LQ instance $I = (D, H, E, \succ_{DH}, \{(l_h, u_h)\}_{h \in H})$.

Algorithm EF-HR-LQ

Step1. Find a stable matching M' of $I' = (D, H, E, \succ_{DH}, \{(0, l_h)\}_{h \in H})$.

Step2. return M' if $|M'(h)| = l_h$ for all $h \in H$, and otherwise "there is no envy-free matching."

Since the Gale-Shapley algorithm finds a stable matching of an HR instance in $O(|E|)$ time, we obtain the following theorem.

▶ **Theorem 7.** *For any HR-LQ instance $I = (D, H, E, \succ_{DH}, \{(l_h, u_h)\}_{h \in H})$, the algorithm* EF-HR-LQ *decides whether I has an envy-free matching or not in $O(|E|)$ time.*

3 Envy-freeness in Classified Stable Matching

In this section, we consider the envy-freeness in a model in which each hospital has lower and upper quotas on subsets of doctors. This can be regarded as an envy-free matching version of the Classified Stable Matching, proposed by Huang [17]. Similarly to Section 2, we have doctors D, hospitals H, acceptable pairs $E \subseteq D \times H$, and preferences $\succ_{DH}$.

The only difference from HR-LQ is that, in the current model, each hospital $h \in H$ has a pair of functions $p_h, q_h : 2^{A(h)} \to \mathbf{Z}$, instead of a pair of numbers l_h, u_h. These functions define a lower and an upper quota for each subset of acceptable doctors. Throughout this paper, we assume that for any hospital h, the functions p_h and q_h satisfy

$$0 \leq p_h(B) \leq q_h(B) \leq |B| \qquad (B \subseteq A(h)).$$

We call such a tuple $(D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$ a **CSM instance**. For each $h \in H$, the family of **acceptable** subsets of doctors is denoted by

$$\mathcal{F}(p_h, q_h) := \{\, X \subseteq A(h) \mid \forall B \subseteq A(h) : p_h(B) \leq |X \cap B| \leq q_h(B) \,\}.$$

For any $h \in H$, we say that $B \subseteq A(h)$ has a **non-trivial lower** (resp., **upper**) **constraint** if $p_h(B) > 0$ (resp., $q_h(B) < |B|$). We denote the family of constrained subsets by

$$\mathcal{C}(p_h, q_h) := \{\, B \subseteq A(h) \mid p_h(B) > 0 \;\text{ or }\; q_h(B) < |B| \,\}.$$

Then, we see that $\mathcal{F}(p_h, q_h)$ is represented as

$$\mathcal{F}(p_h, q_h) = \{\, X \subseteq A(h) \mid \forall B \subseteq \mathcal{C}(p_h, q_h) : p_h(B) \leq |X \cap B| \leq q_h(B) \,\}.$$

For a CSM instance $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$, $M \subseteq E$ is called a **matching** (or, said to be **feasible**) if $|M(d)| \leq 1$ for each $d \in D$ and $M(h) \in \mathcal{F}(p_h, q_h)$ for each $h \in H$.

▶ **Definition 8.** For a matching M, an unassigned pair $(d, h) \in E \setminus M$ is a **blocking pair** if (i) d is unassigned or $h \succ_d M(d)$, and (ii) $M(h) + d \in \mathcal{F}(p_h, q_h)$ or $M(h) + d - d' \in \mathcal{F}(p_h, q_h)$ for some $d' \in M(h)$ with $d \succ_h d'$. A matching M is **stable** if there is no blocking pair.

In Definition 8, the condition $M(h) + d \in \mathcal{F}(p_h, q_h)$ means that h can add d to the current assignment without violating any upper quota, and $M(h) + d - d' \in \mathcal{F}(p_h, q_h)$ means that h can replace d' with d without violating any upper or lower quota. The Classified Stable Matching, introduced by Huang [17], is the problem to decide the existence of a stable matching for a given CSM instance[1]. Because this is a generalization of HR-LQ, there are instances that have no stable matching. Let us consider envy-freeness for a CSM instance.

▶ **Definition 9.** For a matching M, a doctor d has **justified envy** toward d' with $M(d') = h$ if (i) d is unassigned or $h \succ_d M(d)$ and (ii) $M(h) + d - d' \in \mathcal{F}(p_h, q_h)$ and $d \succ_h d'$. A matching M is **envy-free** if no doctor has justified envy.

As in the case of HR-LQ, an envy-free matching can be regarded as a stable matching with reduced upper quotas as follows. For any $h \in H$ and $k \in \mathbf{Z}$ with $0 \leq k \leq q(A(h))$, a function $q'_h : 2^{A(h)} \to \mathbf{Z}$ is called a **k-truncation** of q_h if $q'(A(h)) = k$ and $q'(B) = q(B)$ for every $B \subsetneq A(h)$. Also, we simply say that q'_h is a **truncation** of q_h if there is such $k \in \mathbf{Z}$.

▶ **Proposition 10.** *For $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$, an assignment M is an envy-free matching if and only if M is a stable matching of $I' = (D, H, E, \succ_{DH}, \{(p_h, q'_h)\}_{h \in H})$ such that each q'_h is some truncation of q_h.*

Proof. To show the "only if" part, let M be an envy-free matching of I. For each $h \in H$, let q'_h be the $|M(h)|$-truncation of q_h. Then $M(h) \in \mathcal{F}(p_h, q'_h)$ and $M(h) + d \notin \mathcal{F}(p_h, q'_h)$ for every $d \in A(h) \setminus M(h)$. That is, M is feasible for I' and there is no doctor who claims a hospital's vacant seat. Therefore, if there is a blocking pair $(d, h) \in E \setminus M$ for I', it follows that d has a justified envy toward some d' with $M(d') = h$, which contradicts the envy-freeness of M. Thus, M is a stable matching of I'.

For the "if" part, let M be a stable matching of I'. Clearly, M is feasible for I. Suppose, to the contrary, some doctor d has justified envy toward d' with $M(d') = h$ with respect to I. Then d is unassigned or $h \succ_d M(d)$. Also, we have $d \succ_h d'$ and $M(h) + d - d' \in \mathcal{F}(p_h, q_h)$. Then, $M(h) + d - d' \in \mathcal{F}(p_h, q'_h)$ follows because $|M(h) + d - d'| = |M(h)|$. Hence, (d, h) is a blocking pair in I', a contradiction. ◀

We provide a hardness result for deciding the existence of an envy-free matching. Here, we assume that evaluation oracles of set functions p_h and q_h are available for each hospital h.

▶ **Theorem 11.** *It is NP-hard to decide whether a CSM instance $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$ has an envy-free matching or not. The problem is NP-complete even if the size of $\mathcal{C}(p_h, q_h)$ is at most 4 for each $h \in H$.*

[1] In his original model, each hospital h has a classification $\mathcal{C}_h \subseteq 2^{A(h)}$ and sets a lower and an upper quota for each member of $\mathcal{C}_h$. That is, we are provided $\mathcal{C}(p_h, q_h)$ and the values of p_h, q_h on it, rather than set functions p_h, q_h. Our formulation uses set functions to simplify the arguments in the next section.

Proof. We use reduction from the NP-complete problem (3, B2)-SAT [2], which is a restriction of SAT such that each clause contains exactly three literals and each variable occurs exactly twice as a positive literal and exactly twice as a negative literal. Let $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ be an instance of (3, B2)-SAT with Boolean variables $v_1, v_2, \dots, v_n$. Then, each clause c_j is a disjunction of three literals, (e.g., $c_j = v_1 \vee \neg v_2 \vee \neg v_3$) and each of literals v_i and $\neg v_i$ appears in exactly two clauses. For each variable v_i, denote by $j^*(i,1)$, $j^*(i,2)$ the indices of two clauses that contain v_i. Similarly, denote by $j^*(i,-1)$, $j^*(i,-2)$ the indices of two clauses that contain $\neg v_i$.

We now define a CSM instance corresponding to φ. We have a variable-hospital h_i for each variable v_i, and a clause-hospital h_j for each clause c_j. For each variable v_i, we have four doctors $\{\, d_{i,t} \mid t \in \{1,2,-1,-2\}\,\}$. For each doctor $d_{i,t}$, we have

$$A(d_{i,t}) = \{h_i, h_{j^*(i,t)}\}, \quad h_i \succ_{d_{i,t}} h_{j^*(i,t)}.$$

The set E is defined as the set of all pairs $(d_{i,t}, h)$ such that $h \in A(d_{i,t})$. Then, for each variable-hospital h_i and clause-hospital h_j, we have

$$A(h_i) = \{\, d_{i,t} \mid t \in \{1,2,-1,-2\}\,\},$$
$$A(h_j) = \{\, d_{i,t} \mid j^*(i,t) = j\,\}.$$

Note that $d_{i,t} \in A(h_j)$ implies $v_i \in c_j$ or $\neg v_i \in c_j$. Also, each of $v_i \in c_j$ and $\neg v_i \in c_j$ implies $d_{i,t} \in A(h_j)$ for some unique $t \in \{1,2,-1,-2\}$. Therefore, $|A(h_j)| = 3$ for each clause-hospital h_j. For each variable-hospital h_i, define p_{h_i} and q_{h_i} so that

$$\mathcal{C}(p_{h_i}, q_{h_i}) = \bigcup \{\, \{d_{i,t}, d_{i,t'}\} \mid t \in \{1,2\},\ t' \in \{-1,-2\}\,\},$$
$$p_{h_i}(\{d_{i,t}, d_{i,t'}\}) = q_{h_i}(\{d_{i,t}, d_{i,t'}\}) = 1 \qquad (t \in \{1,2\},\ t' \in \{-1,-2\}).$$

Then, we see that $\mathcal{F}(p_{h_i}, q_{h_i}) = \{D_i^+, D_i^-\}$, where $D_i^+ := \{d_{i,1}, d_{i,2}\}$ and $D_i^- := \{d_{i,-1}, d_{i,-2}\}$. For each clause-hospital h_j, define p_{h_i} and q_{h_i} so that

$$\mathcal{C}(p_{h_j}, q_{h_j}) = \{A(h_j)\}, \quad p_{h_j}(A(h_j)) = 1, \quad q_{h_j}(A(h_j)) = |A(h_j)| = 3.$$

We define preference lists of hospitals arbitrarily. Note that $|\mathcal{C}(p_h, q_h)| \leq 4$ for every hospital. We show that this CSM instance has an envy-free matching if and only if $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is satisfiable.

The "only if" part: Suppose that there is an envy-free matching M. Then, for every variable-hospital h_i, $M(h_i)$ is D_i^+ or D_i^-. For each h_i, set variable v_i to FALSE if $M(h_i) = D_i^+$, and to TRUE if $M(h_i) = D_i^-$. This Boolean assignment satisfies every clause c_j of φ as follows. Because $M(h_j) \in \mathcal{F}(p_{h_j}, q_{h_j})$, we have $|M(h_j)| \geq 1$. Hence, some $d_{i,t}$ with $j^*(i,t) = j$ is assigned to h_j. Then, $d_{i,t} \notin M(h_i)$. There are two cases: (i) $t \in \{1,2\}$, (ii) $t \in \{-1,-2\}$. In the case (i), $d_{i,t} \notin M(h_i)$ implies $M(h_i) \neq D_i^+$, and hence v_i is set to TRUE. Also, $t \in \{1,2\}$ and $j^*(i,t) = j$ imply $v_i \in c_j$. Hence, clause c_j is satisfied. Similarly, in the case (ii), we see that v_i is set to FALSE and we have $\neg v_j \in c_j$. Hence, clause c_j is satisfied.

The "if" part: Suppose that there is a Boolean assignment satisfying φ. Define an assignment M so that

- $M(h_i) = D_i^-$ if v_i is TRUE, and $M(h_i) = D_i^+$ if v_i is FALSE, and
- $M(h_j) = \{\, d_{i,t} \in A(h_j) \mid d_{i,t} \in D_i^+,\ v_i \text{ is } \mathsf{TRUE}\,\} \cup \{\, d_{i,t} \in A(h_j) \mid d_{i,t} \in D_i^-,\ v_i \text{ is } \mathsf{FALSE}\,\}$.

We can observe that $|M(d)| = 1$ for every doctor d, and $M(h_i) \in \mathcal{F}(p_{h_i}, q_{h_i})$ for every variable-hospital h_i. Also, because all clauses are satisfied, the above definition implies $M(h_j) \in \mathcal{F}(p_{h_j}, q_{h_j})$ for every clause-hospital h_j. Then, M is feasible. We now show the envy-freeness of M. Suppose, to the contrary, $d_{i,t}$ has justified envy toward d'. Because we have $|M(d_{i,t})| = 1$, $A(d_{i,t}) = \{h_i, h_{j^*(i,t)}\}$, and $h_i \succ_{d_{i,t}} h_{j^*(i,t)}$, this justified envy implies conditions $d' \in M(h_i)$, $d_{i,t} \notin M(h_i)$ and $M(h_i) + d_{i,t} - d' \in \mathcal{F}(p_{h_i}, q_{h_i})$. As $M(h_i) \in \mathcal{F}(p_{h_i}, q_{h_i}) = \{D_i^+, D_i^-\}$, then we have $\{M(h_i) + d_{i,t} - d', M(h_i)\} = \{D_i^+, D_i^-\}$, which contradicts $|D_i^+ \setminus D_i^-| = |D_i^- \setminus D_i^+| = 2$. ◀

4 Envy-freeness in CSM with Paramodular Quotas

In Section 3, we showed that it is NP-hard in general to decide whether a CSM instance has an envy-free matching or not. This section shows that the problem is solvable in polynomial time if the pair of quota functions is paramodular for each hospital. The proofs of the theorems and corollary in this section can be found in the full version. We first introduce the notion of paramodularity [9].

Let A be a finite set and let $p, q : 2^A \to \mathbf{Z}$. The pair (p, q) is **paramodular** (or, called a **strong pair** [10]) if

- p is **supermodular**, i.e., $p(B) + p(B') \leq p(B \cup B') + p(B \cap B')$ for every $B, B' \subseteq A$,
- q is **submodular**, i.e., $q(B) + q(B') \geq q(B \cup B') + q(B \cap B')$ for every $B, B' \subseteq A$, and
- the **cross-inequality** $q(B) - p(B') \geq q(B \setminus B') - p(B' \setminus B)$ holds for every $B, B' \subseteq A$.

Here we provide examples of constraints that can be represented by paramodular pairs. (See Yokoi [31, Appendices A and B].)

▶ **Example 12** (Laminar Constraints). Let $\mathcal{L} \subseteq 2^A$ be a laminar (or hierarchical) classification (i.e., any $X, Y \subseteq \mathcal{L}$ satisfy $X \subseteq Y$ or $X \supseteq Y$ or $X \cap Y \neq \emptyset$). Let $\hat{p}, \hat{q} : \mathcal{L} \to \mathbf{Z}$ be functions that define a lower and an upper quota for each class. Denote the acceptable set family by $\mathcal{J}(\mathcal{L}, \hat{p}, \hat{q}) := \{\, B \subseteq A \mid \forall X \in \mathcal{L} : \hat{p}(X) \leq |B \cap X| \leq \hat{q}(X) \,\}$. If $\mathcal{J}(\mathcal{L}, \hat{p}, \hat{q})$ is nonempty, then $\mathcal{J}(\mathcal{L}, \hat{p}, \hat{q}) = \mathcal{F}(p, q)$ for some paramodular pair (p, q).

▶ **Example 13** (Staffing Constraints). For a finite set S (e.g., a set of sections of a hospital), let $\Gamma : S \to 2^A$ and $\hat{l}, \hat{u} : S \to \mathbf{Z}$ be functions such that $\Gamma(s) \subseteq A$ represents members acceptable to $s \in S$ and $\hat{l}(s), \hat{u}(s) \in \mathbf{Z}$ represent a lower and an upper quota of each $s \in S$. Let $\mathcal{J}(S, \Gamma, \hat{l}, \hat{u}) \subseteq 2^A$ be a family of subsets $B \subseteq A$ such that there exists a function $\pi : B \to S$ satisfying $\forall d \in B : d \in \Gamma(\pi(d))$ and $\forall s \in S : \hat{l}(s) \leq |\{\, d \in B \mid \pi(d) = s \,\}| \leq \hat{u}(s)$. If $\mathcal{J}(S, \Gamma, \hat{l}, \hat{u})$ is nonempty, then $\mathcal{J}(S, \Gamma, \hat{l}, \hat{u}) = \mathcal{F}(p, q)$ for some paramodular pair (p, q).

For a set function $p : 2^A \to \mathbf{Z}$, its **complement** $\overline{p} : 2^A \to \mathbf{Z}$ is defined by

$$\overline{p}(B) = p(A) - p(A \setminus B) \quad (B \subseteq A).$$

Recall that a CSM instance is represented as a tuple $(D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$, where it is assumed that $0 \leq p_h(B) \leq q_h(B) \leq |B|$ for every $h \in H$ and $B \subseteq A(h)$. Here is the main theorem of this section. We denote by $\mathbf{0}$ a set function that is identically zero.

▶ **Theorem 14.** *For a CSM instance $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$, suppose that (p_h, q_h) is paramodular for each $h \in H$. Then, an instance $I' = (D, H, E, \succ_{DH}, \{(\mathbf{0}, \overline{p_h})\}_{h \in H})$ has at least one stable matching and the following three conditions are equivalent.*

(a) *I has an envy-free matching.*

(b) *Some stable matching M' of I' satisfies $|M'(h)| = p_h(A(h))$ for all $h \in H$.*

(c) *Every stable matching M' of I' satisfies $|M'(h)| = p_h(A(h))$ for all $h \in H$.*

Also, if (b) *holds, then M' is an envy-free matching of I.*

```
Algorithm 1: EF-Paramodular-CSM
  Input: $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$ such that each $(p_h, q_h)$ is paramodular
  Output: return an envy-free matching $M'$, or "there is no envy-free matching."

  Set $N_D \leftarrow E$, $N_H \leftarrow \emptyset$, and let $M'$ be undefined;
  while $M'$ is undefined do
      $R_D \leftarrow \bigcup_{d \in D} \{\, (d, h) \mid h \in N_D(d),\ h \neq \max_{\succ_d} N_D(d) \,\}$;
      $R_H \leftarrow \bigcup_{h \in H} \{\, (d, h) \mid d \in N_H(h),\ p(A(h) \setminus N_H(h)_{\succeq_h d}) = p(A(h) \setminus N_H(h)_{\succ_h d}) \,\}$;
      if $(N_D, N_H) = (E \setminus R_H, E \setminus R_D)$ then
          let $M' \leftarrow N_D \cap N_H$ and break;
      else
          update $(N_D, N_H) \leftarrow (E \setminus R_H, E \setminus R_D)$;
      end
  end
  if $|M'(h)| = p_h(A(h))$ for all $h \in H$ then
      return $M'$;
  else
      return "there is no envy-free matching";
  end
```

Here we sketch the proof of Theorem 14. See the full version for the detailed proof. The existence of a stable matching of I' and the equivalence between (b) and (c) can be shown by using Fleiner's results on the matroid framework [5, 6]. The most difficult part is showing the equivalence between conditions (a) and (b). To show that (a) implies (b), we construct a stable matching M' of I' from an envy-free matching M of I. This construction is achieved by using the fixed-point method of Fleiner [6]. The paramodularity of each (p_h, q_h) (or a generalized matroid structure of each $\mathcal{F}(p_h, q_h)$) is essential to show the existence of a fixed-point satisfying a required condition.

Theorem 14 implies that, when quota function pairs are paramodular, we can decide the existence of an envy-free matching of $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$ by the following algorithm.

Step1. Find a stable matching M' of $I' = (D, H, E, \succ_{DH}, \{(\mathbf{0}, \overline{p_h})\}_{h \in H})$.

Step2. If $|M'(h)| = p_h(A(h))$ for every $h \in H$, then return M'. Otherwise, return "there is no envy-free matching."

Step 1 (i.e., finding a stable matching of I') can be done by the generalized Gale-Shapley algorithm studied in [5, 6] (for the details see the full version). Then, the detailed description of the algorithm is given as follows. Here, for each $h \in H$, $N \subseteq E$, and $d \in N(h)$, we use notations $N(h)_{\succ_h d} := \{\, d' \in N(h) \mid d' \succ_h d \,\}$ and $N(h)_{\succeq_h d} := \{\, d' \in N(h) \mid d' \succ_h d \text{ or } d' = d \,\}$.

In the full version, we show that the assignment M' obtained in the above algorithm is indeed a stable matching of I'. Also, it is shown that N_D is monotone decreasing and N_H is monotone increasing in the algorithm, and hence the "while loop" is iterated at most $2|E|$ times. Thus, we obtain the following theorem.

▶ **Theorem 15.** *For a CSM instance $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h \in H})$ such that each (p_h, q_h) is paramodular, the algorithm* EF-Paramodular-CSM *decides whether I has an envy-free matching or not in $O(|E|^2)$ time, provided that evaluation oracles of $\{p_h\}_{h \in H}$ are available.*

As is shown in Examples 12 and 13, if the acceptable family of each hospital h is defined by a laminar constraint $\mathcal{J}_h := \mathcal{J}(\mathcal{L}_h, \hat{p}_h, \hat{q}_h)$ or by a staffing constraint $\mathcal{J}_h := \mathcal{J}(S_h, \Gamma_h, \hat{l}_h, \hat{u}_h)$, there is a paramodular pair (p_h, q_h) such that $\mathcal{J}_h = \mathcal{F}(p_h, q_h)$. The following corollary states that, in such a case, we can decide the existence of an envy-free matching of $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h\in H})$ even if evaluation oracles of $\{p_h\}_{h\in H}$ are not provided.

▶ **Corollary 16.** *Suppose that, for each $h \in H$, the family of acceptable doctor sets is defined in the form $\mathcal{J}_h := \mathcal{J}(\mathcal{L}_h, \hat{p}_h, \hat{q}_h) \neq \emptyset$ (resp., $\mathcal{J}_h := \mathcal{J}(S_h, \Gamma_h, \hat{l}_h, \hat{u}_h) \neq \emptyset$). Let (p_h, q_h) be a paramodular pair such that $\mathcal{J}_h = \mathcal{F}(p_h, q_h)$. Then, given $\mathcal{L}_h, \hat{p}_h, \hat{q}_h$ (resp., $S_h, \Gamma_h, \hat{l}_h, \hat{u}_h$) for each $h \in H$, one can decide whether $I = (D, H, E, \succ_{DH}, \{(p_h, q_h)\}_{h\in H})$ has an envy-free matching or not in time polynomial in $|E|$ (resp., in $|E|$ and $\max_{h\in H} |S_h|$).*

Proof. As we have Theorem 15, it completes the proof to show that we can simulate an evaluation oracle of each p_h in time polynomial in $|E|$ (resp., in $|E|$ and $|S_h|$). For a paramodular pair (p_h, q_h) with $\mathcal{J}_h = \mathcal{F}(p_h, q_h)$, it is known that, for any $B \subseteq A(h)$, we have $p_h(B) = \min\{\, |X \cap B| \mid X \in \mathcal{J}_h\}$ (see, e.g., [9, Theorem 14.2.8]). Consider a weight function w_B on $A(h)$ such that $w_B(d) = 1$ for every $d \in B$ and $w_B(d) = 0$ for every $d \in A(h) \setminus B$. Then, $p_h(B) = \min\{\, w_B(X) \mid X \in \mathcal{J}_h \,\}$, which is a weight minimization problem on $\mathcal{J}_h$. As shown in [31, Appendix C], if $\mathcal{J}_h$ is defined in the form in the statement, this problem can be reduced to the minimum cost circulation problem, which can be solved in strongly polynomial time [29, 23]. (See [31] for the details of the reduction.) Thus, the proof is completed. ◀

Acknowledgments. I wish to thank the anonymous reviewers whose comments have benefited the paper greatly.

References

1 A. Arulselvan, Á. Cseh, M. Groß, D. F. Manlove, and J. Matuschke. Matchings with lower quotas: Algorithms and complexity. *Algorithmica*, pages 1–24, 2016.

2 P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity Report*, (49), 2003.

3 P. Biró, T. Fleiner, R. W. Irving, and D. F. Manlove. The college admissions problem with lower and common quotas. *Theoretical Computer Science*, 411(34):3136–3153, 2010.

4 L. Ehlers, I. E. Hafalir, M. B. Yenmez, and M. A. Yildirim. School choice with controlled choice constraints: Hard bounds versus soft bounds. *Journal of Economic Theory*, 153:648–683, 2014.

5 T. Fleiner. A matroid generalization of the stable matching polytope. In *Proc. Eighth IPCO,* Lecture Notes in Computer Science **2081**, pages 105–114. Springer-Verlag, Berlin & Heidelberg, 2001.

6 T. Fleiner. A fixed-point approach to stable matchings and some applications. *Mathematics of Operations Research*, 28(1):103–126, 2003.

7 T. Fleiner and N. Kamiyama. A matroid approach to stable matchings with lower quotas. *Mathematics of Operations Research*, 41(2):734–744, 2016.

8 D. Fragiadakis, A. Iwasaki, P. Troyan, S. Ueda, and M. Yokoo. Strategyproof matching with minimum quotas. *ACM Transactions on Economics and Computation*, 4(1):6:1–6:40, 2015.

9 A. Frank. *Connections in Combinatorial Optimization,* Oxford Lecture Series in Mathematics and its Applications, 38. Oxford University Press, Oxford, 2011.

10 A. Frank and É. Tardos. Generalized polymatroids and submodular flows. *Mathematical Programming*, 42(1-3):489–563, 1988.

11 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

12 D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11(3):223–232, 1985.

13 M. Goto, A. Iwasaki, Y. Kawasaki, R. Kurata, Y. Yasuda, and M. Yokoo. Strategyproof matching with regional minimum and maximum quotas. *Artificial Intelligence*, 235:40–57, 2016.

14 D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT Press, Cambridge, MA, 1989.

15 K. Hamada, K. Iwama, and S. Miyazaki. The hospitals/residents problem with quota lower bounds. In *Proc. 19th ESA,* Lecture Notes in Computer Science **6942**, pages 180–191. Springer-Verlag, Berlin & Heidelberg, 2011.

16 K. Hamada, K. Iwama, and S. Miyazaki. The hospitals/residents problem with lower quotas. *Algorithmica*, 74(1):440–465, 2016.

17 C. C. Huang. Classified stable matching. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2010)*, pages 1235–1253. SIAM, Philadelphia, 2010.

18 Y. Kamada and F. Kojima. Efficient matching under distributional constraints: Theory and applications. *The American Economic Review*, 105(1):67–99, 2014.

19 Y. Kamada and F. Kojima. Stability concepts in matching under distributional constraints. *Journal of Economic Theory*, 168:107–142, 2017.

20 D. F. Manlove. *Algorithmics of Matching under Preferences.* World Scientific Publishing, Singapore, 2013.

21 M. Mnich and I. Schlotter. Stable marriage with covering constraints: A complete computational trichotomy. *arXiv preprint arXiv:1602.08230*, 2016.

22 K. Murota. Discrete convex analysis: A tool for economics and game theory. *Journal of Mechanism and Institution Design*, 1(1):151–273, 2016.

23 J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

24 A. E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *The Journal of Political Economy*, 92(6):991–1016, 1984.

25 A. E. Roth. Stability and polarization of interests in job matching. *Econometrica*, 52(1):47–57, 1984.

26 A. E. Roth. On the allocation of residents to rural hospitals: A general property of two-sided matching markets. *Econometrica*, 54(2):425–427, 1986.

27 A. E. Roth and M. A. O. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis.* Cambridge University Press, 1992.

28 É. Tardos. Generalized matroids and supermodular colourings. In *Matroid Theory* (L. Lovász and A. Recski, eds.), pages 359–382. North-Holland, Amsterdam, 1985.

29 É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

30 Q. Wu and A. E. Roth. The lattice of envy-free matchings. *Mimeo*, 2016.

31 Y. Yokoi. A generalized polymatroid approach to stable matchings with lower quotas. *Mathematics of Operations Research*, 42(1):238–255, 2017.