

18th Scandinavian Symposium and Workshops on Algorithm Theory

SWAT 2022, June 27–29, 2022, Tórshavn, Faroe Islands


Edited by

Artur Czumaj

Qin Xin



Editors

Artur Czumaj 

University of Warwick, UK
A.Czumaj@warwick.ac.uk

Qin Xin 

University of the Faroe Islands, Tórshavn, Faroe Islands
QinX@setur.fo

ACM Classification 2012

Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-236-5

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-236-5>.

Publication date

June, 2022

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 <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): <https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.SWAT.2022.0

ISBN 978-3-95977-236-5

ISSN 1868-8969

<https://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.

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

| | |
|---------------------------------------|------|
| Preface | |
| <i>Artur Czumaj and Qin Xin</i> | 0:ix |

Invited Papers

| | |
|---|----------|
| On Realizing a Single Degree Sequence by a Bipartite Graph | |
| <i>Amotz Bar-Noy, Toni Böhnlein, David Peleg, and Dror Rawitz</i> | 1:1–1:17 |
| Time, Clocks and Efficiency of Population Protocols | |
| <i>Leszek Gąsieniec and Grzegorz Stachowiak</i> | 2:1–2:2 |
| Reconstructing the Tree of Life (Fitting Distances by Tree Metrics) | |
| <i>Mikkel Thorup</i> | 3:1–3:2 |

Regular Papers

| | |
|---|------------|
| Compacting Squares: Input-Sensitive In-Place Reconfiguration of Sliding Squares | |
| <i>Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms</i> | 4:1–4:19 |
| Fault-Tolerant Edge-Disjoint s - t Paths – Beyond Uniform Faults | |
| <i>David Adjiashvili, Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt</i> | 5:1–5:19 |
| An Improved ε -Approximation Algorithm for Geometric Bipartite Matching | |
| <i>Pankaj K. Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle</i> | 6:1–6:20 |
| On the Visibility Graphs of Pseudo-Polygons: Recognition and Reconstruction | |
| <i>Safwa Ameer, Matt Gibson-Lopez, Erik Krohn, and Qing Wang</i> | 7:1–7:13 |
| Recognizing Map Graphs of Bounded Treewidth | |
| <i>Patrizio Angelini, Michael A. Bekos, Giordano Da Lozzo, Martin Gronemann, Fabrizio Montecchiani, and Alessandra Tappini</i> | 8:1–8:18 |
| A Novel Prediction Setup for Online Speed-Scaling | |
| <i>Antonios Antoniadis, Peyman Jabbarzade, and Golnoosh Shahkarami</i> | 9:1–9:20 |
| On the Approximability of the Traveling Salesman Problem with Line Neighborhoods | |
| <i>Antonios Antoniadis, Sándor Kisfaludi-Bak, Bundit Laekhanukit, and Daniel Vaz</i> | 10:1–10:21 |
| Dynamic Approximate Multiplicatively-Weighted Nearest Neighbors | |
| <i>Boris Aronov and Matthew J. Katz</i> | 11:1–11:14 |
| MaxSAT with Absolute Value Functions: A Parameterized Perspective | |
| <i>Max Bannach, Pamela Fleischmann, and Malte Skambath</i> | 12:1–12:20 |

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin



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

| | |
|---|------------|
| Dense Graph Partitioning on Sparse and Dense Graphs <i>Cristina Bazgan, Katrin Casel, and Pierre Cazals</i> | 13:1–13:15 |
| The Diameter of Caterpillar Associahedra <i>Benjamin Aram Berendsohn</i> | 14:1–14:12 |
| Stable Approximation Algorithms for the Dynamic Broadcast Range-Assignment Problem <i>Mark de Berg, Arpan Sadhukhan, and Frits Spijksma</i> | 15:1–15:21 |
| Well-Separation and Hyperplane Transversals in High Dimensions <i>Helena Bergold, Daniel Bertschinger, Nicolas Grelier, Wolfgang Mulzer, and Patrick Schneider</i> | 16:1–16:14 |
| Lions and Contamination: Monotone Clearings <i>Daniel Bertschinger, Meghana M. Reddy, and Enrico Mann</i> | 17:1–17:11 |
| Predecessor on the Ultra-Wide Word RAM <i>Philip Bille, Inge Li Gørtz, and Tord Stordalen</i> | 18:1–18:15 |
| An Optimal Algorithm for Product Structure in Planar Graphs <i>Prosenjit Bose, Pat Morin, and Saeed Odak</i> | 19:1–19:14 |
| Online Unit Profit Knapsack with Untrusted Predictions <i>Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen</i> | 20:1–20:17 |
| Nearest-Neighbor Decompositions of Drawings <i>Jonas Cleve, Nicolas Grelier, Kristin Knorr, Maarten Löffler, Wolfgang Mulzer, and Daniel Perz</i> | 21:1–21:16 |
| Approximation Metatheorems for Classes with Bounded Expansion <i>Zdeněk Dvořák</i> | 22:1–22:17 |
| Almost Shortest Paths with Near-Additive Error in Weighted Graphs <i>Michael Elkin, Yuval Gitlitz, and Ofer Neiman</i> | 23:1–23:22 |
| Complexity of Finding Maximum Locally Irregular Induced Subgraphs <i>Foivos Fioravantes, Nikolaos Melissinos, and Theofilos Triommatis</i> | 24:1–24:20 |
| An Almost Optimal Algorithm for Unbounded Search with Noisy Information <i>Junhao Gan, Anthony Wirth, and Xin Zhang</i> | 25:1–25:15 |
| Optimal Bounds for Weak Consistent Digital Rays in 2D <i>Matt Gibson-Lopez and Serge Zamarripa</i> | 26:1–26:20 |
| Matroid-Constrained Maximum Vertex Cover: Approximate Kernels and Streaming Algorithms <i>Chien-Chung Huang and François Sellier</i> | 27:1–27:15 |
| Non-Uniform k -Center and Greedy Clustering <i>Tanmay Inamdar and Kasturi Varadarajan</i> | 28:1–28:20 |
| Most Classic Problems Remain NP-Hard on Relative Neighborhood Graphs and Their Relatives <i>Pascal Kunz, Till Fluschnik, Rolf Niedermeier, and Malte Renken</i> | 29:1–29:19 |

| | |
|--|------------|
| A Scalable Work Function Algorithm for the k -Server Problem <i>Sharath Raghvendra and Rachita Sowle</i> | 30:1–30:20 |
| Erdős–Selfridge Theorem for Nonmonotone CNFs <i>Md Lutfar Rahman and Thomas Watson</i> | 31:1–31:11 |
| Unit-Disk Range Searching and Applications <i>Haitao Wang</i> | 32:1–32:17 |
| Space-Efficient Data Structure for Posets with Applications <i>Tatsuya Yanagita, Sankardeep Chakraborty, Kunihiko Sadakane, and Srinivasa Rao Satti</i> | 33:1–33:16 |

■ Preface

The *Scandinavian Symposium and Workshops on Algorithm Theory* (**SWAT**, formerly the Scandinavian Workshop on Algorithm Theory) has been held every two years beginning in 1988. It alternates with its sister conference, the Algorithms and Data Structures Symposium (WADS), which is usually hosted in Canada. This year marks the 18th SWAT hosted on Faroe Islands.

In response to the call for papers, a total of 90 submissions were received. Each submission was assigned to at least three Program Committee, aided by external subreviewers. The committees decided to accept 30 papers for inclusion in the scientific program. The selection was made by the Program Committees based on originality, quality, and relevance to the topic of the conference. The quality of the manuscripts was very high indeed, and many deserving papers could not be selected.

The Program Committee selected the following paper for the **best student paper award** (for a paper that is solely authored by students): Daniel Bertschinger, Meghana M. Reddy and Enrico Mann. *Lions and Contamination: Monotone Clearings*.

Apart from the contributed talks, SWAT 2022 included invited presentations by Leszek Gąsieniec (University of Liverpool), David Peleg (Weizmann Institute of Science), and Mikkel Thorup (University of Copenhagen). This volume contains all the contributed papers presented at the conference, papers that accompany the invited talk of David Peleg and abstracts of the invited presentations of Leszek Gąsieniec and Mikkel Thorup.

We would like to thank the program committee and the subreviewers for their great effort. For all of the papers, extensive and detailed evaluations were submitted. The program committee consisted of

- Amir Abboud (Weizmann Institute of Science)
- Mikkel Abrahamsen (University of Copenhagen)
- Peyman Afshani (Aarhus University)
- Yossi Azar (Tel Aviv University)
- Alkida Balliu (Gran Sasso Science Institute)
- Soheil Behnezhad (Stanford University)
- Radu Curticapean (ITU Copenhagen)
- Artur Czumaj (chair; University of Warwick)
- Andreas Emil Feldmann (Charles University, Prague)
- Sebastian Forster (University of Salzburg)
- Stefan Funke (University of Stuttgart)
- Petr Golovach (University of Bergen)
- Shaofeng Jiang (Peking University)
- Tomasz Kociumaka (UC Berkeley)
- Amit Kumar (Indian Institute of Technology Delhi)
- Anil Maheshwari (Carleton University)
- Dániel Marx (CISPA Helmholtz Center for Information Security)
- Kitty Meeks (University of Glasgow)
- Slobodan Mitrović (University of California, Davis)
- Pan Peng (University of Science and Technology of China (USTC), Hefei, Anhui)
- Valentin Polishchuk (Linköping University)
- Nicola Prezza (Ca' Foscari University of Venice)
- Chris Schwiegelshohn (Aarhus University)

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).
Editors: Artur Czumaj and Qin Xin



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

- Shay Solomon (Tel Aviv University)
- Uli Wagner (IST Austria)
- Meirav Zehavi (Ben-Gurion University of the Negev)
- Rico Zenklusen (ETH Zurich)

This year's conference was organized by Qin Xin (University of the Faroe Islands) and his team consisting of Hans Blaasvør (financial chair; KT Húsið and University of the Faroe Islands), Olavur Ellefsen (Globe Tracker and University of the Faroe Islands), Jogvan Thomsen (Vinnustovnurin), and Michael Thomsen (Vinnustovnurin and University of the Faroe Islands).

The SWAT conference series is guided by a steering committee consisting of

- Lars Arge† (Aarhus University)
- Magnús M. Halldórsson (chair; Reykjavík University)
- Andrzej Lingas (Lund University)
- Jan Arne Telle (University of Bergen)
- Esko Ukkonen (University of Helsinki)

We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their scholarly effort, and all the referees who assisted the Program Committees in the evaluation process. We are also grateful to all the support staff of the Organizing Committee for organizing SWAT 2022.

We would like to thank Magnús Halldórsson, the Chair of the SWAT Steering Committee, for his continuous support.

May 2022

Artur Czumaj
Qin Xin

On Realizing a Single Degree Sequence by a Bipartite Graph

Amotz Bar-Noy ✉

City University of New York (CUNY), NY, USA

Toni Böhnlein ✉

Weizmann Institute of Science, Rehovot, Israel

David Peleg ✉ 

Weizmann Institute of Science, Rehovot, Israel

Dror Rawitz ✉

Bar Ilan University, Ramat-Gan, Israel

Abstract

This paper addresses the classical problem of characterizing degree sequences that can be realized by a bipartite graph. For the simpler variant of the problem, where a partition of the sequence into the two sides of the bipartite graph is given as part of the input, a complete characterization was given by Gale and Ryser over 60 years ago. However, the general question, in which both the partition and the realizing graph need to be determined, is still open. This paper provides an overview of some of the known results on this problem in interesting special cases, including realizations by bipartite graphs and bipartite multigraphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Degree Sequences, Graph Realization, Bipartite Graphs, Graphic Sequences, Bigraphic Sequences, Multigraph Realization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.1

Category Invited Paper

Funding This work was supported by US-Israel BSF grant 2018043.

1 Introduction

1.1 Background and Motivation

A sequence $d = (d_1, \dots, d_n)$ of nonnegative integers is *graphic* if there exists an n -vertex graph G whose degree sequence $\deg(G)$ satisfies $\deg(G) = d$. The question of recognizing graphic degree sequences was studied extensively in the past six decades. Given a sequence d , the *graphic degree realization (GDR)* problem requires deciding whether d is graphic and constructing a graph G realizing it, if one exists. A complete characterization (implying also an $\mathcal{O}(n)$ time decision algorithm) for graphic degree sequences was given by Erdős and Gallai [16]. An algorithm that, given a sequence d , generates a realizing graph or proves that the sequence is not graphic, was given by Havel and Hakimi [19, 22]. This algorithm runs in time $\mathcal{O}(\sum_i d_i)$, which is optimal¹.

A natural variant of the graphic degree realization problem requires the realizing graph to be *bipartite*. A sequence admitting a bipartite realizing graph is called *bigraphic*, and the corresponding problem is called the *bigraphic degree realization (BDR)* problem. This problem has appeared as an open problem over 40 years ago [33], but did not receive a lot of attention.

¹ Note that $\sum_i d_i = 2m$ where m is the number of edges in a realizing graph if exists.



In contrast, the simpler variant where the partition of d is given as part of the input was studied extensively. Here, the input consists of a partition of d into two sequences, $a = (a_1, \dots, a_p)$ and $b = (b_1, \dots, b_q)$, and it is required to decide whether there exists a bipartite graph $G(A, B, E)$ such that $|A| = p$, $|B| = q$, and the sequences of degrees of the vertices of A and B are equal to a and b , respectively. Hereafter, we refer to such a pair (a, b) as a *bigraphic degree partition*, and to the problem as the *given partition* version of the bigraphic degree realization problem, BDR^P .

Necessary and sufficient conditions for a pair of sequences (a, b) to be a bigraphic degree partition were given in 1957 by Gale and Ryser [17, 34]. These conditions yield also a polynomial time decision algorithm for BDR^P , which can be thought of as a variant of the Havel-Hakimi algorithm for general graphs applied to one side of the partition.

An obvious question is whether the Gale-Ryser conditions can be used for attacking the (single sequence) bigraphic degree realization problem BDR. One natural strategy is to search for a bigraphic degree partition for the given sequence d , relying on the fact that $d_1 < n$ is a necessary condition for a sequence d to be graphic (or bigraphic), and for such d , a partition can be found (if one exists) in polynomial time, since the PARTITION problem is pseudo-polynomial (cf. [5, 13]). Unfortunately, it is possible that some partitions of d are bigraphic while other partitions are not (see Example 1 in Sect. 5). Moreover, the number of different partitions for a given sequence may be exponential in its length (see Examples 2 & 3 in Sect. 5). Still, one may hope that the special structure required by a bigraphic degree partition may assist us in searching for them. Unfortunately, so far we have not been able to fully characterize the class of bigraphic degree sequences, or to determine whether the problem is *NP*-hard. In this paper we report what we perceive to be some of the more interesting findings on the problem.

1.2 Results

We present two types of results. We first identify special instances for which one can solve the BDR problem, i.e., decide whether a given sequence is bigraphic or not and if so, generate a realizing graph. Second, we describe realizations by bipartite *multigraphs* (namely, graphs that allow parallel edges) for special instances where the BDR problem is decided in the negative or is unsolved. The multigraph realizations are generated with the objective of minimizing the *maximum* multiplicity in order to come close to resolving the bipartite realization problem, i.e., finding *approximate* realizations.

The notation of graphic and bigraphic sequences is extended to handle multigraphs. A sequence d of non-negative integers is said to be *t-graphic* (*t-bigraphic*) if it admits a (bipartite) multigraph realizations with maximum multiplicity of at most t parallel edges. If a bipartite multigraph realization is based on a partition (a, b) , we say that partition (a, b) is *t-bigraphic*.

In the following, we classify the known results into several categories depending on the type of instances that are being considered.

Small Instances. The first category of instances concerns cases where the BDR problem can be resolved exactly due to the fact that the instance is “small” in some sense. In Section 3, we focus on two such cases. The first is when the given sequence d admits only a small number of partitions $N_{Part}(d)$. Formally, it is required that $N_{Part}(d) = \mathcal{O}(n^c)$ for some constant c . For such sequences, it is possible to exploit the fact that the PARTITION problem is pseudo-polynomial. To do that, we use an *output-sensitive* algorithm for generating all the partitions of d , namely, an algorithm requiring time $\mathcal{O}(n^{c'})$ per partition for some constant

c' . A special subcase of this case involves sequences with a constant number of distinct degrees, since a sequence with only a constant number of different degrees can have at most polynomially many different partitions.

The second case of “small” instances concerns sequences whose maximum degree is small. Specifically, we show that for every partitionable nonincreasing n -integer sequence $d = (d_1, \dots, d_n)$, if $d_1^2 \leq t \cdot \sum_i d_i/2$, then d is a t -bigraphic degree sequence, and moreover, any partition (a, b) of d is a t -bigraphic degree partition. An alternative (weaker) condition on d_1 is that $d_1^2 \leq t \cdot n/2$.

High-Low Partitions. We then shift our attention to specific and significant types of partitions, referred to as *High-Low partitions*. A High-Low partition of a non-increasing sequence d has the form $HL(d) = (H, L)$ where $H = (d_1, \dots, d_k)$ and $L = (d_{k+1}, \dots, d_n)$ for some k . Clearly, this pair (H, L) is a balanced partition only if $\sum_{i=1}^k d_i = \sum_{i=k+1}^n d_i$.

For High-Low partitions, the first Gale-Ryser conditions are key to the realizability of the sequence. These conditions state that the largest degree on each side does not exceed the number of vertices on the other side (formally, $d_1 \leq n - k$ and $d_{k+1} \leq k$). A (balanced) High-Low partition (H, L) that satisfies the first Gale-Ryser conditions is referred to as a *well-behaved* High-Low partition.

The fact that a High-Low partition is well-behaved does not guarantee that it is bigraphic (see Example 4 in Sect. 5). However, as described in Sect. 4, for a non-increasing sequence d that admits a well-behaved High-Low partition (H, L) , the BDR problem turns out to be solvable [4]. This follows from the fact that for a sequence d admitting a well-behaved High-Low partition (H, L) , if (H, L) is not bigraphic, then *no* partition of d is bigraphic, hence d itself is not bigraphic. It follows that if d has a well-behaved High-Low partition, then it can be decided in polynomial time whether d is bigraphic or not. Moreover, if d happens to be bigraphic, a bipartite graph realizing d can be computed in polynomial time (e.g., using the adapted Havel-Hakimi algorithm described in Section 2).

In Section 4 we also discuss bipartite multigraph realizations based on High-Low partitions. It turns out that even in case a well-behaved High-Low partition fails to be bigraphic, it is still *2-bigraphic*. More generally, defining a parameter $t(d)$ indicating the extent to which $HL(d)$ violates the first Gale-Ryser conditions, we have the following: If an r -graphic sequence d admits a High-Low partition $HL(d)$, then $HL(d)$ is t -bigraphic where $t = \max\{t(d), 2r\}$.

Equal Partitions. We explore another specific and important type of partitions, referred to as *equal partitions*. An n -integer sequence d admits an equal partition if d is *even*, namely, each integer occurring in d appears in it an even number of times. At the cost of a slight notational inconsistency, resolved by context, we adopt the compact notation $d = (d_1^{n_1}, \dots, d_q^{n_q})$, where $\sum_{i=1}^q n_i = n$, for a sequence consisting of n_i copies of the integer d_i for $1 \leq i \leq q$. Then the sequence d is even if n_i is even for every $1 \leq i \leq q$. For such a sequence d , the *equal partition* is $EQP(d) = (a, b)$ where $a = b = (d_1^{n_1/2}, \dots, d_q^{n_q/2})$.

The equal partition does not display a property similar to that of well-behaved High-Low partitions, i.e., a “well-behaved” equal partition does not allow us to resolve the BDR problem. Specifically, there are even sequences d with a well-behaved equal partition $EQP(d) = (a, b)$ such that (a, b) is not bigraphic but other partitions of d are bigraphic (see Example 5 in Sect. 5). However, as shown in [3], if the sequence d is graphic and even, then the equal partition is 2-bigraphic. More generally, for multigraph realizations with bounded maximum multiplicity, the following holds. Let d be an even and r -graphic degree sequence with equal partition $EQP(d) = (a, b)$. Then (a, b) is $2r$ -bigraphic.

High-Low vs. Equal Partitions. In some sense, the High-Low partition and the equal partition are two extremes: whereas the High-Low partition tries to *differentiate* the two sides as much as possible, taking all the largest elements to one side and all the smallest elements to the other, the equal partition attempts to *equalize* the two parts as much as possible.

Interestingly, there are bigraphic even sequences for which the High-Low partition is bigraphic while the equal partition is not, or vice versa (see Examples 5 and 6 in Sect. 5). One might speculate that if d is bigraphic and has both a High-Low partition and an equal partition, then at least one of them must be bigraphic, but even that turns out to be false (see Example 1 in Sect. 5).

1.3 Related Work

The two key questions on degree sequences studied in the literature concern identifying necessary and sufficient conditions for a sequence to be graphic, and developing efficient algorithms for computing a realizing graph if exists. As mentioned above, Erdős and Gallai [16] are the first to present a characterization of graphic sequences (several alternative proofs exist, see [10, 2, 40, 14, 38, 39, 47, 26].) Havel [22] and Hakimi [19] provide a different characterization, also, implying an algorithm to construct a realizing graph.

Several related questions are considered in the literature: Given a degree sequence d , (1.) find all the (non-isomorphic) graphs that realize it. (2.) count all its (non-isomorphic) realizing graphs. (3.) sample a random realization as uniformly as possible. These questions are extensively studied, see [10, 16, 19, 22, 24, 36, 38, 44, 45, 46]. Applications to network design, randomized algorithms, social networks [6, 12, 15, 29] and chemical networks [37] exists. Miller [30] shows that only a subset of the Erdős and Gallai inequalities needs to be checked in order to decide if a degree sequence is graphic. The literature also includes surveys on degree sequences, see [41, 42, 43].

Additional intriguing directions include finding characterizations for degree sequences of specific graph families. To that end, we call a degree sequence *potentially P -graphic* if it has a realizing graph having the graph theoretic property P . Rao [33] surveys results (see references therein) on various properties like k -edge connected, k -vertex connected, hamiltonian and tournament. As an open problem characterizing potentially bipartite sequences is mentioned, i.e., the BDR problem.

Moreover, a characterization is known for trees (cf. [18]). The family of *planar* graphs was studied to some extent. The existing results provide a characterization for planar graphic k -sequences, where the difference between the largest and the smallest degree is bounded by k , for $k = 0, 1, 2$ [1, 35]. Full characterizations for the degree sequences of *threshold* graphs (see [20]), *split* graphs (see [21]), *matrogenic* graphs (see [28]) and *difference* graphs (see [20]) are known. Degree sequences of *chordal*, *interval*, and *perfect* graphs are considered in [9].

As mentioned above, the BDR^P was solved in [17, 34] (for an alternative proof see [27]). As the title “Combinatorial properties of matrices of zeros and ones” suggests, the problem motivating Ryser [34] has, naturally, a pair of sequences as its input. Sufficient conditions for a pair of sequences to be bigraphic were studied in [7, 48].

Owens and Trent [31] were interested in the realization problem for multigraphs. Given a degree sequence, their results provide a multigraph realizations minimizing the total number of parallel edges or loops (improved algorithms are presented in [32, 25]). The opposite objective of maximizing the total number of parallel edges is, however, proven to be NP -hard (see [23]).

2 Preliminaries

Let $H = (V, E)$ be a multigraph without loops. In this case, E is a multiset. Denote by $E_H(v, w)$ the multiset of edges connecting $v, w \in V$. The *maximum multiplicity* of H is

$$\text{MaxMult}(H) = \max_{(v,w) \in E} (|E_H(v, w)|).$$

2.1 Degree Sequences of Graphs and Multigraphs

Let $d = (d_1, d_2, \dots, d_n)$ be a sequence of nonnegative integers in nonincreasing order². The *volume* of d is $\sum d = \sum_{i=1}^n d_i$. Note that every graphic sequence must have even volume. We call a sequence with even volume a *degree sequence*.

The characterization of Erdős and Gallai [16] for graphic degree sequences is as follows.

► **Theorem 1** (Erdős-Gallai [16]). *A degree sequence $d = (d_1, d_2, \dots, d_n)$ is graphic if and only if*

$$\sum_{i=1}^{\ell} d_i \leq \ell(\ell - 1) + \sum_{i=\ell+1}^n \min\{\ell, d_i\}, \quad (1)$$

for $\ell = 1, \dots, n$.

We call Equation (1) the ℓ -th *Erdős-Gallai inequality* EG_ℓ . Theorem 1 implies an $\mathcal{O}(n)$ algorithm to verify whether a sequence is graphic.

Let r be a positive integer. Then, a degree sequence d is *r -graphic* if there exists a multigraph H such that $\deg(H) = d$ and $\text{MaxMult}(H) \leq r$. A characterization for r -graphic sequences was shown by Chungphaisan [11].

► **Theorem 2** (Chungphaisan [11]). *Let r be a positive integer. Degree sequence $d = (d_1, d_2, \dots, d_n)$ is r -graphic if and only if*

$$\sum_{i=1}^{\ell} d_i \leq r\ell(\ell - 1) + \sum_{i=\ell+1}^n \min\{r\ell, d_i\}, \quad (2)$$

for $\ell = 1, \dots, n$.

The minimum r such that a sequence d is r -graphic can be computed in polynomial time.

2.2 Degree Sequences of Bipartite Graphs and Multigraphs

Let d be a degree sequence such that $\sum d = 2m$ for some integer m . A *block* of d is a subsequence a such that $\sum a = m$. The set of all blocks of sequence d is defined as $B(d) := \{a \subset d \mid \sum a = m\}$. For each $a \in B(d)$ there is a disjoint $b \in B(d)$ that completes it to form a partition of d (so that merging them in sorted order yields d). We call such a pair $a, b \in B(d)$ a (*balanced*) *partition* of d since $\sum a = \sum b$. Denote the set of all degree partitions of d by $\text{BP}(d) = \{\{a, b\} \mid a, b \in B(d), d \setminus a = b\}$.

The Gale-Ryser theorem characterizes bigraphic degree partitions.

² All sequence that we consider are assumed to be in a non-increasing order.

► **Theorem 3** (Gale-Ryser [17, 34]). *Let d be a degree sequence and partition $(a, b) \in \text{BP}(d)$ where $a = (a_1, a_2, \dots, a_p)$ and $b = (b_1, b_2, \dots, b_q)$. The partition (a, b) is bigraphic if and only if*

$$\sum_{i=1}^{\ell} a_i \leq \sum_{i=1}^q \min\{\ell, b_i\} \tag{3}$$

for $\ell = 1, \dots, p$.

We refer to Equation (3) as the ℓ -th Gale-Ryser inequality GR_{ℓ}^L on the left. By symmetry, the partition (a, b) is bigraphic if and only if $\sum_{i=1}^{\ell} b_i \leq \sum_{i=1}^p \min\{\ell, a_i\}$, for $\ell = 1, \dots, q$. We refer to this equation as the ℓ -th Gale-Ryser inequality GR_{ℓ}^R on the right.

Let t be a positive integer. A degree sequence d is t -bigraphic if d has a partition $(a, b) \in \text{BP}(d)$ such that there is a bipartite multigraph $H = (A, B, E)$ such that $\text{MaxMult}(H) \leq t$, $|A| = |a|$, $|B| = |b|$, and the sequences of degrees of the vertices of A and B are equal to a and b , respectively. We also say that partition (a, b) is t -bigraphic. Miller [30] cites the following result of Berge characterizing t -bigraphic partitions.

► **Theorem 4** (Berge [30]). *Consider a positive integer t , a degree sequence d and a partition $(a, b) \in \text{BP}(d)$ where $a = (a_1, \dots, a_p)$ and $b = (b_1, \dots, b_q)$. The partition (a, b) is t -bigraphic if and only if*

$$\sum_{i=1}^{\ell} a_i \leq \sum_{i=1}^q \min\{\ell t, b_i\}, \tag{4}$$

for $\ell = 1, \dots, p$.

2.3 Havel-Hakimi Algorithm for Bipartite Graphs

Kleitman and Wang [26] generalize the Havel-Hakimi theorem implying an algorithm where the 'pivot' can be chosen freely. It is folklore that the same approach can be extended to bigraphic sequences and a given partition. In the following, we formalize this result and start with introducing some notation. Let d be a degree sequence and $(a, b) \in \text{BP}(d)$ where $a = (a_1, \dots, a_p)$ and $b = (b_1, \dots, b_q)$. We assume that partition (a, b) satisfies the first Gale-Ryser conditions, i.e., $a_1 \leq q$ and $b_1 \leq p$ hold. Let $i \leq p$ be some index. Define

$$a^{-i} = a \setminus a_i = (a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_p),$$

and

$$\text{RED}(b, a_i) = (b_1 - 1, \dots, b_{a_i} - 1, b_{a_i+1}, \dots, b_q).$$

Moreover, let $G = (A, B, E)$ be a bipartite graph realizing (a, b) . For a positive integer $\ell \leq |B|$, define the subset $\text{MaxDeg}(B, \ell) \subseteq B$ to contain vertices with degrees b_1, \dots, b_{ℓ} . Ties are broken arbitrarily to ensure that $|\text{MaxDeg}(B, \ell)| = \ell$.

► **Lemma 5.** *Let d be a degree sequence and $(a, b) \in \text{BP}(d)$. Let $i \leq |a|$ be some index. If (a, b) is bigraphic, then there is a bipartite graph $G = (A, B, E)$ where vertex $v \in A$ has degree a_i and is adjacent to each vertex in $\text{MaxDeg}(B, a_i)$.*

Proof. Let d , (a, b) , and index i as in the theorem. Assume that (a, b) is bigraphic, and let $G = (A, B, E)$ be a bipartite graph realizing (a, b) where vertex $v \in A$ has degree a_i . Denote $B' = \text{MaxDeg}(B, a_i)$. If v is adjacent to each vertex in B' , we are done. Otherwise there are

vertices $u \in B'$ and $w \in B \setminus B'$ such that $(v, u) \notin E$ and $(v, w) \in E$. By definition of B' , we have that $\deg(u) \geq \deg(w)$. It follows that there is a vertex $v' \in A$ such that $(v', u) \in E$ and $(v', w) \notin E$. Now, we construct a graph $G' = (A, B, E')$ by applying an edge flip operation to G such that G' is bipartite, realizes (a, b) and v is adjacent to one more vertex in B' (comparing G and G'). For the edge flip operation, remove edges $(v, w), (v', u) \in E$ and add edges $(v, u), (v', w)$ to E' . Verify that G' satisfies the claimed properties. The operation can be applied until v is adjacent to each vertex in B' , and the lemma is shown. \blacktriangleleft

► Theorem 6. *Let d be a degree sequence and $(a, b) \in \text{BP}(d)$. Also, let $i \leq |a|$ be some index, $a' = a^{-i}$, and $b' = \text{RED}(b, a_i)$. Then, (a, b) is bigraphic if and only if (a', b') is bigraphic.*

Proof. Let $d, (a, b)$, index i , and (a', b') be as in the theorem.

First, assume that (a, b) is bigraphic. Due to Lemma 5 there is a bipartite graph $G = (A, B, E)$ realizing (a, b) where vertex $v \in A$ has degree a_i and v is adjacent to vertices $u_1, \dots, u_{a_i} \in B$ having degrees b_1, \dots, b_{a_i} . Let G' be the result of removing vertex v and its incident edges from graph G . Verify that G' realizes (a', b') , and consequently (a', b') is bigraphic.

Now, assume that (a', b') is bigraphic. Let $G = (A, B, E)$ be a bipartite graph realizing (a', b') . Construct a graph $G' = (A', B, E')$ from G by adding a new vertex v to A , i.e., $A' = A \cup v$. Next, connect vertex v to vertices $u_1, \dots, u_{a_i} \in B$ having degrees $b_1 - 1, b_2 - 1, \dots, b_{a_i} - 1$. Verify that G' is bipartite, vertex v has degree a_i , and that G' realizes (a, b) . It follows that (a, b) is bigraphic. \blacktriangleleft

Similar to the Havel-Hakimi [19, 22] characterization for graphic degree sequences, Theorem 6 implies a polynomial time algorithm that, given a partition (a, b) , constructs a bipartite graph realizing (a, b) or decides that (a, b) is not bigraphic.

3 Small Instances

3.1 Output-Sensitive Algorithms: Small Number of Partitions

Typically, the running time of an algorithm is measured as a function of the input size. However, in situations where the output may be very long, it is of interest to bound the time complexity also as a function of the output size. More explicitly, we say that A is a polynomial time *output-sensitive* algorithm for a problem Π if for every input I of length $\ell(I)$, whose output $\Pi(I)$ is of length $\ell(\Pi(I))$, A computes $\Pi(I)$ in time polynomial in $\max\{\ell(I), \ell(\Pi(I))\}$.

This notion may be useful in the context of the bigraphic degree realization problem. For a degree sequence d , let $N_{\text{Part}}(d) = |\text{BP}(d)|$ denote the number of different block partitions of d . The BDR problem can be solved on d by enumerating all possible partitions and applying Theorem 3 for each of them. This may lead to a polynomial time output-sensitive algorithm for the problem provided that the enumeration procedure requires $\mathcal{O}(n^c)$ time, for some constant c , to generate the next partition.

We design an *output-sensitive* algorithm for BDR based on the natural enumeration procedure for block partitions. Recall that given an integer sequence $d = (d_1, d_2, \dots, d_n)$, such that $d_i < n$ for every i , it is possible to find a block partition of d in polynomial time, by using the pseudo-polynomial dynamic programming algorithm for the partition problem, which in this case becomes polynomial. We claim that it is possible to find *all* block partitions of d in time $\mathcal{O}(T_{\text{Part}}(n) \cdot n \cdot N_{\text{Part}}(d))$, where $T_{\text{Part}}(n)$ is the time complexity of the best pseudo-polynomial time algorithm for deciding the PARTITION problem. To establish this, we describe a straightforward recursive algorithm. The algorithm uses a procedure $\text{DP}(d, A)$ that receives as input a sequence

1:8 On Realizing a Single Degree Sequence by a Bipartite Graph

$$d = (d_1^{n_1}, d_2^{n_2}, \dots, d_q^{n_q}) \quad (5)$$

of $n = n_1 + \dots + n_q$ integers and a subsequence

$$A = (d_1^{j_1}, \dots, d_k^{j_k}) \quad (6)$$

of d , where $0 \leq k \leq q$ and $0 \leq j_i \leq n_i$ for $i = 1, \dots, k$. Setting

$$B = B(A, d) = (d_1^{n_1 - j_1}, \dots, d_k^{n_k - j_k}), \quad (7)$$

the procedure decides, in time polynomial in n , if $A \parallel B$ can be completed into a partition (A', B') of d (namely, such that $A' \parallel B' = d$, $\sum A' = \sum B' = \sum d/2$, $A \subseteq A'$, $B \subseteq B'$).

Our recursive algorithm, $\text{ALG}(d, A, j)$, gets as input a sequence d and a subsequence A as in Eq. (5) and (6). and $0 \leq j \leq n_k$. Setting $B = B(A, d)$ as in Eq. (7), the algorithm returns a set \mathcal{P} containing all partitions (A', B') of d such that $A' \parallel B' = d$, $\sum A' = \sum B' = \sum d/2$, $A \circ (d_{k+1}^j) \subseteq A'$, $B \circ (d_{k+1}^{n_{k+1} - j}) \subseteq B'$. The set of all partitions of d is obtained by invoking $\text{ALG}(d, \emptyset, 0)$ (thinking of d as augmented with an “empty prefix” $d_0^{n_0} = 0^0$).

■ **Algorithm 1** Algorithm $\text{ALG}(d, A)$.

Input: d, A as in Eq. (5) and (6).

Set $B \leftarrow B(d, A)$ as in Eq. (7).

1. Set $\mathcal{P} \leftarrow \emptyset$.
 2. Invoke Procedure $\text{DP}(d, A)$.
 3. If the procedure returned “YES” then do:
 - a. If $k = q$ then set $\mathcal{P} \leftarrow \{(A, B)\}$.
 - b. Else (* $k < q$ *)
 - repeat** for $j_{k+1} = 0$ to n_{k+1} :
 - i. Set $A' \leftarrow A \circ (d_{k+1}^{j_{k+1}})$
Set $B' \leftarrow B \circ (d_{k+1}^{n_{k+1} - j_{k+1}})$
 - ii. Recursively invoke Algorithm $\text{ALG}(d, A')$, which returns $\mathcal{P}_{j_{k+1}}$.
 - iii. Set $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_{j_{k+1}}$.
 4. Return \mathcal{P} .
-

► **Observation 7.** *The algorithm returns all partitions of d .*

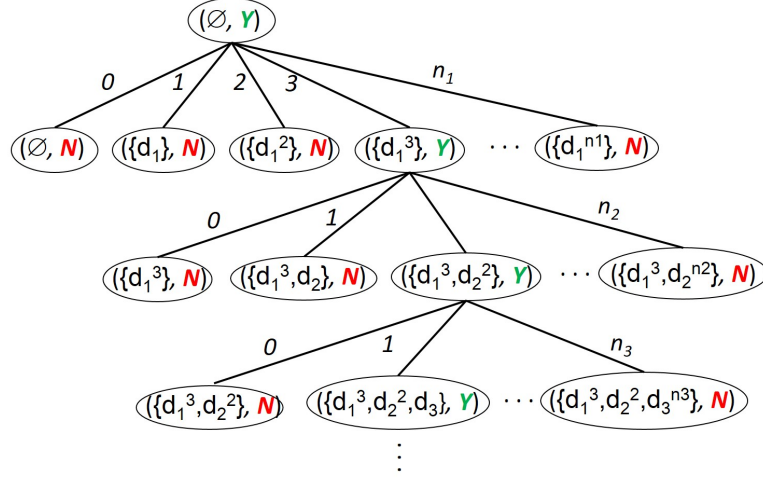
► **Observation 8.** *The algorithm runs in time $\mathcal{O}(N_{\text{Part}}(d) \cdot n \cdot T_{\text{DP}})$.*

Note that the complexity of Algorithm ALG is dominated by the total time spent on the invocations of Procedure DP. Therefore, to prove Obs. 8, we need to show that when executing Algorithm ALG on a sequence d , the number of invocations of Procedure DP, K_{DP} , satisfies

$$K_{\text{DP}} \leq \mathcal{O}(N_{\text{Part}}(d) \cdot n) \quad (8)$$

To see this, let us illustrate the recursive execution of the algorithm on d by an *execution tracing tree* T_{EX} consisting of $q + 1$ levels. Each node in the tree is labeled by a pair (A, R) , where $R \in \{Y, N\}$, and corresponds to one invocation of Procedure DP. The first entry in the label corresponds to the parameter A in the invocation, i.e., the root of the tree is

marked (\emptyset, R) , and all nodes on level $1 \leq k \leq q$ are marked (A, R) for some subsequence A of d as in Eq. (6). The path leading to a node (A, R) in the tree captures the set A in the corresponding invocation. $R = N$ indicates that there are no partitions of d matching A and $B = B(A, d)$, whereas $R = Y$ indicates that there is at least one partition of d matching A and B . See Figure 1.



■ **Figure 1** Illustration of the execution tracing tree T_{EX} of algorithm ALG, in the case where d has only one partition.

Note that in this tree, every node labeled (A, N) is a leaf, as the algorithm does not perform any additional recursive calls for this subsequence A . We refer to leaves labeled (A, N) (respectively, (A, Y)) as *N-leaves* (resp., *Y-leaves*), and denote their number by L_N (resp., L_Y). Also, denote the number of internal nodes on level k of the tree (which are also labeled by (A, Y)) by $I(k)$.

Observe that K_{DP} , the number of invocations of Procedure DP during the execution of Algorithm ALG, equals the number of nodes in T_{EX} , i.e.,

$$K_{DP} = |V(T_{EX})| = L_Y + L_N + \sum_{k=0}^{q-1} I(k). \quad (9)$$

To bound this number, we redistribute the “charge” for the invocations of Procedure DP as follows. For $0 \leq k \leq q-1$, we reassign the charge for the invocation at an *N-leaf* v on level $k+1$ to its parent w on level k . Note that w can be charged by at most n_{k+1} *N-leaves*, since it has $n_{k+1} + 1$ children and at least one of them is marked *Y*. It follows that

$$K_{DP} \leq L_Y + \sum_{k=0}^{q-1} I(k) \cdot (n_{k+1} + 1). \quad (10)$$

Observe that $I(k) \leq L_Y$ for every $0 \leq k \leq q-1$, since every internal node has at least one child labeled (A, Y) . It follows that

$$K_{DP} \leq L_Y + L_Y \cdot \sum_{k=0}^{q-1} (n_{k+1} + 1) = L_Y + L_Y \cdot (n + k). \quad (11)$$

Observe also that $L_Y = N_{Part}(d)$, the number of distinct partitions of d , hence

$$K_{DP} \leq N_{Part}(d) \cdot (n + k + 1), \quad (12)$$

establishing Eq. (8) and proving Obs. 8. We get the following.

1:10 On Realizing a Single Degree Sequence by a Bipartite Graph

► **Lemma 9.** *The BDR problem admits a polynomial time output sensitive algorithm. More specifically, given an integer sequence $d = (d_1, d_2, \dots, d_n)$, such that $d_i < n$ for every i , it is possible to find all block partitions of d in time $\mathcal{O}(T_{\text{Part}}(n) \cdot n \cdot N_{\text{Part}}(d))$, where $T_{\text{Part}}(n)$ is the time complexity of the best pseudo-polynomial time algorithm for deciding the PARTITION problem.*

Due to Theorem 4, the minimum t such that a given partition is t -bigraphic can be computed efficiently implying the following result.

► **Corollary 10.** *Let d be a degree sequence of length n such that $N_{\text{Part}}(d) = \mathcal{O}(n^c)$ for some constant c . Then, the minimum t such that d is t -bigraphic can be computed in polynomial time. As a special case, the BDR problem for d can be solved in polynomial time.*

We remark that a useful special subclass consists of sequences with a *constant* number of different degrees since such a sequence can have at most polynomially many different partitions.

► **Corollary 11.** *Let p be some constant and $d = (d_1^{n_1}, d_2^{n_2}, \dots, d_p^{n_p})$ a degree sequence where $n = \sum_{i=1}^p n_i$. Then, $N_{\text{Part}}(d) = \mathcal{O}(n^c)$ for some constant c .*

3.2 Small Maximum Degree

Towards attacking the realizability problem of general bigraphic sequences, we first look at the question of bounding the total deviation of a nonincreasing sequence $d = (d_1, \dots, d_n)$ as a function of its maximum degree, $\Delta = d_1$.

Burstein and Rubin [8] consider the realization problem for directed graphs with loops, which is equivalent to BDR^P . They give the following sufficient condition for a pair of sequences to be the in- and out-degrees of a directed graph with loops.

► **Theorem 12 ([8]).** *Consider a degree sequence d with a partition $(a, b) \in \text{BP}(d)$ assuming that a and b have the same length p . Let $\sum a = \sum b = pc$ where c is the average degree. If $a_1 b_1 \leq pc + 1$, then d is realizable by a directed graph with loops.*

In the following, their result is extended to bipartite multigraphs with bounded maximum multiplicity, i.e., to t -bigraphic sequences. We make use of the following straightforward technical claim.

► **Observation 13.** *Consider a nonincreasing integer sequence $x = (x_1, \dots, x_k)$ of total sum $\sum x = X$. Then, $\sum (x[\ell]) \geq \ell X/k$, for every $1 \leq \ell \leq k$.*

Proof. Since x is nonincreasing, $\text{avg}(x_1, \dots, x_\ell) \geq \text{avg}(x_{\ell+1}, \dots, x_k)$, or more formally, $(\sum_{i=1}^{\ell} x_i)/\ell \geq (\sum_{i=\ell+1}^k x_i)/(k - \ell)$. Consequently,

$$X = \sum_{i=1}^k x_i = \sum_{i=1}^{\ell} x_i + \sum_{i=\ell+1}^k x_i \leq \sum_{i=1}^{\ell} x_i + \frac{k-\ell}{\ell} \sum_{i=1}^{\ell} x_i = \frac{k}{\ell} \sum_{i=1}^{\ell} x_i,$$

implying the claim. ◀

► **Lemma 14.** *Let t be a positive integer. Consider a degree sequence d of length n with a partition $(a, b) \in \text{BP}(d)$. If $a_1 \cdot b_1 \leq t \cdot \sum d/2$, then (a, b) is t -bigraphic.*

Proof. Let t, d and (a, b) as in the lemma. We first verify that the condition $a_1 \cdot b_1 \leq t \sum d/2$ implies that $b_1 \leq t|a|$. Towards a contradiction, suppose that $|a| < b_1/t$. It follows that

$$\sum a \leq a_1 \cdot |a| < a_1 \cdot b_1/t \leq \sum d/2 = \sum a,$$

a contradiction.

Let $X = \sum a = \sum b = \sum d/2$. By the assumption, $a_1 \cdot b_1 \leq tX$. Note that the conjugate sequence \tilde{b} is nonincreasing, $\sum(\tilde{b}[b_1]) = X$, and $\tilde{b}_i = 0$ for $i > b_1$. By Observation 13,

$$\sum(\tilde{b}[\ell t]) \geq \ell t X/b_1 \geq \ell a_1$$

for every $1 \leq \ell \leq b_1/t$. As $a_i \leq a_1$ for every i , we also have

$$\sum(a[\ell]) \leq \ell a_1$$

for every $1 \leq \ell \leq b_1/t$. Combined, we have

$$\sum(a[\ell]) \leq \sum(\tilde{b}[\ell t])$$

for every $1 \leq \ell \leq b_1/t$. The same inequality holds also for $|a| \geq \ell > b_1/t$, since $\sum(a[\ell]) \leq \sum a = X = \sum(\tilde{b}[\ell t])$. Note that $\sum(\tilde{b}[\ell t]) = \sum_{i=1}^m \min\{\ell t, b_i\}$. It follows that $\sum(a[\ell]) \leq \sum_{i=1}^m \min\{\ell t, b_i\}$ for $1 \leq \ell \leq |a|$, implying the lemma due to Theorem 4. ◀

Example 7 (see Sect. 5) establishes the following.

► **Lemma 15.** *The above bound is tight for some degree sequences.*

Lemma 14 is stated for a given partition (BDR^P) . For BDR, we immediately have the following (with the second observation following from the first as $\sum d \geq n$).

► **Corollary 16.** *Let t be a positive integer. For every partitionable degree sequence $d = (d_1, \dots, d_n)$,*

(1.) *if $d_1^2 \leq t \cdot \sum d/2$, then any partition $(a, b) \in \text{BP}(d)$ is t -bigraphic.*

(2.) *if $d_1^2 \leq t \cdot n/2$, then any partition $(a, b) \in \text{BP}(d)$ is t -bigraphic.*

This allows us to state the following for bounded-degree sequences whose maximum degree Δ is constant.

► **Corollary 17.** *Let Δ, t be positive integers. For every $n \geq 2\Delta^2/t$, and for every degree sequence $d = (d_1, \dots, d_n)$ such that $d_1 \leq \Delta$, if d is partitionable then it is t -bigraphic.*

The extreme bound of this type is obtained when the sequence d has a balanced High-Low partition, in which case we get the following.

► **Corollary 18.** *Let t be a positive integer. Consider a degree sequence $d = (d_1, \dots, d_n)$ with a balanced High-Low partition (H, L) where $H = (d_1, \dots, d_k)$ and $L = (d_{k+1}, \dots, d_n)$. If $d_1 \cdot d_{k+1} \leq t \cdot \sum d/2$, then (H, L) is t -bigraphic.*

4 Realizations based on the Equal or High-Low partitions

4.1 Realizations using the High-Low partition

Recall that a well-behaved High-Low partition is a balanced High-Low partition (H, L) , $H = (d_1, \dots, d_k)$ and $L = (d_{k+1}, \dots, d_n)$, which satisfies the first Gale-Ryser conditions, i.e., $d_1 \leq n - k$ and $d_{k+1} \leq k$. Such a partition may or may not be bigraphic (see Example 4 in Sect. 5). However, for a non-increasing sequence d that admits a well-behaved High-Low partition (H, L) , the BDR problem turns out to be solvable [4].

More explicitly, when d admits a well-behaved High-Low partition (H, L) , it suffices to test the (entire collection of) Gale-Ryser conditions on (H, L) . The realizability of d is then decided as follows.

- If all the Gale-Ryser conditions are met, then (H, L) is a bigraphic degree partition, hence d is a bigraphic degree sequence.
- Conversely, if one or more of the Gale-Ryser conditions is violated for (H, L) , then *every* partition of d must violate one Gale-Ryser condition and d has *no* bigraphic degree partition. Consequently, d itself is not a bigraphic degree sequence.

Relying on the adapted Havel-Hakimi theorem described in Sect. 2, and on the resulting algorithm for computing a realizing bipartite graph given a bipartite degree partition, we conclude the following.

► **Theorem 19** ([4]). *Let d be a degree sequence with a well-behaved High-Low partition. It can be decided in polynomial time whether d is bigraphic or not. If d happens to be bigraphic, a bipartite graph realizing d can be computed in polynomial time.*

Hereafter, we examine degree sequences that have a balanced High-Low partition but are not well-behaved. Our goal is to generate bipartite multigraphs with low maximum multiplicity of parallel edges based on the High-Low partition.

In the following, let r be a positive integer, and let d be an r -graphic degree sequence with High-Low partition $HL(d) = (H, L)$ where $H = (d_1, \dots, d_k)$ and $L = (d_{k+1}, \dots, d_n)$, for some integer $k \in [1, n - 1]$. We quantify the violation of the first Gale-Ryser conditions with the following definitions. Let

$$t_H(d) = \left\lceil \frac{d_1}{n - k} \right\rceil \quad \text{and} \quad t_L(d) = \left\lceil \frac{d_{k+1}}{k} \right\rceil,$$

and define $t(d) = \max\{t_H(d), t_L(d)\}$. (Note that sequence d has a well-behaved High-Low partition if $t(d) = 1$.) First, we observe that $t_H(d)$ is bounded for r -graphic sequences.

► **Lemma 20** ([4]). *Let d be an r -graphic degree sequence with High-Low partition $HL(d) = (H, L)$. Then, $t_H(d) \leq 2r$.*

The main result is the following.

► **Theorem 21** ([4]). *Let d be an r -graphic degree sequence with High-Low partition $HL(d) = (H, L)$ and let $t = \max\{t(d), 2r\}$. Then, (H, L) is t -bigraphic.*

Example 8 in Sect. 5 shows that the conclusion of Theorem 21 does not hold if the degree sequence d is not r -graphic. Theorem 21 is complemented by an existential lower bound. In [4], it is shown that there are degree sequences d with High-Low partition $HL(d)$ such that $t(d) > 1$, and d is not t' -bigraphic for any $t' < t(d)$.

For graphic degree sequences, we state the following result.

► **Corollary 22** ([4]). *Let d be a graphic degree sequence with High-Low partition (H, L) and $t = t(d)$.*

- (i) *If $t = 1$, then (H, L) is 2-bigraphic.*
- (ii) *If $t > 1$, then (H, L) is t -bigraphic.*

In case there is a well-behaved High-Low partition, Theorem 19 implies the following.

► **Corollary 23** ([4]). *Let d be a graphic degree sequence with well-behaved High-Low partition $HL(d) = (H, L)$. Then, either*

- (i) *(H, L) is bigraphic, or*
- (ii) *d is not bigraphic and (H, L) is 2-bigraphic.*

To close this section, we present bounds on $t_L(d)$ and $t_H(d)$ in case degree sequence d is r -graphic or bigraphic. The next theorem establishes a bound on $t_L(d)$ for an r -graphic sequence. (A bound on $t_H(d)$ is already shown with Lemma 20.)

► **Theorem 24** ([4]). *Let d be an r -graphic sequence with High-Low partition (H, L) . Then,*

$$t_H(d) \leq 2r, \quad \text{and} \quad t_L(d) \leq \left\lceil \frac{r(k+1)}{2} \right\rceil.$$

Examples 9 & 10 in Sect. 5 show that the bound of Theorem 24 is tight and that for graphic sequences, $t_L(d) < t_H(d)$ as well as $t_H(d) < t_L(d)$ can occur.

Finally, the next theorem gives improved bounds for bigraphic degree sequences.

► **Theorem 25** ([4]). *Let d be a bigraphic sequence with High-Low partition $HL(d)$. Then,*

$$t_H(d) \leq 1, \quad \text{and} \quad t_L(d) \leq \left\lceil \frac{k+2+1/k}{4} \right\rceil.$$

4.2 Realizations using the Equal partition

We next consider degree sequences that are even, i.e., where each degree occurs an even number of times. Let q be a positive integer, and let d be an even degree sequence consisting of n_i copies of the integer d_i for $1 \leq i \leq q$ where $\sum_{i=1}^q n_i = n$. We adopt the notation $d = (d_1^{n_1}, \dots, d_q^{n_q})$. As n_i is even, for every $1 \leq i \leq q$, the equal partition $EQP(d) = (a, b)$ where $a = b = (d_1^{n_1/2}, \dots, d_q^{n_q/2})$ is well-defined.

As mentioned earlier, a well-behaved equal partition does not seem to enable resolving the BDR problem, as there are even sequences d with an equal partition $EQP(d) = (a, b)$ that satisfies the first Gale-Ryser conditions where (a, b) is not bigraphic but other partitions of d are bigraphic (see Example 5 in Sect. 5). However, it is shown in [3] that if the sequence d is graphic and even, then the equal partition is 2-bigraphic. In general, we have the following result.

► **Theorem 26** ([3]). *Let d be an even and r -graphic degree sequence of length n with equal partition $EQP(d) = (a, b)$. Then, (a, b) is $2r$ -bigraphic.*

5 Examples

We adopt the notation $d = (d_1^{n_1}, \dots, d_q^{n_q})$, where $\sum_{i=1}^q n_i = n$, for a sequence consisting of n_i copies of the integer d_i for $1 \leq i \leq q$.

1:14 On Realizing a Single Degree Sequence by a Bipartite Graph

► **Example 1.** Consider the sequence $d = (6^2, 4^2, 2^6)$.

This sequence is even, so it has an equal partition. It also has a balanced High-Low partition, as well as a third partition:

- (i.) $a = (6, 4, 2^3)$ and $b = (6, 4, 2^3)$ (the equal partition),
- (ii.) $a' = (6^2, 4)$ and $b' = (4, 2^6)$ (the High-Low partition),
- (iii.) $a'' = (6^2, 2^2)$ and $b'' = (4^2, 2^4)$.

However, only the last partition, (a'', b'') , is bigraphic.

► **Example 2.** Consider the (non-graphic) sequence $d = (n - 1, n - 2, \dots, 3, 2, 1)$ for n divisible by 4. Split d into length-4 subsequences

$$B_1 = (1, 2, 3, 4), \quad B_2 = (5, 6, 7, 8), \quad \dots$$

For each subsequence $B_j = (x, x + 1, x + 2, x + 3)$ for $x = 4(j - 1) + 1$, it is possible to place $(x, x + 3)$ on one side of the partition and $(x + 1, x + 2)$ on the other side. This yields $2^{n/4}$ different partitions of d .

► **Example 3.** Consider the graphic sequence $d = (n, n, n - 1, n - 1, \dots, 2, 2, 1, 1)$ of length $2n$, for n divisible by 4. Split d into length-8 subsequences

$$B_1 = (1, 1, 2, 2, 3, 3, 4, 4), \quad B_2 = (5, 5, 6, 6, 7, 7, 8, 8), \quad \dots$$

Each subsequence $B_j = (x, x, x + 1, x + 1, x + 2, x + 2, x + 3, x + 3)$, for $x = 4(j - 1) + 1$, has three partitions:

- (i.) $a = (x, x + 1, x + 2, x + 3)$ and $b = (x, x + 1, x + 2, x + 3)$,
- (ii.) $a' = (x, x, x + 3, x + 3)$ and $b' = (x + 1, x + 1, x + 2, x + 2)$,
- (iii.) $a'' = (x + 1, x + 1, x + 2, x + 2)$ and $b'' = (x, x, x + 3, x + 3)$.

This yields $3^{n/4}$ different partitions of d .

► **Example 4.** Consider the sequence $d = ((6m)^m, (2m)^{5m+1}, 1^{2m})$.

This sequence has a well-behaved High-Low partition

$$H = ((6m)^m, (2m)^{m+1}), \quad L = ((2m)^{4m}, 1^{2m}),$$

but it is not bigraphic.

► **Example 5.** Consider the sequence $d = ((k^2)^k, k^{k^2}, 1^{k^2})$.

Its High-Low partition

$$H = ((k^2)^k, k^{k/2}), \quad L = (k^{k^2-k/2}, 1^{k^2})$$

is bigraphic, while its equal partition

$$a = b = ((k^2)^{k/2}, k^{k^2/2}, 1^{k^2/2})$$

is not.

► **Example 6.** Consider the sequence $d = (k^k, 1^{k^2-2k})$.

Its equal partition

$$a = b = (k^{k/2}, 1^{k^2/2-k})$$

is bigraphic, while its High-Low partition

$$H = (k^{k-1}), \quad L = (k, 1^{k^2-2k})$$

is not.

► **Example 7.** Consider the sequence $d = (t^{2k})$ for positive integers t, k such that $t > k$. This sequence has a unique partition

$$(a, a) \in \text{BP}(d) \quad \text{where} \quad a = (t^k).$$

One can verify that

$$\frac{2(a_1 \cdot a_1)}{\sum d} = \frac{t}{k} \leq \left\lceil \frac{t}{k} \right\rceil.$$

The partition (a, a) is $\lceil t/k \rceil$ -bigraphic but no better.

► **Example 8.** Consider the non-graphic sequence $d = ((9m)^{m-1}, 6m + 1, (3m)^{3m-1}, 1^1)$ for some positive integer m . Its High-Low partition is

$$H = ((9m)^{m-1}, 6m + 1), \quad L = ((3m)^{3m-1}, 1^1).$$

We have $t_H(d) = t_L(d) = 3$, but the conditions of Theorem 4 for 3-bigraphic degree sequences are violated. Specifically, the condition for index $m - 1$ requires

$$9m(m - 1) \leq (3m - 1) \cdot 3 \cdot (m - 1) + 1,$$

which is false.

► **Example 9.** Consider the graphic sequence $d = (6, 3^6)$ which has exactly one (High-Low) partition (H, L) with

$$H = (6, 3^2), \quad L = (3^4).$$

One can verify that $t_L(d) = 2$ and $t_H(d) = 1$.

► **Example 10.** Consider the degree sequence $d' = ((\frac{k(k+1)}{2})^{k+1}, 1^{\frac{k(k+1)}{2}(k-1)})$, for a positive integer k . To see that d' is graphic, observe that $\sum d'$ is even, and that the $(k + 1)$ th-EG inequality holds (For such a block sequence this is sufficient, see, e.g., [30]). The $(k + 1)$ th-EG inequality requires

$$(k(k + 1)/2) \cdot (k + 1) \leq k(k + 1) + (k(k + 1)/2) \cdot (k - 1),$$

which trivially holds. Moreover, $HL(d') = (H', L')$ where

$$H' = ((\frac{k(k + 1)}{2})^k), \quad L' = ((\frac{k(k + 1)}{2}), 1^{\frac{k(k+1)}{2}(k-1)}).$$

Hence, $|H'| = k$ and $d_{k+1} = \frac{k(k+1)}{2}$.

References

- 1 Patrick Adams and Yuri Nikolayevsky. Planar bipartite biregular degree sequences. *Discr. Math.*, 342:433–440, 2019.
- 2 Martin Aigner and Eberhard Triesch. Realizability and uniqueness in graphs. *Discr. Math.*, 136:3–20, 1994.
- 3 Amotz Bar-Noy, Toni Böhnlein, David Peleg, and Dror Rawitz. On realizing even degree sequences by bipartite graphs. Unpublished manuscript, 2022.
- 4 Amotz Bar-Noy, Toni Böhnlein, David Peleg, and Dror Rawitz. On the role of high-low partitions in realizing a degree sequence by a bipartite graph. Unpublished manuscript, 2022.

- 5 Richard Bellman. Notes on the theory of dynamic programming iv-maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, 1956.
- 6 Joseph K. Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, 6(4):489–522, 2011.
- 7 D. Burstein and J. Rubin. Sufficient conditions for graphicality of bidegree sequences. *SIAM J. Discr. Math.*, 31:50–62, 2017.
- 8 David Burstein and Jonathan Rubin. Sufficient conditions for graphicality of bidegree sequences. *SIAM Journal on Discrete Mathematics*, 31(1):50–62, 2017.
- 9 A. A. Chernyak, Z. A. Chernyak, and R. I. Tyshkevich. On forcibly hereditary p -graphical sequences. *Discr. Math.*, 64:111–128, 1987.
- 10 Sheshayya A. Choudum. A simple proof of the Erdős-Gallai theorem on graph sequences. *Bull. Austral. Math. Soc.*, 33(1):67–70, 1991.
- 11 V Chungphaisan. Conditions for sequences to be r -graphic. *Discr. Math.*, 7(1-2):31–39, 1974.
- 12 Brian Cloteaux. Fast sequential creation of random realizations of degree sequences. *Internet Mathematics*, 12(3):205–219, 2016.
- 13 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
- 14 Geir Dahl and Truls Flatberg. A remark concerning graphical sequences. *Discr. Math.*, 304(1-3):62–64, 2005.
- 15 Dóra Erdős, Rainer Gemulla, and Evimaria Terzi. Reconstructing graphs from neighborhood data. *ACM Trans. Knowledge Discovery from Data*, 8(4):23:1–23:22, 2014.
- 16 Paul Erdős and Tibor Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.
- 17 D. Gale. A theorem on flows in networks. *Pacific J. Math.*, 7:1073–1082, 1957.
- 18 Gautam Gupta, Puneet Joshi, and Amitabha Tripathi. Graphic sequences of trees and a problem of Frobenius. *Czechoslovak Math. J.*, 57:49–52, 2007.
- 19 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph –I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.
- 20 Peter L. Hammer, Toshihide Ibaraki, and Bruno Simeone. Threshold sequences. *SIAM J. Algebra. Discr.*, 2(1):39–49, 1981.
- 21 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1:275–284, 1981.
- 22 V. Havel. A remark on the existence of finite graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.
- 23 Heather Hulett, Todd G Will, and Gerhard J Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Oper. Res. Lett.*, 36(5):594–596, 2008.
- 24 P.J. Kelly. A congruence theorem for trees. *Pacific J. Math.*, 7:961–968, 1957.
- 25 Daniel J Kleitman. Minimal number of multiple edges in realization of an incidence sequence without loops. *SIAM J. on Applied Math.*, 18(1):25–28, 1970.
- 26 Daniel J Kleitman and Da-Lun Wang. Algorithms for constructing graphs and digraphs with given valences and factors. *Discrete Mathematics*, 6(1):79–88, 1973.
- 27 Manfred Krause. A simple proof of the gale-ryser theorem. *The American Mathematical Monthly*, 103(4):335–337, 1996.
- 28 P. Marchioro, A. Morgana, R. Petreschi, and B. Simeone. Degree sequences of matrogenic graphs. *Discrete Mathematics*, 51(1):47–61, 1984.
- 29 Milena Mihail and Nisheeth Vishnoi. On generating graphs with prescribed degree sequences for complex network modeling applications. *3rd ARACNE*, 2002.
- 30 Jeffrey W Miller. Reduced criteria for degree sequences. *Discrete Mathematics*, 313(4):550–562, 2013.
- 31 AB Owens and HM Trent. On determining minimal singularities for the realizations of an incidence sequence. *SIAM J. on Applied Math.*, 15(2):406–418, 1967.

- 32 Alvin B Owens. On determining the minimum number of multiple edges for an incidence sequence. *SIAM J. on Applied Math.*, 18(1):238–240, 1970.
- 33 S. B. Rao. A survey of the theory of potentially p -graphic and forcibly p -graphic degree sequences. In *Combinatorics and graph theory*, volume 885 of *LNM*, pages 417–440, 1981.
- 34 H.J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canad. J. Math.*, 9:371–377, 1957.
- 35 E. F. Schmeichel and S. L. Hakimi. On planar graphical degree sequences. *SIAM J. Applied Math.*, 32:598–609, 1977.
- 36 Gerard Sierksma and Han Hoogeveen. Seven criteria for integer sequences being graphic. *J. Graph Theory*, 15(2):223–231, 1991.
- 37 Akutsu Tatsuya and Hiroshi Nagamochi. Comparison and enumeration of chemical graphs. *Computational and structural biotechnology*, 5, 2013.
- 38 Amitabha Tripathi and Himanshu Tyagi. A simple criterion on degree sequences of graphs. *Discr. Appl. Math.*, 156(18):3513–3517, 2008.
- 39 Amitabha Tripathi, Sushmita Venugopalan, and Douglas B. West. A short constructive proof of the Erdős-Gallai characterization of graphic lists. *Discr. Math.*, 310(4):843–844, 2010.
- 40 Amitabha Tripathi and Sujith Vijay. A note on a theorem of Erdős & Gallai. *Discr. Math.*, 265(1-3):417–420, 2003.
- 41 R. I. Tyshkevich, A. A. Chernyak, and Z. A. Chernyak. Graphs and degree sequences: a survey, I. *Cybernetics*, 23:734–745, 1987.
- 42 R. I. Tyshkevich, A. A. Chernyak, and Z. A. Chernyak. Graphs and degree sequences: a survey, II. *Cybernetics*, 24:137–152, 1988.
- 43 R. I. Tyshkevich, A. A. Chernyak, and Z. A. Chernyak. Graphs and degree sequences: a survey, III. *Cybernetics*, 24:539–548, 1988.
- 44 Regina Tyshkevich. Decomposition of graphical sequences and unigraphs. *Discr. Math.*, 220:201–238, 2000.
- 45 S.M. Ulam. *A collection of mathematical problems*. Wiley, 1960.
- 46 N.C. Wormald. Models of random regular graphs. *Surveys in Combin.*, 267:239–298, 1999.
- 47 Igor E Zverovich and Vadim E Zverovich. Contributions to the theory of graphic sequences. *Discrete Mathematics*, 105(1-3):293–303, 1992.
- 48 Igor E. Zverovich and Vadim E. Zverovich. Contributions to the theory of graphic sequences. *Discr. Math.*, 105(1-3):293–303, 1992.

Time, Clocks and Efficiency of Population Protocols

Leszek Gąsieniec¹ ✉ 🏠 

University of Liverpool, UK

Grzegorz Stachowiak ✉ 

University of Wrocław, Poland

Abstract

The model of population protocols is used to study distributed processes based on pairwise interactions between simple anonymous agents drawn from a large population of size n . The order in which agents meet in pairs is determined by the random scheduler, s.t., each consecutive pair is chosen uniformly at random. After each interaction the state of the relevant agents are amended according to the predefined transition function (the actual code of the algorithm) which governs the considered process. The state space of agents is often fixed and the size n is not known in advance, i.e., not hard-coded in the transition function. We assume that a population protocol starts in the predefined initial configuration of agents' states representing the input. And if successful, the protocol stabilises in a final configuration of states forming the output representing the solution to the considered problem.

The time complexity of a population protocol refers to the number of interactions required to stabilise this protocol in a final configuration. We also define *parallel time* as the time complexity divided by n . Note that the parallel time of the system and the expected local time of each agent, i.e., the number of interactions observed by each agent, are correlated. Several mechanisms, known as *phase clocks*, have been developed to measure parallel time more accurately than counting local interactions. Most of the clocks target counting $\Theta(\log n)$ parallel time required to fully synchronise all agents in the population. There are leader (and junta) based phase clocks which utilise a fixed number of states [2, 4]. This type of clocks allows also counting any poly-logarithmic time while preserving fix state utilisation. The other type refers to leaderless clocks utilising $\Theta(\log n)$ states [1, 5]. This type allows approximate counting of parallel time as fixed resolution clocks [5] or oscillators [1]. Another clock type introduced recently in [3] enables counting $\Theta(n \log n)$ parallel time utilising a fixed number of states and either leaders or connections in the network constructor model.

We also discuss parallel efficiency of population protocols referring to protocols operating in $\Theta(\text{poly } \log n)$ parallel time, we propose extensions of the population protocol model leading to further improvement in state and time utilisation, and we state some open problems.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Population protocols, phase clocks, oscillators, parallel time and efficiency

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.2

Category Invited Paper

References

- 1 D. Alistarh, J. Aspnes, and R. Gelashvili. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 2221–2239. SIAM, 2018.
- 2 D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008.

¹ Corresponding author



2:2 Time, Clocks and Efficiency of Population Protocols

- 3 L. Gašieniec, P.G. Spirakis, and G. Stachowiak. New clocks, fast line formation and self-replication population protocols. *CoRR*, abs/2111.10822, 2021. [arXiv:2111.10822](#).
- 4 L. Gašieniec and G. Stachowiak. Enhanced phase clocks, population protocols, and fast space optimal leader election. *J. ACM*, 68(1):2:1–2:21, 2021.
- 5 D. Doty, M. Eftekhari, L. Gašieniec, E.E. Severson, P. Uznanski, and G. Stachowiak. A time and space optimal stable population protocol solving exact majority. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCs 2021*, pages 1044–1055. IEEE, 2021.

Reconstructing the Tree of Life (Fitting Distances by Tree Metrics)

Mikkel Thorup  

BARC, Department of Computer Science, University of Copenhagen, Denmark

Abstract

We consider the numerical taxonomy problem of fitting an $S \times S$ distance matrix D with a tree metric T . Here T is a weighted tree spanning S where the path lengths in T induce a metric on S . If there is a tree metric matching D exactly, then it is easily found. If there is no exact match, then for some k , we want to minimize the L_k norm of the errors, that is, pick T so as to minimize

$$\|D - T\|_k = \left(\sum_{i,j \in S} |D(i,j) - T(i,j)|^k \right)^{1/k}.$$

This problem was raised in biology in the 1960s for $k = 1, 2$. The biological interpretation is that T represents a possible evolution behind the species in S matching some measured distances in D . Sometimes, it is required that T is an ultrametric, meaning that all species are at the same distance from the root.

An evolutionary tree induces a hierarchical classification of species and this is not just tied to biology. Medicine, ecology and linguistics are just some of the fields where this concept appears, and it is even an integral part of machine learning and data science. Fundamentally, if we can approximate distances with a tree, then they are much easier to reason about: many questions that are NP-hard for general metrics can be answered in linear time on tree metrics. In fact, humans have appreciated hierarchical classifications at least since Plato and Aristotle (350 BC).

The numerical taxonomy problem is important in practice and many heuristics have been proposed. In this talk we will review the basic algorithmic theory, results and techniques, for the problem, including the most recent result from FOCS'21 [3]. They paint a varied landscape with big differences between different moments, and with some very nice open problems remaining.

- At STOC'93, Farach, Kannan, and Warnow [4] proved that under L_∞ , we can find the optimal ultrametric. Almost all other variants of the problem are APX-hard.
- At SODA'96, Agarwala, Bafna, Farach, Paterson, and Thorup [1] showed that for any norm L_k , $k \geq 1$, if the best ultrametric can be α -approximated, then the best tree metric can be 3α -approximated. In particular, this implied a 3-approximation for tree metrics under L_∞ .
- At FOCS'05, Ailon and Charikar [2] showed that for any L_k , $k \geq 1$, we can get an approximation factor of $O((\log n)(\log \log n))^{1/k}$ for both tree and ultrametrics. Their paper was focused on the L_1 norm, and they wrote “Determining whether an $O(1)$ approximation can be obtained is a fascinating question”.
- At FOCS'21, Cohen-Addad, Das, Kipouridis, Parotsidis, and Thorup [3] showed that indeed a constant factor is possible for L_1 for both tree and ultrametrics. This uses the special structure of L_1 in relation to hierarchies.
- The status of L_k is wide open for $1 < k < \infty$. All we know is that the approximation factor is between $\Omega(1)$ and $O((\log n)(\log \log n))$.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Numerical taxonomy, computational phylogenetics, hierarchical clustering

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.3

Category Invited Paper

Funding *Mikkel Thorup*: Supported by Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.



© Mikkel Thorup;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 3; pp. 3:1–3:2

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

References

- 1 Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, 1999. Announced at SODA 1996.
- 2 Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. *SIAM J. Comput.*, 40(5):1275–1291, 2011. Announced at FOCS 2005.
- 3 Vincent Cohen-Addad, Debarati Das, Evangelos Kipouridis, Nikos Parotsidis, and Mikkel Thorup. Fitting distances by tree metrics minimizing the total error within a constant factor. In *Proc. 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 468–479, 2021. doi:10.1109/FOCS52979.2021.00054.
- 4 Martin Farach, Sampath Kannan, and Tandy J. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1/2):155–179, 1995. Announced at STOC 1993.

Compacting Squares: Input-Sensitive In-Place Reconfiguration of Sliding Squares

Hugo A. Akitaya  



University of Massachusetts Lowell, MA, USA

Erik D. Demaine  

Massachusetts Institute of Technology,
Cambridge, MA, USA

Matias Korman 

Siemens Electronic Design Automation,
Wilsonville, OR, USA

Irina Kostitsyna  

Eindhoven University of Technology,
The Netherlands

Irene Parada  

Technical University of Denmark, Lyngby,
Denmark

Willem Sonke  

Eindhoven University of Technology,
The Netherlands

Bettina Speckmann  

Eindhoven University of Technology,
The Netherlands

Ryuhei Uehara  

Japan Advanced Institute of Science and
Technology, Ishikawa, Japan

Jules Wulms  

Technische Universität Wien, Austria

Abstract

Edge-connected configurations of square modules, which can reconfigure through so-called sliding moves, are a well-established theoretical model for modular robots in two dimensions. Dumitrescu and Pach [Graphs and Combinatorics, 2006] proved that it is always possible to reconfigure one edge-connected configuration of n squares into any other using at most $O(n^2)$ sliding moves, while keeping the configuration connected at all times.

For certain pairs of configurations, reconfiguration may require $\Omega(n^2)$ sliding moves. However, significantly fewer moves may be sufficient. We prove that it is NP-hard to minimize the number of sliding moves for a given pair of edge-connected configurations. On the positive side we present Gather&Compact, an input-sensitive in-place algorithm that requires only $O(\bar{P}n)$ sliding moves to transform one configuration into the other, where \bar{P} is the maximum perimeter of the two bounding boxes. The squares move within the bounding boxes only, with the exception of at most one square at a time which may move through the positions adjacent to the bounding boxes. The $O(\bar{P}n)$ bound never exceeds $O(n^2)$, and is optimal (up to constant factors) among all bounds parameterized by just n and \bar{P} . Our algorithm is built on the basic principle that well-connected components of modular robots can be transformed efficiently. Hence we iteratively increase the connectivity within a configuration, to finally arrive at a single solid xy -monotone component.

We implemented Gather&Compact and compared it experimentally to the in-place modification by Moreno and Sacristán [EuroCG 2020] of the Dumitrescu and Pach algorithm (MSDP). Our experiments show that Gather&Compact consistently outperforms MSDP by a significant margin, on all types of square configurations.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Sliding cubes, Reconfiguration, Modular robots, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.4

Related Version *Full Version*: <https://arxiv.org/abs/2105.07997>

Supplementary Material *Software*: <https://alga.win.tue.nl/software/compacting-squares/>

Funding Irene Parada was supported by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) “Dynamic Network Analysis” and Jules Wulms was supported partially by the Austrian Science Fund (FWF) under grant P31119 and partially by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035.



© Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 4; pp. 4:1–4:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements Parts of this work were initiated at the 5th Workshop on Applied Geometric Algorithms (AGA 2020) and at the 2nd Virtual Workshop on Computational Geometry. We thank all participants for discussions and an inspiring and productive atmosphere. We thank Fabian Klute for discussions on the computational experiments.

1 Introduction

Self-reconfigurable modular robots [20] promise adaptive, robust, scalable, and cheap solutions in a wide range of technological areas, from aerospace engineering to medicine. Modular robots are envisioned to consist of identical building blocks arranged in a lattice and are intended to be highly versatile, due to their ability to reconfigure into arbitrary forms. An actual realization of this vision depends on fast and reliable reconfiguration algorithms, which hence have become an area of growing interest.

One of the best-studied paradigms of modular robots is the *sliding cube model* [11]. In this model, a robot *configuration* is a face-connected set of cubic modules on the cubic grid. The cubes can perform two types of (*sliding*) moves, illustrated in two dimensions in Figure 1.



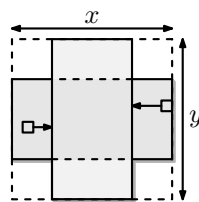
■ **Figure 1** Moves admitted by the sliding cube model: (a) slide, (b) convex transition.

First, a module can *slide* along two face-adjacent cubes to reach a face-adjacent empty grid cell. Second, a module m can make a *convex transition* around a module m' to end in an edge-adjacent empty grid cell. For this second move to be feasible, also the grid cell (not occupied by m') face-adjacent to both the starting and the ending positions must be empty. Moves need to maintain connectivity, that is, a cube c is movable only if the configuration without c would still be face-connected. There are several prototypes of modular robots that realize the sliding cube model in 2D [6, 9, 12]. Units of multiple other prototypes, including expandable and contractible units [17, 18] as well as large classes of modular robots [4, 16], can be arranged into cubic *meta-modules* consisting of several units such that the meta-module can perform slide and convex transition moves. Thus, algorithmic solutions in the sliding cube model can be applied to modular robot systems realizing other models.

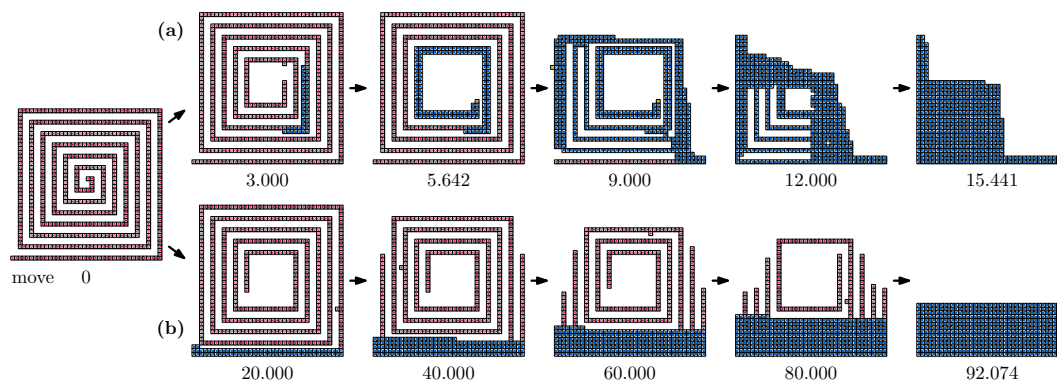
Another well-studied model for modular robots is the *pivoting cube model* [7, 19]. This model strengthens the free-space requirements for each move and has also been realized by some existing prototypes. Previous work [2] showed very recently that in the pivoting cube model in two dimensions it is PSPACE-hard to decide whether it is possible at all to reconfigure one configuration into the other. However, if one allows six auxiliary squares in addition to the input configuration, then there is a worst-case optimal reconfiguration algorithm [1]. Other models for squares relax the face-connectivity condition [8], restrict or enlarge the set of allowed moves [13], or relax the free-space requirements [5].

In this paper we study the reconfiguration problem for the sliding cube model in two dimensions (the *sliding square model*). Given two configurations of n unlabeled squares (each describing the relative positions of squares), we compute a short sequence of moves that transforms one configuration into the other, while preserving edge-connectivity at all times. Dumitrescu and Pach [10] described an algorithm which transforms any two configurations of n squares into each other using $O(n^2)$ moves. This bound is worst-case optimal: there are pairs of configurations (a horizontal and a vertical line) which require $\Omega(n^2)$ moves for any transformation. However, for other pairs of configurations, significantly fewer moves suffice.

We show in Section 2 that it is NP-hard to minimize the number of moves for a given pair of edge-connected configurations. Due to the $O(n^2)$ upper bound on the number of moves, the corresponding decision problem is NP-complete. In Section 3, we present an input-sensitive and in-place algorithm for self-reconfiguration, based on the “compact-and-deploy” approach. Using the basic principle that well-connected components of modular robots can be transformed efficiently, our algorithm iteratively increases the connectivity within a configuration, to arrive at a single solid xy -monotone component, before deploying it into the target configuration. Hence, our algorithm builds the target configuration in such a way that the lower left corner of the bounding boxes of both configurations are aligned. Our algorithm is *input-sensitive*: it requires only $O(\bar{P}n)$ moves to transform one configuration into the other, where \bar{P} is the maximum perimeter of the two bounding boxes. Our algorithm is also *in-place*: only one square at a time is allowed to move outside the respective bounding box, and then only through cells vertex-adjacent to the bounding box.



Lower bound. Our $O(\bar{P}n)$ bound is optimal (up to constant factors) among all bounds parameterized by just n and \bar{P} . Given \bar{P} and n , consider a source and target configuration that each consist of a rectangle filled with squares, with dimensions $x \times y/2$ and $x/2 \times y$, respectively, such that $n = x/2 \cdot y = x \cdot y/2$ and $\bar{P} = \Theta(x + y)$. This is satisfied by choosing $x = \Theta(\bar{P})$ and $y = \Theta(n/\bar{P})$. Without loss of generality assume $x > y$, so that $\bar{P} = \Theta(x)$. Given the source configuration, the target configuration may be built anywhere in the grid, and for such a given *target (configuration) position*, we call all grid cells that should be filled with squares *target cells*. For any target position, every source square s that is not in a target cell requires at least as many moves as the L_∞ -distance d_s (along the grid) between s and its closest target cell. Given the set S of source squares not on target cells, the configuration then requires at least $\sum_{s \in S} d_s$ moves. For any target position, S contains at least an $x/4 \times y$



■ **Figure 2** A spiral configuration in a 40×40 bounding box. (a) Gather&Compact: gathering done after 5.642 moves; total 15.441 moves. (b) MSDP [10, 15]: total 92.074 moves.

Video: ■ <https://tinyurl.com/algaspiral>.

rectangle R of squares, either to the left or to the right of the vertical strip occupied by the target configuration. Each robot r in the i -th column of R has $d_r \geq i$, hence the total required movement is at least $\sum_{i=1}^{x/4} iy = \Theta((x/4)^2y) = \Theta(xn) = \Theta(Pn)$.

Comparison with Dumitrescu and Pach. The algorithm by Dumitrescu and Pach [10] constructs a canonical shape from both input configurations. In the original paper this canonical shape is a strip that extends to the right of a rightmost square and hence, necessarily, their algorithm always requires $\Omega(n^2)$ moves. Moreno and Sacristán [14, 15] modify the algorithm of Dumitrescu and Pach to be in-place; their canonical shape is a rectangle within the bounding box of the input. For either type of canonical shape their algorithm roughly proceeds as follows. If there is a square which is a leaf in the edge-adjacency graph, then the algorithm attempts to move this square along the boundary towards the canonical configuration. If this leaf square “gets stuck” on the way, and hence increases its connectivity, or if there is no leaf in the first place, then the algorithm identifies a 2-connected square on the outside of the configuration which it can move towards the canonical configuration. Hence, if configurations are tree-like (such as the spiral in Figure 2), then each square moves along all remaining squares, for a total of $\Omega(n^2)$ moves (see Figure 2 bottom row). However, the width and the height of this spiral configuration is $O(\sqrt{n})$. Our algorithm gathers $\Theta(\sqrt{n})$ squares from the end of the spiral and then compacts in a total of $O(n\sqrt{n})$ moves.

The in-place modification by Moreno and Sacristán of Dumitrescu and Pach (henceforth MSDP) has the potential to use fewer than $\Theta(n^2)$ moves in practice. In Section 4 we compare our Gather&Compact to MSDP experimentally; Gather&Compact consistently outperforms MSDP by a significant margin, on all types of square configurations.

2 Hardness of optimal reconfiguration

In this section we sketch the proof of Theorem 1, details can be found in the full version [3].

► **Theorem 1.** *Let \mathcal{C} and \mathcal{C}' be two configurations of n squares each and let k be a positive integer. It is NP-complete to determine whether we can transform \mathcal{C} into \mathcal{C}' using at most k moves while maintaining edge-connectivity at all times.*

We provide a reduction from PLANAR MONOTONE 3SAT. In particular, we start from a rectilinear drawing of a planar monotone 3SAT instance \mathcal{C} with l variables and m clauses (see Figure 3). We create a problem instance of the reconfiguration problem whose size is polynomial in l and m ; we show that \mathcal{C} can be satisfied if and only if the corresponding reconfiguration problem can be solved in at most $66m + 24l$ moves.

We replace each variable with a variable gadget, highlighted by an orange-shaded area in Figure 4 and summarized in Figure 5a. Consecutive variable gadgets are connected by a horizontal line of squares forming a central path of cycles through all variable gadgets (pink squares). More precisely, the gadget associated with variable x_i has k_i O-shaped cycles in the path of cycles, where k_i is the number of times that the variable x_i appears in \mathcal{C} . Each O-shaped cycle has two *prongs* (yellow squares). The spacing between gadgets is large enough for different variable gadgets not to interact in an optimal reconfiguration process.

The source and target configurations in the variable gadgets are very similar. The only difference is that the positions marked with \times in Figure 5a must be emptied and the positions marked with \circ must be occupied. Using an earth movers argument one can argue that a square must be transferred from the right of the gadget to the left, and the minimum number of moves required to do so is the horizontal distance between the positions, in this case

$20k_i + 20$. There are two paths that achieve this bound, namely a path along the top (shown in Figure 5a) or one along the bottom. These correspond to setting the variable to **true** or **false**, respectively. The remaining required changes need four additional moves.

► **Lemma 2.** *At least $20k_i + 24$ moves are necessary to reconfigure the variable gadget x_i . Moreover, this number of moves can be achieved only by moving one of the right \times squares to the left \circ position at the same height, following the path shown in Figure 4 or the equivalent path connecting the other \times and \circ pair along the bottom.*

Lemma 2 shows that to reconfigure using as few moves as possible we must transfer one of the two \times squares on the right to a \circ position on the left. During the process, the \times square creates cycles involving the central path of cycles and either all upper or lower prongs.

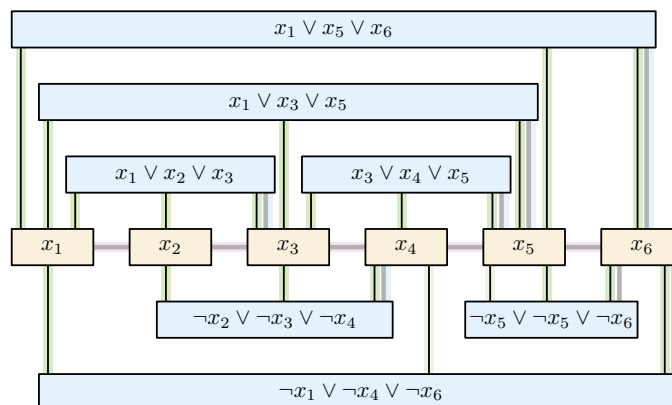
The clause gadget mainly consists of a set of squares forming a *pitchfork* (\pitchfork) shape (blue squares in Figure 5c). The pitchfork has three *tines* consisting of two squares each. Each tine corresponds to a literal in the clause. We add a path of squares connecting the \pitchfork shape to the central path of cycles so that the source configuration is connected (gray squares). Most squares are in both the source and the target configurations. The only exception is one square (marked with \times) that wants to be transferred to a nearby position (marked with \circ). However, the move is initially not possible as it would disconnect the \pitchfork part.

The wire gadgets are connected to the variable gadgets and part of them is placed very close to each of the tines of a pitchfork. A wire gadget is a path of squares that form a \sqcap shape for positive literals and a \sqcup shape for negative ones (see Figure 5b, green squares). Each wire gadget is attached to a different prong of the corresponding variable gadget. This associates each literal in a clause to a wire gadget and a prong (note that there can be spare prongs). The goal is to allow creating a different connection between the \pitchfork of a clause gadget and the central path of cycles in two moves as long as the prong is in a cycle.

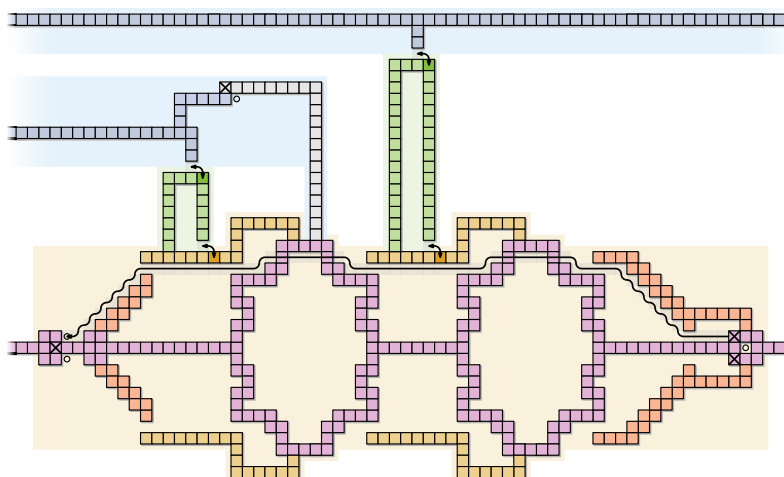
To avoid interference between different gadgets we place the clause gadgets at different heights and make the vertical separations between gadgets large enough.

► **Lemma 3.** *At least six moves are necessary to reconfigure a clause gadget, and six moves suffice if and only if a prong associated to a literal in the clause is part of a cycle.*

The six moves required by a clause gadget are in fact additional to the $20k_i + 24$ moves required to reconfigure the gadget for variable x_i and to the six moves required by any other clause gadget. If we allow only the minimum number of moves per gadget, Lemma 2 forces



■ **Figure 3** Rectilinear drawing of a PLANAR MONOTONE 3SAT instance. Our reduction attaches the variable gadgets horizontally and the clause gadgets next to the rightmost literal in the clause.



■ **Figure 4** An overview of the reduction. (Background colors match Figure 3).

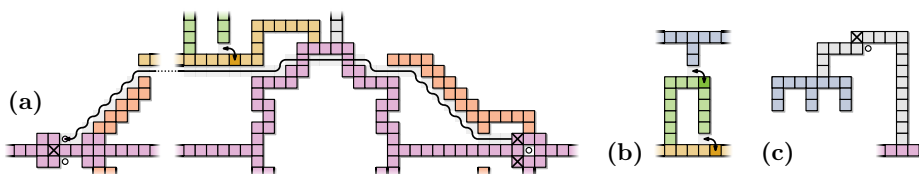
that in each variable gadget either the upper or the lower prongs become part of cycles. Moreover, Lemma 3 requires that for each clause there is a prong associated to a literal in the clause that becomes part of a cycle as part of the reconfiguration of the variable gadgets. This implies that if a reconfiguration sequence exists, the 3SAT instance must be satisfiable. In the other direction, if the 3SAT instance is satisfiable, then we show how to order the moves carefully to reconfigure with the minimum number of moves required.

► **Lemma 4.** *A PLANAR MONOTONE 3SAT instance can be solved if and only if the corresponding reconfiguration problem instance can be solved using $66m + 24l$ moves.*

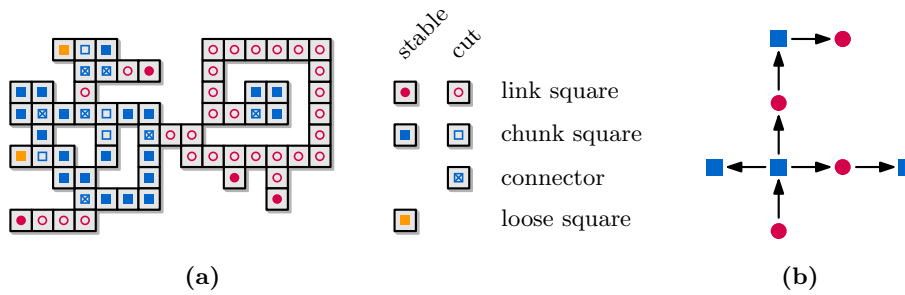
3 Input-sensitive in-place algorithm

To describe our input-sensitive reconfiguration algorithm, we first need to introduce the following definitions and notations. Let \mathcal{C} be an edge-adjacent configuration of squares on the rectangular grid and let G be the *edge-adjacency graph* of \mathcal{C} . In G each node represents a square and two nodes are connected by an edge, if the corresponding squares are edge-adjacent. With slight abuse of notation we identify the squares and the nodes in the graph. A square $s \in \mathcal{C}$ is a *cut square* if $\mathcal{C} \setminus \{s\}$ is disconnected. Otherwise, s is a *stable square*.

A configuration \mathcal{C} is *xy-monotone* if \mathcal{C} contains the entire leftmost column and bottommost row of \mathcal{C} 's bounding box, and each row or column of \mathcal{C} forms a single contiguous set of squares. A *chunk* is any inclusion-maximal set of squares in \mathcal{C} enclosed by (and including) a simple cycle σ in G of length at least 4 (its *boundary cycle*), plus any squares with degree 1 in G edge-adjacent to σ (its *loose squares*). A chunk constitutes a well-connected component that can be efficiently transformed towards an *xy-monotone* configuration.



■ **Figure 5** (a) Variable gadget. (b) Wire gadget. (c) Clause gadget.



■ **Figure 6** (a) A configuration \mathcal{C} . (b) The corresponding component tree T .

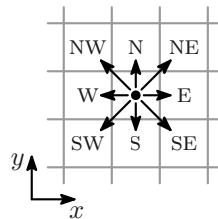
A *link* is a connected component of squares which are not in any chunk. A square is a *connector* if it is a chunk square edge-adjacent to a square in a link or in another non-overlapping chunk, or if it is the single overlapping square of two chunks. By definition a connector is always a cut square. The *size* of a chunk C is the number of squares contained in C (which includes its boundary cycle and any loose squares).

Figure 6a shows an example configuration with its chunks, links, connectors, and cut/stable squares marked. Note that a square can be part of two chunks simultaneously, in which case it must be a connector (for example, see the leftmost connector in Figure 6a). A chunk can contain both cut and stable squares.

The *component tree* T of \mathcal{C} has a vertex for each chunk/link and an edge (u, v) iff the chunks/links represented by u and v have edge-adjacent squares or share a square (when chunks are adjacent), see Figure 6b. The component containing the leftmost square in the bottom row of \mathcal{C} , the *root square*, is the root of T . Chunks in leaves of T we call *leaf chunks*.

A *hole* in \mathcal{C} is a finite maximal vertex-connected set of empty grid cells. The infinite vertex-connected set of empty grid cells is the *outside*. If a chunk C encloses a hole in \mathcal{C} , we say that C is *fragile*. Otherwise, we say that C is *solid*. The *boundary* of \mathcal{C} is the set of squares vertex-adjacent to any grid cell on the outside. The *boundary* of a hole H is the set of squares vertex-adjacent to any grid cell in H . Note that the boundary of a hole is edge-connected. We can construct T in $O(n)$ time by walking along the boundary of \mathcal{C} .

Consider now the bounding box B of \mathcal{C} on the square grid. We refer to the bottommost leftmost grid cell inside B as the *origin*. Let P be the perimeter of B , then any square in \mathcal{C} can be connected to the origin by an *xy-monotone* path of at most $P/2$ squares.



Let $c = (x, y)$ be a grid cell. We use compass directions (N, NE, E, etc.) to indicate neighbors of c . When we use grid coordinates, we assume the usual directions (the x -axis increases towards E and the y -axis increases towards N, so the N-neighbor of c is $(x, y + 1)$). Similarly, we indicate slide moves using compass directions (‘a W-move’) and convex transitions using a sequence of two compass directions (‘a WS-move’: a movement toward w followed by a movement towards s).

Algorithmic outline. In the first phase of our algorithm we ensure that the leaves of the component tree T are sufficiently large and well-connected. Specifically, we gather squares from the leaves of T until each leaf is a chunk of size at least P . In Section 3.1 we explain how to grow chunks using at most $O(P)$ moves per square that was moved. During this process, the final position of each square is chosen inside bounding box B , but squares can move through the layer of grid cells adjacent to B .

After gathering, all leaves are *heavy chunks* of size at least P . Our goal is now to make each leaf chunk contain the origin, while ensuring that all squares remain part of their chunk. A heavy leaf chunk C contains a sufficient number of squares to be transformed into a chunk containing both the connector of C and the origin: we can connect the connector with the origin by an xy -monotone path of at most $P/2$ squares; two such paths, which are disjoint, form a new boundary cycle for C . We do not explicitly construct these two paths, but instead we *compact* the configuration by filling holes and using lexicographically monotone movement towards the origin for squares in heavy leaf chunks. In Section 3.2 we explain the details of the compaction algorithm and prove that it leaves us with a solid xy -monotone component.

During compaction each square in a leaf chunk makes only lexicographic monotone moves towards the origin while staying inside B : s- and w-moves (slides), as well as sw-, ws-, nw-, and wn-moves (convex transitions). In some cases, a square in the leftmost column or bottom row can exit B , and move along the bounding box to enter the same column/row again closer to the origin. This is the only time a non-lexicographic monotone move is used, and every square can perform it at most $O(P)$ times. Hence compacting takes $O(Pn)$ moves.

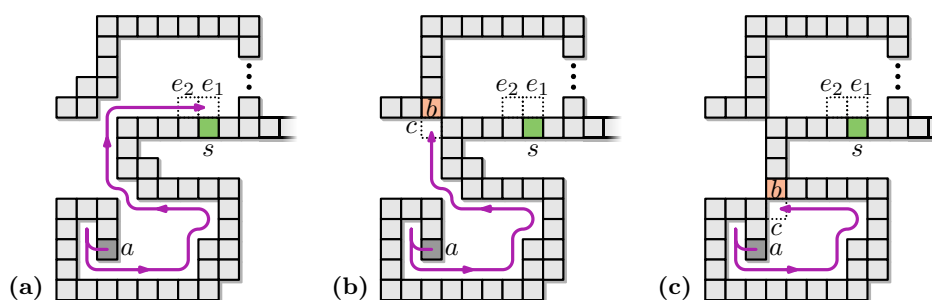
When compacting, every (heavy) leaf chunk will eventually contain a square at the origin. This means that the whole configuration becomes a single chunk, as all leaves of the component tree have merged into a single component. Therefore, once compacting has finished we arrive at an xy -monotone configuration that fits inside B . If at any point during this process the configuration becomes xy -monotone, then we simply stop. In particular, if the configuration is xy -monotone at the start, for example squares in only a single row or column, then we do not have to gather or compact, even though there are no heavy chunks. See the top row of Figure 2 for a visual impression of our algorithm.

In the special case that the input configuration \mathcal{C} contains less than P squares, we first ensure that \mathcal{C} contains the origin and then execute the gathering and compaction as before. The number of moves is trivially bounded by $O(Pn) = O(P^2)$, see Section 3.4.

Finally, in Section 3.3 we show how to convert any xy -monotone configuration into a different xy -monotone configuration with at most $O(\bar{P}n)$ moves, where \bar{P} is the maximum perimeter of the bounding boxes of source and target configurations. Thus, since all moves are reversible, we can transform the source into the target configuration via this transformation.

► **Theorem 5.** *Let \mathcal{C} and \mathcal{C}' be two configurations of n squares each, let P and P' denote the perimeters of their respective bounding boxes, and let $\bar{P} = \max\{P, P'\}$. We can transform \mathcal{C} into \mathcal{C}' using at most $O(\bar{P}n)$ moves while maintaining edge-connectivity at all times.*

Proof. For any two configurations \mathcal{C} and \mathcal{C}' of n squares each, we can apply gathering and compacting to find xy -monotone configurations M in $O(Pn)$ and M' in $O(P'n)$ moves, for \mathcal{C} and \mathcal{C}' respectively. If we want to transform \mathcal{C} into \mathcal{C}' , we first gather and compact \mathcal{C} into M , transform M into M' in $O(\bar{P}n)$ moves, and proceed by reversing the sequence of steps for \mathcal{C}' to get configuration \mathcal{C}' . In Sections 3.1, 3.2 and 3.3 we show that gathering, compacting and transforming xy -monotone configurations require the appropriate number of moves, such that the total number of moves is $O(Pn + \bar{P}n + P'n) = O(\bar{P}n)$. ◀



■ **Figure 7** Light square s (green); filling cells e_1 and e_2 makes s part of a chunk; a stable square a (dark grey) moves towards e_1 along the boundary. **(a)** a reaches e_1 . **(b)** square b (brown) part of a component outside of D , moving a to c creates a chunk containing s . **(c)** square b part of a component in D , moving a to c creates a hole; its inner boundary will not be traversed again.

3.1 Gathering

In this section we show how to gather squares from the leaves of the component tree T until we create a chunk of size at least P that is a leaf of T . In the following, let s be a connector or a cut square in a link. By definition, s lies on the boundary of \mathcal{C} . Since s is a cut square, removing s from \mathcal{C} results in at least two connected components. One of these components contains s from the root of T . We say that the other (up to three) components are *descendants* of s . Let D be the set of squares in the descendant components of s . We say that the *capacity* of s is $|D|$, and that s is *light* if its capacity is less than P and *heavy* otherwise.

► **Lemma 6.** *Let s be a light square with descendant squares D . Then s can be made part of a chunk with a sequence of $O(P)$ moves by squares in D . This procedure is in-place.*

Proof. Observe that there exist up to two empty cells e_1 (and e_2) neighboring s , such that moving squares there results in a chunk component containing s (see Figure 7). Cell e_1 (and e_2) can be chosen such that they lie inside bounding box B : these cells must exist since always at least three neighboring cells are inside B , unless B is a single row/column and the configuration was already *xy-monotone*. If such cells are already occupied by squares then s is already in a chunk. We argue that we can move squares from the descendant components of s into these empty cells with at most $O(P)$ moves. Once this is accomplished, we repeat the process in the descendant components, for the next light square of maximal capacity, until no light squares remain in the component tree below the chunk containing s .

Let $D' \subseteq D$ be a subset of boundary squares in the descendants of s of the subconfiguration $D \cup \{s\}$. Select an arbitrary stable square $a \in D'$. Such a square exists because of the following: if there is a link component in D that is a leaf in the component tree, then its degree-1 node is stable; and if there is a chunk component in D that is a leaf in the component tree, then an extremal square of the chunk in one of the NE, NW, SE, or SW directions is stable (only one of them can be a connector square).

Consider moving a along the boundary of D towards e_1 . Let E_a be the set of cells that a needs to cross to reach e_1 . If E_a is empty, then we simply move a to e_1 (see Figure 7a), and repeat the procedure for e_2 (if it exists). In this case, a takes $O(P)$ moves to get to e_1 , since a can take a simple path along the at most P descendants in D .

Now consider the case where E_a is not empty. Let b be the first square in E_a on the way from a to e_1 ; let c be the square in E_a that is just before and edge-adjacent to b . As b is not part of the boundary along which the path from a to e_1 is considered, it must be vertex-adjacent to a square that is on that part of the boundary.

There can be two cases: either $b \notin D$ or $b \in D$. In the first case, moving a to c merges a component in D with some component outside of D (see Figure 7b). Thus, a chunk is created that contains s , resulting in s no longer being a light square.

In the second case, when $b \in D$, moving a to c creates a chunk within D (see Figure 7c). In this case we select another arbitrary stable square a' in the new subconfiguration $D \cup \{s, c\} \setminus a$, and repeat the procedure. Observe that the empty squares traversed by a are now part of a hole in the new chunk. Thus the path from a' to e_1 does not overlap with the path taken by a . Let $\{a_0, a_1, a_2, \dots\}$ be the sequence of such stable squares chosen by our algorithm as candidates to be moved to e_1 . For any square a_i , its path along the boundary to e_1 does not overlap with any of the cells traversed by all a_j with $j < i$. Thus, there is some k such that a_k either reaches e_1 , or it merges the components within and outside of D into a chunk containing s . The total number of empty cells traversed by all squares a_i ($0 \leq i \leq k$) is $O(P)$.

We repeat the above procedure to fill e_2 . Note that the path taken by a (and a') may not always be inside bounding box B . When this path exits B , it will always stay adjacent to cells in \mathcal{C} , and hence it will use only the single row/column of cells adjacent to B . ◀

Using the procedure described in Lemma 6, we can iteratively reduce the number of light squares to obtain a component tree where all leaves are chunks of size at least P : any light square with capacity $P - 1$ will form a chunk of size P with all its descendants.

► **Lemma 7.** *An in-place reconfiguration of $O(Pn)$ moves exists, which ensures that all leaves in the component tree are chunks of size at least P .*

Proof. By Lemma 6, we can make a light square s part of a chunk in $O(P)$ moves by moving cubes from D , the set of descendants of s . This in-place process creates new light squares only if removal of a square a breaks a cycle in D . Thus, every new light square is part of D .

We repeat the procedure, selecting a light square of maximal capacity at every step. Overall, a square can be light at most once in the process. Thus, after $O(Pn)$ moves no light squares remain, and all the leaves in the component tree are chunks of size at least P . Note that, while the root square always has capacity $n - 1$, an adjacent square can have capacity $2 \leq |D| \leq P$, and the resulting chunk will be the root component, either because the root square is on the boundary cycle, or is a loose square. In case the adjacent square is too light, namely $|D| < 2$, then the root component may stay a link. ◀

3.2 Compacting

After the gathering phase, each leaf of the component tree T is a heavy chunk, that is, each leaf is a chunk of size at least P . In this section, we describe how we compact the configuration in order to turn it into a *left-aligned histogram* containing the origin. A left aligned histogram has a vertical base, and extends only rightward. Our procedure uses three types of moves: LM-moves, corner moves, and chain moves, which we discuss below. We iteratively apply any of these moves. The correctness of the algorithm does not depend on the order in which moves are executed, as long as the moves are *valid* (defined below). Our implementation assigns priorities to the various types of moves and chooses according to these priorities whenever multiple moves are possible (see Section 4).

LM-moves. We say that a move is a *lexicographically monotone move* (LM-move for short) if it is either an s- or w-move (slides), or an SW-, WS-, NW-, or WN-move (convex transitions). Note that an LM-move will never move a square to the east. Squares can move to the north, but only when they also move to the west. Hence, if a square starts at coordinate (x, y) , and we perform a series of LM-moves, it stays in the region $\{(x', y') \mid x' \leq x \wedge y' \leq x - x' + y\}$.

Let s be a square in a heavy leaf chunk C of \mathcal{C} , and consider an LM-move made by s . We say that this move is *valid* if s stays inside bounding box B , and all squares $s' \in C$ are still in a single chunk after the move. While compacting, we allow only valid LM-moves, that is, we allow each chunk to grow, but a chunk can never lose any squares.

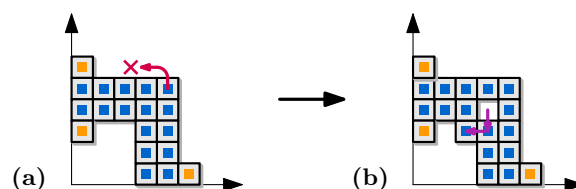
Corner moves. LM-moves on their own are not necessarily sufficient to compact a chunk into a suitable left-aligned shape. For example, consider the configuration in Figure 8a, which does not admit any valid LM-moves. However, it has a concave corner that we can fill with two moves (see Figure 8b), to expand the chunk in that direction. Repeating such corner moves allows us to make the chunk in the example left-aligned.

We define corners of a chunk C with boundary cycle σ as follows. A *top corner* (Figure 9a–d) is an empty cell with squares $b_1, b_2, b_3 \in \sigma$ as N-, NE-, and E-neighbors. Similarly, a *bottom corner* (Figure 9e–h) is an empty cell with squares $b_1, b_2, b_3 \in \sigma$ as S-, SE-, and E-neighbors. Note that a corner can be either inside a hole in C (*internal corner*), or on the outside of C (*external corner*), and we treat both of these in the same way.

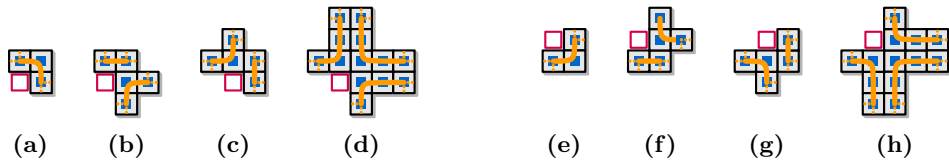
Let s be a top corner in C with neighbors b_1, b_2, b_3 as above. In the case where b_1, b_2, b_3 are consecutive squares in σ (Figure 9a), we can fill s by two slide moves: first move either b_1 or b_3 into s , and then move b_2 into the cell left empty by the first move. We call this a *top corner move*. We can fill a bottom corner with consecutive b_1, b_2, b_3 (Figure 9e) in the same way, just mirrored vertically (a *bottom corner move*). Just like for LM-moves, we say that a corner move is *valid* if all squares $s' \in C$ are still in a single chunk after the corner move. Note that all corners where b_1, b_2, b_3 are not consecutive in σ (Figure 9b–d and 9f–h) do not allow valid corner moves, as b_1 and/or b_2 becomes a connector.

Chain moves. Besides LM- and corner moves, we need a special move to prevent getting stuck when each LM-move is invalid because it would move outside the bounding box B . For example, some squares on the bottom row or leftmost column of B would be able to perform LM-moves if they were situated in any other row/column of B , as shown in Figure 10.

A *chain move* is a series of moves that is started by such an LM-move that violates validity only by leaving bounding box B . A chain move for a square s in the bottom row of B requires an empty cell $e = (x, 0)$ closer to the origin, and works as follows. Square s must be able to perform an LM-move, more precisely an SW-move that is invalid only because it leaves B . We want to place s in this empty cell e , unless it creates a link component, which happens only if the square on position $(x + 1, 0)$ is a loose square. We slide such a loose square upwards with a N-move, and identify the emptied cell as e . Note that e is again the closest empty cell in the bottom row, closer to the origin. We can then move s to e by performing an SW-move, a series of W-moves, and finally a WN-move into e . For a square s in the leftmost column, the direction of all moves is mirrored in $x = y$.



■ **Figure 8** (a) A configuration that does not admit LM-moves. For example, an NW-move (in red) of the top-right square is not valid. (b) Two slide moves expand the concave corner in SW direction.



■ **Figure 9** Empty squares shown in red are corners. The boundary cycle of the chunk is shown in orange. (a)–(d) Top corners; (e)–(h) bottom corners. Corners shown in (a) and (e) can be both external and internal. Corners shown in (b)–(d) and (f)–(h) can only be internal.

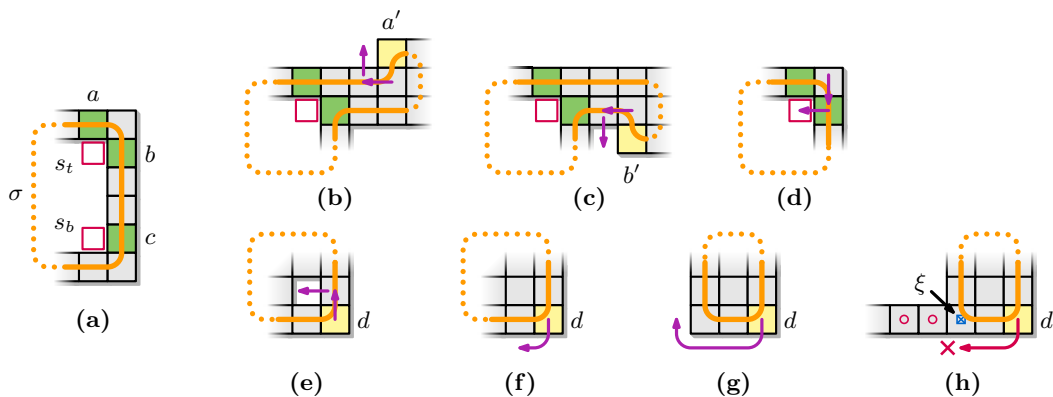


■ **Figure 10** Examples of chain moves: a square moves outside B to the first empty cell closer to the origin. This may require a loose square to move as well.

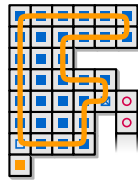
► **Lemma 8.** *Let C be a leaf chunk that does not admit valid LM-moves, corner moves, or chain moves. Then C is solid, and its boundary cycle σ outlines a left-aligned histogram.*

Proof. We first show that C is solid. Assume to the contrary that C has a hole. Consider the top- and bottommost empty squares s_t and s_b of the rightmost column of empty squares of any hole in C (s_t may be equal to s_b). Let a and b be the N- and E-neighbors of s_t , respectively, and let c be the E-neighbor of s_b (see Figure 11a). We know that $a, b, c \in \sigma$, because otherwise moving a or b into s_t , or c into s_b , would be a valid LM-move. C has at most one connector ξ , which is part of σ . We now show that ξ lies strictly between a and b on σ , and also that ξ lies strictly between c and a , to arrive at a contradiction.

Let σ' be the part of σ strictly between a and b , walking along the boundary of C in the clockwise order. If σ' visits the row above a (leaving the row of a for the first time at square a'), then there exists a bottom corner move filling the w-neighbor of a' (see Figure 11b). This move is valid since the part of C to the right of s_t by definition does not contain any holes. Similarly, if σ' visits the row below b (returning to the row of b for the last time at square b'), then there exists a top corner move filling the w-neighbor of b' (see



■ **Figure 11** Lemma 8: (a) A chunk hole with s_t and s_b marked. (b)–(d) Possible valid moves for σ' . (e)–(g) Possible valid moves for σ'' . (h) If a chain move is not possible then ξ is left of d .



■ **Figure 12** The boundary cycle σ of the leaf chunk C outlines a left-aligned histogram.



■ **Figure 13** The chunk C forms a double- Γ shape, with one or two loose squares (orange).

Figure 11c). We conclude that the part of S attached to a and b is a protrusion of two rows tall. The protrusion is non-empty, because otherwise we can either move a loose square with an LM-move or fill s_t with a corner move (see Figure 11d). Consider the rightmost column of the protrusion. If this column contains a loose square, we can move it with an LM-move. Hence, the column contains a square in the row of a and a square in the row of b . If neither of these is ξ , then we can perform an LM-move. Hence, σ' contains ξ .

Let σ'' be the part of σ strictly between c and a in clockwise order. Walk over σ'' until encountering the first square d whose N- and W-neighbors are adjacent to d on σ . Assume that ξ is not d or one of its neighbors. If d or the N-neighbor of d have a loose square attached to them, we can perform an LM-move on this loose square. Otherwise, if the NW-neighbor of d is part of a hole in C , then this hole can be filled with a bottom corner move (see Figure 11e). Otherwise, there are three cases: either (1) we can perform an S- or SW-move on d (see Figure 11f), or (2) we can perform a horizontal chain move (see Figure 11g), or (3) the chain move is impossible because no empty square e is available to move to (see Figure 11h). In cases (1) and (2), by contradiction ξ is d or one of its neighbors; in case (3), ξ needs to be in the bottommost row. In any case, σ'' contains ξ .

As σ' and σ'' are disjoint, they cannot both contain ξ , which results in a contradiction. Hence, C is solid. To show that σ outlines a left-aligned histogram, we observe that any external corner (with neighbors b_1, b_2, b_3 as defined above) admits a valid corner move. Indeed, none of b_1, b_2, b_3 can be a loose square, as those would admit LM-moves. Furthermore, as C is solid, a boundary square cannot be a cut square for squares on the inside of σ . Finally, since C is a leaf chunk, the only cut square in σ is its connector. As only one out of b_1 and b_3 can be this connector, we can perform a valid corner move starting with the other (non-connector) square. Therefore, by our assumption that there are no corner moves in C , there cannot be external corners, and thus σ outlines a left-aligned histogram (see Figure 12). ◀

A set of squares is a *double- Γ* if it fills the top two rows and left two columns of its bounding box.

► **Lemma 9.** *Let C be a leaf chunk that does not contain the origin and does not admit valid LM-moves, corner moves, or chain moves. Then the squares of the boundary cycle of C are a double- Γ (see Figure 13).*

Proof. Let C^* be the set of squares outlined by σ . By Lemma 8, C^* is a left-aligned histogram. Let $\{r_1, r_2, \dots\}$ be the rows in C^* , ordered from top to bottom. The connector ξ of C lies on σ , and thus in C^* . Assume that ξ is in row r_i ($i \geq 3$). In that case, the leftmost square of r_1 has a valid move m : a WS-move or a vertical chain move. In particular, m is not blocked by a loose square in the leftmost column of C , because that column can contain only one loose square in the last row of C^* (otherwise one of the other loose squares would admit a valid S-move). Similarly, C cannot contain a loose square, which could block m , in its topmost row. Indeed, such a loose square would admit a W-, WS-move, or a vertical

chain move (if it has an s-neighbor in σ), or it would admit an s-move (if it has a w- or e-neighbor in σ). The existence of m leads to a contradiction, so ξ lies on r_1 or r_2 . Because C^* is 2-connected, this implies that $|r_1| = |r_2|$, forming the horizontal leg of the double- Γ .

Consider the last row r_k ($k \geq 3$) such that $|r_k| > 2$. The rightmost square in r_k has a valid sw-, s-move, or horizontal chain move (which, by a similar argument as before, cannot be blocked by a loose square). On the other hand, $|r_k| \geq 2$, again because of 2-connectivity. Therefore, for all rows r_i ($i \geq 3$), $|r_i| = 2$, forming the vertical leg of the double- Γ . ◀

► **Lemma 10.** *Let \mathcal{C} be a configuration in which each leaf is a chunk of size at least P . Let \mathcal{C}' be the result of exhaustively performing valid LM-moves, corner moves, or chain moves on \mathcal{C} . This reconfiguration is in-place and \mathcal{C}' is xy -monotone.*

Proof. In the compaction phase we iteratively apply LM-, corner, and chain moves on the configuration in which each leaf chunk contains at least P squares. After compaction, a leaf chunk can either contain the origin, or not. Consider such a chunk C that does not contain the origin. By Lemma 9, the cycle of C will outline a double- Γ , and since C is a leaf, it has at least P squares. Gathering and compacting are in-place, since squares always move to a cell inside the initial bounding box B of the configuration. Even if a square moves outside B during gathering or chain moves it always ends inside B (by Lemma 6 and by definition, respectively). Hence the connector of C will also be inside B . Since P is the perimeter of B , any double- Γ of P squares completely inside B will reach the bottom left corner of B . Thus, C must contain the origin, as one of the top two rows connects to the connector inside B . As a result, every leaf chunk contains the origin at some point during compaction.

Once every leaf chunk contains the origin, the whole configuration is one single chunk: all leaves of the component tree form a single component now. Continuing the compacting hence results in a left-aligned histogram, by Lemma 8. Finally consider the topmost row r of this histogram that is longer than the row below it. During compaction, the rightmost cube of r can perform a valid LM-move, namely a s- or sw-move to the row below it. Note that these moves cannot put cubes outside of the original bounding box B of \mathcal{C} . Thus, once the compacting phase is completed, the configuration is xy -monotone inside B . ◀

► **Lemma 11.** *There is an in-place reconfiguration of $O(Pn)$ moves of a configuration in which all leaves are chunks of size at least P to an xy -monotone configuration.*

Proof. Using Lemmata 8, 9, and 10, we can transform a configuration \mathcal{C} , in which all leaves are chunks of size at least P , to an xy -monotone configuration. Let $s = (x, y)$ be a square in \mathcal{C} . We assign to s the score $d(s) = 2x + y$, and let $d = \sum_{s \in \mathcal{C}} d(s)$. Each LM-, and bottom corner move performed in \mathcal{C} decreases d by at least 1, while every top corner move decreases d by two. Initially, $d \leq |\mathcal{C}| \cdot P$, so the total number of LM- and corner moves is also at most $|\mathcal{C}| \cdot P$. Every square s is involved in at most $P/2$ chain moves, since each chain move places s closer to the origin in the bottom row/leftmost column. Furthermore, every chain move adds at most one additional move for a loose square, which increases the above score by at most two, hence the total number of moves as a result of chain moves is also at most $O(|\mathcal{C}| \cdot P)$. ◀

3.3 Transforming xy -monotone configurations

After gathering and compacting we arrive at an xy -monotone configuration. However, this configuration is not unique and hence we need to be able to transform between such configurations. We use a potential function to guide this transformation.

► **Lemma 12.** *Let \mathcal{C}_1 and \mathcal{C}_2 be two xy -monotone configurations of n squares each, let P and P' denote the perimeters of their respective bounding boxes, and let $\bar{P} = \max\{P, P'\}$. We can reconfigure \mathcal{C}_1 into \mathcal{C}_2 using at most $O(\bar{P}n)$ moves, while remaining in-place.*

Proof. Let $\mathcal{C} := \mathcal{C}_1$. For each grid cell $c = (x, y)$, we define the *potential* of c to be $\phi(c) = x + y$. Let s be the bottommost square in $\mathcal{C} \setminus \mathcal{C}_2$ whose cell has maximum potential, and let e be the topmost empty grid cell with minimum potential that is occupied in \mathcal{C}_2 . We iteratively move s to e in \mathcal{C} until $\mathcal{C} = \mathcal{C}_2$. We first show that \mathcal{C} remains xy -monotone. Removing s cannot break this property: by definition of ϕ and since \mathcal{C}_2 is xy -monotone, s does not have N-, NE-, or E-neighbors in \mathcal{C} . Moreover, if it has a NW-neighbor then, by xy -monotonicity of \mathcal{C} , it has a W-neighbor too. Similarly, adding a square in e maintains xy -monotonicity. By the definition of ϕ and since \mathcal{C}_2 is xy -monotone, the cells neighboring e in the s, sw, and w directions must be occupied if they are inside the bounding box of \mathcal{C} . Moreover, xy -monotonicity of \mathcal{C} guarantees that e does not have N-, NE-, or E-neighbors.

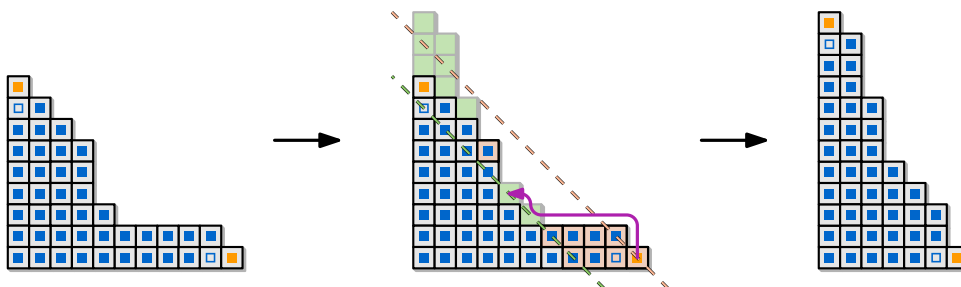
In every step we move a square from a position occupied in \mathcal{C}_1 to a position occupied in \mathcal{C}_2 , hence the perimeter of the bounding box of configuration \mathcal{C} is $O(P + P') = O(\bar{P})$. Moving one square along the boundary of \mathcal{C} is allowed for in-place reconfiguration. Since this takes at most $O(\bar{P})$ moves per square and no square is moved more than once, it takes $O(\bar{P}n)$ moves in total to reconfigure \mathcal{C}_1 into \mathcal{C}_2 . ◀

3.4 Light configurations

We say that a configuration \mathcal{C} is *light*, if it consists of fewer than P squares, where P is the perimeter of the bounding box of \mathcal{C} . Our algorithm, as explained in the main text, cannot directly handle such configurations if \mathcal{C} does not contain the origin: there are too few squares to guarantee that compacting will always result in a chunk that contains the origin. However, we can use a simple preprocessing step to ensure that \mathcal{C} will contain the origin.

For a light configuration \mathcal{C} which does not contain the origin, we select a stable square as in the gathering phase: a stable square in a link, or an extremal stable square in a chunk. We iteratively move this stable square along the boundary of \mathcal{C} to the empty cell e that is the w-neighbor of the root square. We iteratively continue to do so until \mathcal{C} contains the origin. Note that e must necessarily be empty.

At this point, we can simply gather and compact \mathcal{C} and arrive at an xy -monotone configuration, for the following reason. The gathering phase works as in the main text, since we can iteratively apply Lemma 6 on the light square closest to the root (which can be the root itself), to get a single chunk. As the root square is located at the origin, and we never move the root during gathering, we get a chunk containing the origin. Similarly, in



■ **Figure 14** Transforming between two xy -monotone configurations. The dashed lines go through cells with the same potential.

the compaction phase, this chunk will become a solid left-aligned histogram by Lemma 8. Since it already contains the origin, and we do only monotone moves towards the origin, this chunk still contains the origin. Finally the topmost row r of this histogram, that is longer than the row below it, still has valid LM-moves. Hence \mathcal{C} is xy -monotone after compaction.

There are at most $P/2$ empty cells to the left of the root. We fill each of these cells by walking along the boundary of \mathcal{C} . Since configuration \mathcal{C} consists of less than P squares, this requires at most $O(Pn) = O(P^2)$ moves. Both gathering and compacting take $O(Pn)$ moves, as proven in the main text, so including the preprocessing, we still arrive at a bound of $O(Pn)$ for the number of moves.

4 Experiments

We experimentally compared our Gather&Compact algorithm to the JavaScript implementation¹ of the in-place modification by Moreno and Sacristán [14, 15] of the Dumitrescu and Pach [10] algorithm, which we refer to as MSDP in the remainder of this section. The original algorithm by Dumitrescu and Pach always requires $\Theta(n^2)$ moves, since it builds a horizontal line to the right of a rightmost square as canonical shape. The in-place modification of Moreno and Sacristán has the potential to be more efficient in practice, since it builds a rectangle within the bounding box of the input.

We captured the output (sequence of moves) of MSDP and reran the reconfiguration sequences in our tool, to be able to verify movement sequences, count moves, and generate figures. Doing so, we discovered that MDSP was occasionally executing illegal moves, see the full version for details and for our corresponding adaptations [3]. Some of these issues could be traced to the same origin: MSDP is breaking convex transitions into two separate moves and sometimes acts on the illegal intermediate state. The number of moves we report in Table 1 counts one move both for convex transitions and for slides; hence the numbers can be lower than the numbers Moreno and Sacristán report. However, our adaptations do replace illegal moves with the corresponding (and generally longer) legal movement sequences, and hence the number of moves can also be higher than those they report.

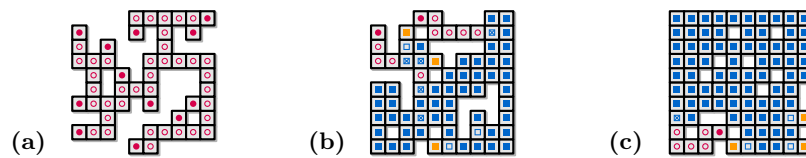
We use square grids of sizes 10×10 , 32×32 , 55×55 , 80×80 , 100×100 for our experiments. The data sets for MSDP were created by hand and are not available.² We attempted to create meaningful data sets of the same nature by starting with a fully filled square grid and

¹ <https://dccg.upc.edu/people/vera/TFM-TFG/Flooding/>

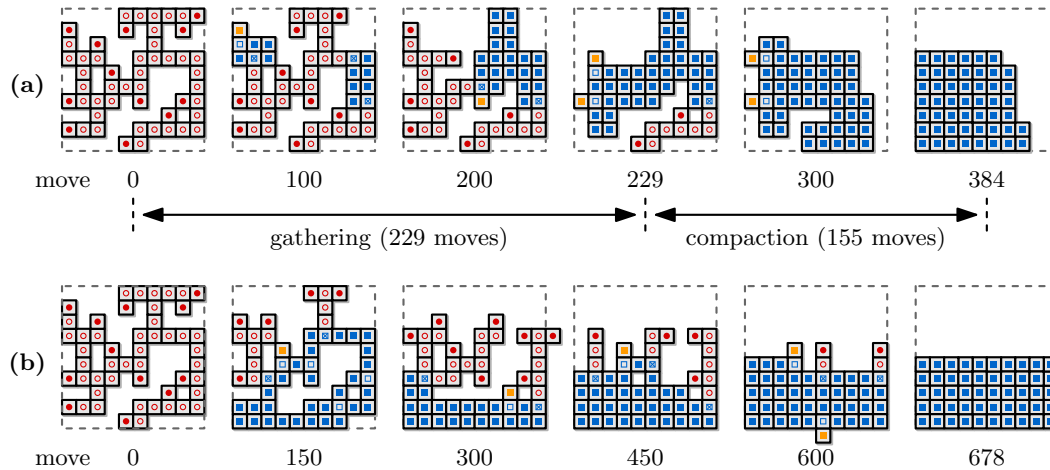
² V. Sacristán, personal communication, April 2021.

■ **Table 1** The number of moves for Gather&Compact and MSDP on various grid sizes ($D \times D$, such that $P = 4D$) and densities (in % of $D \times D$). Averages and standard deviations (in % of average) over 10 randomly generated instances are shown.

| D | Gather&Compact | | | MSDP | | |
|-----|----------------|------------|-----------|---------------|-------------|------------|
| | 50% | 70% | 85% | 50% | 70% | 85% |
| 10 | 237 31% | 156 16% | 95 8% | 502 19% | 427 21% | 233 35% |
| 32 | 5.395 4% | 4.188 5% | 2.529 8% | 28.759 12% | 18.447 13% | 10.027 8% |
| 55 | 25.916 2% | 20.024 3% | 12.124 4% | 193.390 8% | 116.431 12% | 61.617 8% |
| 80 | 77.745 2% | 60.516 2% | 36.395 3% | 638.847 12% | 344.529 9% | 235.413 5% |
| 100 | 150.666 1% | 118.232 2% | 69.488 3% | 1.318.232 11% | 743.133 17% | 513.113 7% |



■ **Figure 15** Example input instances on a 10×10 grid: density (a) 50%; (b) 70%; (c) 85%.



■ **Figure 16** Execution of the two algorithms on one of the input instances for grid size 10×10 , density 50%. (a) Gather&Compact; (b) MSDP. Video: <https://tinyurl.com/alga10x10>.

then removing varying percentages of squares while keeping the configuration connected. We arrived at three densities, namely (50%, 70%, 85%), which arguably capture the different types of inputs well (see Figure 15). For each value, the density of the configurations generated is close to homogeneous. The configurations with 85% density are a generalization of the “dense” configurations in the data sets for MSDP. The configurations with 70% density correspond to the so-called “medium” configurations in the data sets for MSDP, which combine the two different substructures considered for that density. The edge-adjacency graphs of the configurations with 50% density are essentially trees and, especially in the larger configurations, many leaves are not on the outer boundary (resembling the “nested” configurations in the evaluation of MSDP). For both algorithms we count moves until they reach their respective canonical configurations. Our online material³ contains our code for Gather&Compact, the input instances, and the adapted version of MSDP.

The compaction step of Gather&Compact does not rely on any particular order of the available valid moves. Our implementation prioritizes squares by descending L_∞ -distance to the origin. We also prioritize downwards LM-moves (w, ws, sw, s) over upwards LM-moves (WN, NW), and top corner moves over bottom moves.

Table 1 summarizes our results and Figure 16 shows snapshots for both algorithms on a particular instance. We observe that Gather&Compact always uses significantly fewer moves than MSDP, even on high density instances where most squares are already in place. This is likely due to the fact that MSDP walks squares along the boundary of the configuration, while Gather&Compact shifts squares locally into better position. Figure 16b shows this behavior at move 600 where one can observe a square on its way along the bottom boundary.

³ <https://alga.win.tue.nl/software/compacting-squares/>

5 Conclusion

We introduced the first universal in-place input-sensitive algorithm to solve the reconfiguration problem for the sliding cube model in two dimensions. Our Gather&Compact algorithm is input-sensitive with respect to the size of the bounding box of the source and target configurations. We experimentally established that Gather&Compact not only improves the existing theoretical bounds, but that it also leads to significantly fewer moves in practice.

We showed that minimizing the number of moves for reconfiguration is NP-complete in two dimensions. The question then arises whether the problem admits approximation algorithms. Our NP-hardness proof can be adapted to show APX-hardness in the 3D sliding cube model and we conjecture that the problem is also APX-hard for sliding squares.

There may still be room to improve on the algorithm in this paper. Specifically, it may be possible to improve the hidden constants, by gathering to leaf chunks of size $P/2$ instead of P . These chunks still have enough squares to reach the origin, but they have to give up 2-connectivity to do so, and hence the algorithm becomes more complex. Once all leaves create an xy -monotone path to the origin, the configuration again consists of a single chunk, and thus the remaining parts of our algorithm still apply.

Finally, extending our algorithm to three dimensions is currently work in progress. While well-connected components can also be transformed more efficiently in 3D, the algorithm may require a higher degree of connectivity than 2-connectivity.

References

- 1 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.
- 2 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Dylan H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Kortén, Jayson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *Proc. 37th International Symposium on Computational Geometry (SoCG)*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.SoCG.2021.10.
- 3 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. *CoRR*, abs/2105.07997, 2021. arXiv:2105.07997.
- 4 Greg Aloupis, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Robin Flatland, John Iacono, and Stefanie Wuhler. Efficient reconfiguration of lattice-based modular robots. *Computational Geometry: Theory and Applications*, 46(8):917–928, 2013. doi:10.1016/j.comgeo.2013.03.004.
- 5 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O’Rourke, Val Pinciu, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhler. Efficient constant-velocity reconfiguration of crystalline robots. *Robotica*, 29(1):59–71, 2011. doi:10.1017/S026357471000072X.
- 6 Byoung Kwon An. EM-Cube: Cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Proc. 2008 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3149–3155, 2008. doi:10.1109/ROBOT.2008.4543690.
- 7 Nora Ayanian, Paul J. White, Ádám Hálász, Mark Yim, and Vijay Kumar. Stochastic control for self-assembly of XBots. In *Proc. ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE)*, pages 1169–1176, 2008. doi:10.1115/DETC2008-49535.

- 8 Nadia M. Benbernou. Geometric algorithms for reconfigurable structures. PhD thesis, Massachusetts Institute of Technology, 2011.
- 9 Chih-Jung Chiang and Gregory S. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10:91–106, 2001. doi:10.1023/A:1026552720914.
- 10 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22:37–50, 2006. doi:10.1007/s00373-005-0640-1.
- 11 Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proc. 2003 IEEE/RSJ International Conference on Intelligent Robots and System*, pages 2460–2467, 2003. doi:10.1109/IR0S.2003.1249239.
- 12 Kazuo Hosokawa, Takehito Tsujimori, Teruo Fujii, Hayato Kaetsu, Hajime Asama, Yoji Kuroda, and Isao Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. 1998 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2858–2863, 1998. doi:10.1109/ROBOT.1998.680616.
- 13 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019. doi:10.1016/j.jcss.2018.12.001.
- 14 Joel Moreno. In-place reconfiguration of lattice-based modular robots. Bachelor’s thesis, Universitat Politècnica de Catalunya, 2019.
- 15 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *Proc. 36th European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020.
- 16 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module design for efficient reconfiguration of modular robots. *Autonomous Robots*, 45(4):457–472, 2021. doi:10.1007/s10514-021-09977-6.
- 17 Daniela Rus and Marsette Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proc. 2000 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1726–1733, 2000. doi:10.1109/ROBOT.2000.844845.
- 18 John W. Suh, Samuel B. Homans, and Mark Yim. Telecubes: mechanical design of a module for self-reconfigurable robotics. In *Proc. 2002 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4095–4101, 2002. doi:10.1109/ROBOT.2002.1014385.
- 19 Cynthia Sung, James Bern, John Romanishin, and Daniela Rus. Reconfiguration planning for pivoting cube modular robots. In *Proc. 2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1933–1940, 2015. doi:10.1109/ICRA.2015.7139451.
- 20 Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007. doi:10.1109/MRA.2007.339623.

Fault-Tolerant Edge-Disjoint $s-t$ Paths – Beyond Uniform Faults

David Adjiashvili ✉

Department of Mathematics, ETH Zürich, Switzerland

Felix Hommelsheim ✉ 

Department of Mathematics and Computer Science, Universität Bremen, Germany

Moritz Mühlenthaler ✉ 

Laboratoire G-SCOP, Grenoble INP, Univ. Grenoble-Alpes, France

Oliver Schaudt

Department of Mathematics, RWTH Aachen University, Germany

Abstract

The Edge-disjoint $s-t$ Paths Problem ($s-t$ EDP) is a classical network design problem whose goal is to connect for some $k \geq 1$ two given vertices of a graph under the condition that any $k - 1$ edges of the graph may fail. We extend the simple uniform failure model of the $s-t$ EDP as follows: the edge set of the graph is partitioned into *vulnerable*, and *safe* edges, and a set of at most k vulnerable edges may fail, while safe edges do not fail. In particular we study the *Fault-Tolerant Path* (FTP) problem, the counterpart of the Shortest $s-t$ Path problem in this non-uniform failure model as well as the Fault-Tolerant Flow (FTF) problem, the counterpart of $s-t$ EDP. We present complexity results alongside exact and approximation algorithms for both problems. We emphasize the vast increase in complexity of the problems compared to $s-t$ EDP.

2012 ACM Subject Classification Theory of computation → Routing and network design problems; Theory of computation → Network flows; Mathematics of computing → Approximation algorithms

Keywords and phrases graph algorithms, network design, fault tolerance, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.5

Related Version *Full Version*: <https://arxiv.org/abs/2009.05382>

1 Introduction

The *Minimum-Cost Edge-Disjoint $s-t$ Path Problem* ($s-t$ EDP) is a classical network design problem defined as follows. Given an edge-weighted directed graph $D = (V, A)$, two terminal vertices $s, t \in V$ and an integer parameter $k \in \mathbb{Z}_{\geq 0}$, find k edge-disjoint paths connecting s and t with minimum total cost. Equivalently, the problem $s-t$ EDP asks for the minimum cost of connecting two nodes in a network, given that any $k - 1$ edges can “fail” and hence be a-posteriori removed from the graph. The assumption here is that faults are *uniform* in the sense that every edge in the graph is equally vulnerable. Our goal is to advance the understanding of network design problems in the presence of *non-uniform* faults. To this end we study a natural generalization of $s-t$ EDP called the *Fault-Tolerant Path* (FTP) problem, in which we partition the set of edges into *vulnerable* and *safe* edges. The task is to find a minimum-cost subgraph of a given graph that contains an $s-t$ path after removing any k vulnerable edges from the graph. Formally, the problem FTP is defined as follows.



© David Adjiashvili, Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt; licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 5; pp. 5:1–5:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Fault-Tolerant Path (FTP)

Instance: edge-weighted directed graph $D = (V, A)$, two vertices $s, t \in V$, set $M \subseteq A$ of *vulnerable* edges, and integer $k \in \mathbb{Z}_{\geq 0}$.

Task: Find minimum-cost set $S \subseteq A$ such that $S \setminus F$ contains an s - t path for every $F \subseteq M$ with $|F| \leq k$.

Observe that if $M = A$ then FTP is exactly s - t EDP. We also study a generalization of s - t EDP with a simpler but still non-uniform fault model: The problem Fault-Tolerant Flow (FTF) asks for $\ell \geq 1$ fault-tolerant disjoint s - t paths, assuming that only a single edge can be a-posteriori removed from the graph:

Fault-Tolerant Flow (FTF)

Instance: edge-weighted directed graph $D = (V, A)$, two vertices $s, t \in V$, set $M \subseteq A$ of *vulnerable* edges, and integer $\ell \in \mathbb{Z}_{\geq 0}$.

Task: Find minimum cost set $S \subseteq A$ such that $S \setminus f$ contains ℓ disjoint s - t paths for every $f \in M$.

1.1 Results

Consider the following well-known polynomial-time algorithm for s - t EDP: Assign unit capacities to all edges in G and find a minimum-cost s - t flow of value k . The integrality property of the LP formulation of the *Minimum-Cost s - t Flow* (MCF) problem guarantees that there is always an integer extreme-point. Such a point corresponds to a set of edges of an optimal solution and can be found in polynomial time (see for example [18]). It is natural to ask whether this approach works also for FTP, which generalizes s - t EDP. We give a negative answer by showing that FTP is NP-hard and hence the existence of a polynomial time algorithm for FTP is unlikely. In fact, the existence of constant-factor approximation algorithms is unlikely even when input graphs are directed acyclic graphs. On the positive side we provide polynomial-time algorithms for arbitrary graphs and $k = 1$ as well as directed acyclic graphs and fixed k .

We furthermore investigate the approximability of FTP using its fractional relaxation FRAC-FTP, which is defined as follows.

Fractional FTP (FRAC-FTP)

Instance: edge-weighted directed graph $D = (V, A)$, two vertices $s, t \in V$, set $M \subseteq A$ of the edges, and integer $k \in \mathbb{Z}_{\geq 0}$.

Task: Find minimum cost capacity vector $x : A \rightarrow [0, 1]$ such that for every $F \subseteq M$ with $|F| \leq k$, the maximum s - t flow in $G_F = (V, A \setminus F)$ capacitated by x is at least one.

Observe that by adding the requirement that $x \in \{0, 1\}^A$ to FRAC-FTP, we obtain FTP. Recall that for MCF the value of an optimal integer solution is equal to the value of an optimal fractional solution. We show that in contrast to MCF the integrality gap of FRAC-FTP is bounded by $k + 1$ and that this bound is essentially tight in the sense that there is an infinite family of instances with integrality gap arbitrarily close to $k + 1$. This result also leads to a simple LP-based $(k + 1)$ -approximation algorithm for FTP, which we

then combine with an algorithm for the case $k = 1$ to obtain a k -approximation algorithm for FTP. Note that FTP also admits the following simple $(k + 1)$ -approximation algorithm: Replace each safe edge with $k + 1$ parallel edges and find $k + 1$ edge-disjoint paths from s to t with minimum cost. It is not clear however how to obtain a k -approximation based on this algorithm, so our LP-based analysis is justified.

The second problem we study is FTF, which asks for $\ell \geq 1$ disjoint s - t paths in the presence of non-uniform single-edge faults. Observe that in the special case of uniform faults (every edge is vulnerable) an optimal solution is a minimum-cost s - t flow of value $k + \ell$ which can be computed in polynomial time. We show that as before the presence of non-uniform faults makes the problem much harder. In fact, it is as hard to approximate as FTP, despite the restriction to single-edge faults (the same result holds for FTF on undirected graphs). On the positive side, we give a simple polynomial-time $(\ell + 1)$ -approximation algorithm for FTF which computes a MCF with appropriately chosen capacities.

Note that our positive results for FTP imply a polynomial-time algorithm for FTF when $\ell = 1$. Together with the hardness of FTF in general this motivates the questions how the complexity of FTF depends on the number ℓ of disjoint paths. To this end, we fix ℓ and study the corresponding slice *Fault-Tolerant ℓ -Flow* of FTF. Our main result is a 2-approximation algorithm for this problem. In a nutshell, the algorithm first computes a minimum-cost ℓ -flow and then makes the resulting ℓ disjoint paths fault tolerant by solving the corresponding *augmentation problem*. We solve the augmentation problem by reducing it to a shortest path problem; it is basically a dynamic programming algorithm in disguise. However, the reduction is quite involved: in order to construct the instance of Shortest s - t -Path, we solve at most $n^{2\ell}$ instances of the Min-cost Strongly Connected Subgraphs problem on ℓ terminal pairs, all of which can be done in polynomial time for fixed ℓ . Hence, the overall running time is polynomial for fixed ℓ .

In the light of our approximation results for Fault-Tolerant ℓ -Flow, one may wonder whether the problem may even admit a polynomial-time algorithm (assuming $P \neq NP$, say). An indication in this direction is that for a number of problems with a similar flavor, including robust paths [3], robust matchings [16] or robust spanning trees [2], hardness results were obtained by showing that the corresponding augmentation problems are hard. However, our results mentioned above show that this approach does not work for FTF. We show that a polynomial-time algorithm for Fault-Tolerant ℓ -Flow implies polynomial-time algorithms for 1-2-connected Directed 2 Steiner Tree. Whether this problem is NP-hard or not is a long-standing open question.

1.2 Related Work

The shortest path problem is a classical problem in the area of combinatorial optimization and as such, it has received considerable attention also in the context of fault tolerance, see for example [4, 10, 13, 6, 19, 20]. Most of the variants of the Shortest Path Problem studied in these references, as well as FTP and FTF, are subsumed by the Capacitated Survivable Network Design Problem, which due to its generality is hard to approximate even within a factor of $2^{\log^{1-\delta}(n)}$ on directed graphs for any $\delta > 0$ under standard complexity assumptions [7]. The problems FTP and FTF also fit in the framework of *bulk-robustness* introduced by Adjashvili, Stiller and Zenklusen [3]. In this model, we are given a set of *failure scenarios*, that is, a set of subsets of resources that may fail simultaneously. The task is to find a minimum-cost subset of the resources such that a desired property (e.g., connectivity of a graph) is maintained, no matter which failure scenario materializes. Adjashvili, Stiller and Zenklusen considered bulk-robust counterparts of the Shortest Path and Minimum

Matroid basis problems. For bulk-robust shortest paths on undirected graphs they give a $O(k + \log n)$ -approximation algorithm, where k is the maximum size of a failure scenario. However, the running-time of this algorithm is exponential in k . Note that their bulk-robust shortest path problem generalizes FTP, and therefore the same approximation guarantee holds for FTP. Our approximation algorithm for FTP significantly improves on this bound, on both the approximation guarantee and the running-time. Furthermore, Adjiashvili [1] obtained an LP-based $O(k^2)$ -approximation algorithm for bulk-robust shortest paths on planar graphs.

Uniform failure models have been considered for other classical connectivity problems, such as the Minimum Spanning Tree problem: Here, if any k edges of the input graph may fail we obtain the Minimum k -Edge Connected Spanning Subgraph (k -ECSS) problem. For k -ECSS, Gabow, Goemans, Tardos and Williamson [12] gave a polynomial time $(1 + \frac{c}{k})$ -approximation algorithm for k -ECSS, for some fixed constant c . The authors also show that for some constant $c' < c$, the existence of a polynomial time $(1 + \frac{c'}{k})$ -approximation algorithm implies $P = NP$. The more general Generalized Steiner Network problem admits a polynomial 2-approximation algorithm due to Jain [17]. This is also the best known bound for weighted 2-ECSS. Non-uniform single-edge failures for the minimum spanning tree problem have been considered in [2] and a 2-approximation algorithm for this problem has been given recently by Boyd et al. [5]. A problem of a similar flavor but with uniform single-edge faults is Robust Matching Augmentation, which asks for a minimum-cost subgraph such that after the removal of any single edge, the resulting graph contains a perfect matching [16]. This problem is as hard to approximate as Directed Steiner Forest, which is known to admit no $\log^{2-\epsilon}$ -approximation algorithm unless $NP \subseteq ZTIME(n^{\text{polylog}(n)})$ [15]. The approximation hardness of FTF is a consequence of this result.

1.3 Notation

We mostly consider directed graphs, which we denote by (V, A) , where V is the set of vertices set and A the set of arcs. Undirected graphs are denoted by (V, E) where E is the edge set. When we consider edge-weighted graphs we assume throughout that the weights are non-negative. Let $G = (V, A)$ be a digraph with vulnerable arcs $M \subseteq A$. We denote by $\overline{M} := A \setminus M$ the set of safe arcs. Furthermore, for any set $\emptyset \neq X \subsetneq V$ of vertices of G , we denote by $\delta(X)$ the set of arcs $vw \in A$ such that $v \in X$ and $w \notin X$.

1.4 Organization

The remainder of this paper is organized as follows. In Section 2, we present our results on FTP. We first show that FTP on undirected graphs is a special case of FTP on directed graphs. We provide exact polynomial algorithms for two special cases of FTP in Section 2.1. In Section 2.2 we relate FTP and FRAC-FTP by proving a tight bound on the integrality gap and show how this result leads to a k -approximation algorithm for FTP. In Section 2.3 we study the approximation hardness of FTP. Section 3 contains the results on the problem FTF. Approximation hardness of FTF is shown in Section 3.1. Section 3.2 contains the approximation algorithms for FTF with and without fixed flow value ℓ . Furthermore, in Section 3.3, we relate the complexity of FTF with fixed ℓ to other problems of open complexity. Section 4 concludes the paper and contains open problems.

2 Fault-Tolerant Paths

Assuming non-negative edge-weights, the shortest path problem on undirected graphs is a special case of the same problem on directed graphs: we may replace each undirected edge by two anti-parallel directed edges and conclude that any shortest path in the resulting digraph corresponds to a shortest path in the original undirected graph. We show that this observation extends to FTP. For this purpose we show that any solution to an FTP instance on an undirected graph admits an orientation, such that in each failure scenario a directed s - t path remains (assuming that if an undirected edge fails, both corresponding anti-parallel arcs fail). As a consequence, the positive results for FTP on directed graphs given in sections 2.1 and 2.2 also hold for FTP on undirected graphs.

► **Proposition 1.** *Let $X \subseteq E$ be a feasible solution to an instance of FTP on an undirected graph (V, E) . Then there is an orientation \vec{X} of X such that $(V, \vec{X} - F)$ contains a directed s - t path for every $F \subseteq M$ with $|F| \leq k$.*

Proof. Let us assume for a contradiction that there is no such orientation. A set Y of (undirected and directed) edges is a *partial orientation* of X if there is a partition of X into sets X_1 and X_2 such that $Y = X_1 \cup \vec{X}_2$, where \vec{X}_2 is an orientation of X_2 . Let Y be a partial orientation of X that maximizes the number of directed edges such that $(V, \vec{X} - F)$ contains a directed s - t path for every $F \subseteq M$ with $|F| \leq k$. By our assumption, there is at least one undirected edge $e = vw$ in Y . Furthermore, there are two sets $S_1, S_2 \subseteq V$ of vertices, such that $\{s\} \subseteq S_1, S_2 \subseteq V \setminus \{t\}$, $v \in S_1 \setminus S_2$, and $w \in S_2 \setminus S_1$. Note that $vw \in \delta(S_1)$ and $wv \in \delta(S_2)$.

Since e is needed in both directions for Y to be feasible, there is some $F \subseteq M$, $|F| \leq k$ such that $X \setminus F$ contains an s - t path that must leave S_1 via vw . Therefore, the cut $\delta(S_1)$ contains at most $k + 1$ edges and all of them except possibly e are vulnerable. The same holds for $\delta(S_2)$ and therefore we have $|\delta(S_1)| = |\delta(S_2)| = k + 1$. From the feasibility of Y and the fact that all edges in $\delta(S_1)$ and $\delta(S_2)$ except possibly e are vulnerable, it follows that $|\delta(S_1 \cap S_2)| \geq k + 1$ and $|\delta(S_1 \cup S_2)| \geq k + 1$. By the submodularity of the cut function $|\delta(\cdot)|$ we have

$$2k + 2 = |\delta(S_1)| + |\delta(S_2)| \geq |\delta(S_1 \cap S_2)| + |\delta(S_1 \cup S_2)| \geq 2k + 2 \quad (1)$$

so we have equality throughout. Furthermore, $|\delta(\cdot)|$ satisfies the following identity

$$|\delta(S_1)| + |\delta(S_2)| = |\delta(S_1 \cap S_2)| + |\delta(S_1 \cup S_2)| + |A(S_1 \setminus S_2, S_2 \setminus S_1)| + |A(S_2 \setminus S_1, S_1 \setminus S_2)| ,$$

but the observation that e is an edge connecting $S_1 \setminus S_2$ and $S_2 \setminus S_1$, together with the fact the we have equality in (1) yields a contradiction to the previous identity. ◀

2.1 Exact Algorithms

In this section we give polynomial-time algorithms for FTP on arbitrary graphs, where at most one edge can fail ($k = 1$) and FTP on directed acyclic graphs (DAGs) for fixed k . We start with the following useful observation.

► **Lemma 2.** *Let $I = (D = (V, A), s, t, M, k)$ be an FTP-instance and $X \subseteq A$. Then the following statements are equivalent.*

1. X is a feasible solution to I .
2. The network (V, X) with capacities $c_e = 1$ if $e \in M$ and $c_e = \infty$ otherwise admits an s - t flow of value at least $k + 1$.

Proof. Let $X \subseteq E(D)$. First, suppose that X is a feasible solution to the FTP instance (D, M, k) . Suppose for a contradiction that the network $((V, X), c)$ capacitated by $c_e = 1$ if $e \in M$ and $c_e = \infty$ otherwise admits no s - t flow of value at least $k + 1$. Then, by the Max-Flow Min-Cut Theorem, there is some capacitated cut $\delta(V')$ for some $V' \subseteq V$ with $s \in V'$ and $t \notin V'$ such that $c(\delta(V')) < k + 1$. By the definition of c , this implies that $\delta(V')$ does not contain any safe edge. But then $F := \delta(V')$ is a cut in (V, X) of size at most k , a contradiction to the feasibility of X .

Now, suppose that X is not a feasible solution to the FTP instance (D, M, k) . Then there is some capacitated cut $\delta(V')$ for some $V' \subseteq V$ with $s \in V'$ and $t \notin V'$ such that $c(\delta(V')) \leq k$. But then, by the Max-Flow Min-Cut Theorem, the network $((V, X), c)$ admits no s - t flow of value at least $k + 1$. ◀

We consider the restriction of FTP to $k = 1$. An s - t *bipath* in the graph $D = (V, A)$ is a union of two (not necessarily disjoint) s - t paths $P_1, P_2 \subseteq A$. In the context of 1-FTP we call a bipath $Q = P_1 \cup P_2$ *robust* if $P_1 \cap P_2 \cap M = \emptyset$. Note that every robust s - t bipath Q in G is a feasible solution to the 1-FTP instance. Indeed, consider any vulnerable edge $e \in M$. Since $e \notin P_1 \cap P_2$ it holds that either $P_1 \subseteq Q - e$, or $P_2 \subseteq Q - e$. It follows that $Q - e$ contains some s - t path. The next lemma shows that every feasible solution of the 1-FTP instance contains a robust s - t bipath.

► **Lemma 3.** *Every feasible solution S^* to an 1-FTP instance contains a robust s - t bipath.*

Proof. We assume without loss of generality that S^* is a minimal feasible solution with respect to inclusion. Let $Y \subseteq S^*$ be the set of bridges in (V, S^*) . From feasibility of S^* , we have $Y \cap M = \emptyset$. Consider any s - t path P in S^* . Let u_1, \dots, u_r be the set of vertices incident to $Y = P \cap Y$. Let u_i and u_{i+1} be such that $u_i u_{i+1} \notin Y$. (if such an edge does not exist, we have $Y = P$, which means that P is a robust s - t bipath). Note that S^* must contain two edge-disjoint u_i - u_{i+1} paths L_1, L_2 . Taking as the set Y together with all such pairs of paths L_1, L_2 results in a robust bipath. ◀

We conclude from the previous discussion and Lemma 3 that all minimal feasible solutions to the 1-FTP instance are robust bipaths. This observation leads to the simple polynomial-time algorithm for 1-FTP that, using flow-techniques, computes for any pair of vertices u, v the length of i) a min-cost u - v path using only safe edges and ii) two edge-disjoint u - v paths of minimum cost. In a second step the algorithm computes a minimum-cost s - t path in a complete graph with respect to the minimum of the two computed costs. The resulting s - t path corresponds to a min-cost s - t bipath in the original graph and hence, by Lemma 3, an optimal robust s - t path.

► **Theorem 4.** *1-FTP admits a polynomial-time algorithm.*

Proof. To solve 1-FTP we need to find the minimum cost robust s - t bipath. To this end let us define two length functions $\ell_1, \ell_2 : V \times V \rightarrow \mathbb{R}_{\geq 0}$. For two vertices $u, v \in V$ let $\ell_1(u, v)$ denote the shortest path distance from u to v in the graph $(V, A \setminus M)$, and let $\ell_2(u, v)$ denote the cost of the shortest pair of edge-disjoint u - v paths in D . Clearly, both length functions can be computed in polynomial time (e.g. using flow techniques). Finally, set $\ell(u, v) = \min\{\ell_1(u, v), \ell_2(u, v)\}$. Construct the complete graph on the vertex set V and associate the length function ℓ with it. Observe that by definition of ℓ , any s - t path in this graph corresponds to a robust s - t bipath with the same cost, and vice versa. It remains to find the shortest s - t bipath by performing a single shortest s - t path in the new graph. For every edge uv in this shortest path, the optimal bipath contains the shortest u - v path in $(V, A \setminus M)$ if $\ell(u, v) = \ell_1(u, v)$, and the shortest pair of u - v paths in D , otherwise. ◀

We now consider the problem k -FTP (for fixed $k \in \mathbb{N}$) on layered graphs. The generalization to a directed acyclic graph is done via a standard transformation, which we describe later. Recall that a layered graph $D = (V, A)$ is a graph with a partitioned vertex set $V = V_1 \cup \dots \cup V_r$ and a set of edges satisfying $A \subset \bigcup_{i \in [r-1]} V_i \times V_{i+1}$. We assume without loss of generality that $V_1 = \{s\}$ and $V_r = \{t\}$. For every $i \in [r-1]$ we let $A_i = A \cap V_i \times V_{i+1}$. We reduce k -FTP to a shortest path problem in a larger graph. The following definition sets the stage for the algorithm.

► **Definition 5.** An i -configuration is a vector $d \in \{0, 1, \dots, k+1\}^{V_i}$ satisfying $\sum_{v \in V_i} d_v = k+1$. We let $\text{supp}(d) = \{v \in V_i : d_v > 0\}$. For an i -configuration d^1 and an $(i+1)$ -configuration d^2 we let

$$V(d^1, d^2) = \text{supp}(d^1) \cup \text{supp}(d^2) \quad \text{and} \quad A(d^1, d^2) = A[V(d^1, d^2)].$$

We say that an i -configuration d^1 precedes an $(i+1)$ -configuration d^2 if the following flow problem is feasible. The graph is defined as $H(d^1, d^2) = (V(d^1, d^2), A(d^1, d^2))$. The demand vector ν and the capacity vector c are given by

$$\nu_u = \begin{cases} -d_u^1 & \text{if } u \in \text{supp}(d^1) \\ d_u^2 & \text{if } u \in \text{supp}(d^2) \end{cases} \quad \text{and} \quad c_e = \begin{cases} 1 & \text{if } e \in M \\ \infty & \text{if } e \in E \setminus M, \end{cases}$$

respectively. If d^1 precedes d^2 we say that the link (d^1, d^2) exists. Finally, the cost $\ell(d^1, d^2)$ of this link is set to be minimum value $w(A')$ over all $A' \subseteq A(d^1, d^2)$, for which the previous flow problem is feasible, when restricted to the set of edges A' .

The algorithm constructs a layered graph $\mathcal{H} = (\mathcal{V}, \mathcal{A})$ with r layers $\mathcal{V}_1, \dots, \mathcal{V}_r$. For every $i \in [r]$ the set of vertices \mathcal{V}_i contains all i -configurations. Observe that since $V_1 = \{s\}$ and $V_r = \{t\}$, we have that \mathcal{V}_1 and \mathcal{V}_r contain one vertex each, which we denote by c^s and c^t , respectively. The edges correspond to links between configurations. Every edge is directed from the configuration with the lower index to the one with the higher index. The cost is set according to Definition 5. The following lemma provides the required observation, which immediately leads to a polynomial-time algorithm.

► **Lemma 6.** Every c^s - c^t path P in \mathcal{H} corresponds to a fault-tolerant path S with $w(S) \leq \ell(P)$, and vice-versa.

Proof. Consider first a fault-tolerant path $S \subseteq A$. We construct a corresponding c^s - c^t path in \mathcal{H} as follows. Consider any $k+1$ s - t flow f^S , induced by S . Let p^1, \dots, p^l be a path decomposition of f^S and let $1 \leq \rho_1, \dots, \rho_l \leq k+1$ (with $\sum_{i \in [l]} \rho_i = k+1$) be the corresponding flow values.

Since D is layered, the path p^j contains exactly one vertex v_j^i from V_i and one edge e_j^i from A_i for every $j \in [l]$ and $i \in [r]$. For every $i \in [r]$ define the i -configuration d^i with

$$d_v^i = \sum_{j \in [l]: v=v_j^i} \rho_j,$$

if some path p^j contains v , and $d_v^i = 0$, otherwise. The fact that d^i is an i -configuration follows immediately from the fact that f^S is a $(k+1)$ -flow. In addition, for the same reason d^i precedes d_{i+1} for every $i \in [r-1]$. From the latter observations and the fact that $d^1 = c^s$ and $d^r = c^t$ it follows that $P = d^1, d^2, \dots, d^r$ is a c^s - c^t path in \mathcal{H} with cost $\ell(P) \leq w(S)$.

Consider next an c^s - c^t path $P = d^1, \dots, d^r$ with cost $\ell(P) = \sum_{i=1}^{r-1} \ell(d^i, d^{i+1})$. The cost $\ell(d^i, d^{i+1})$ is realized by some set of edges $R_i \subseteq A(d^i, d^{i+1})$ for every $i \in [r-1]$. From Definition 5, the maximal s - t flow in the graph $D' = (V, R)$ is at least $k+1$, where

$R = \cup_{i \in [r-1]} R_i$. Next, Lemma 2 guarantees that there exists some feasible solution $S \subseteq R$, the cost of which is at most $\ell(P)$. In the latter claim we used the disjointness of the sets R_i , which is due the layered structure of the graph G . This concludes the proof of the lemma. \blacktriangleleft

Finally, we observe that the number of configurations is bounded by $O(n^{k+1})$, which implies that k -FTP can be solved in polynomial time on layered graphs.

To obtain the same result for directed acyclic graphs we perform the following transformation of the graph. Let v_1, \dots, v_n be a topological sorting of the vertices in D . Replace every edge $e = v_i v_j$ ($i < j$) with a path $p_e = v_i, u_{i+1}^e, \dots, u_{j-1}^e, v_j$ of length $j - i + 1$ by subdividing it sufficiently many times. Set the cost of the first edge on the path to $w'(v_i u_{i+1}^e) = w(v_i v_j)$ and set the costs of all other edges on the path to zero. In addition, create a new set of faulty edges M' , which contains all edges in a path p_e if $e \in M$. It is straightforward to see that the new instance of FTP is equivalent to the original one, while the obtained graph after the transformation is layered. We summarize the result as follows.

► **Theorem 7.** *There is a polynomial-time algorithm for k -FTP restricted to instances with a directed acyclic graph.*

2.2 Integrality Gap and Approximation Algorithms

In this section we study the natural fractional relaxation FRAC-FTP of FTP and prove a tight bound on its integrality gap. That is, we bound the worst-case ratio of the value of an optimal solution of an FTP instance and the corresponding optimal value of FRAC-FTP. This result also suggests a simple approximation algorithm for FTP with ratio $k + 1$. We then combine this algorithm with the algorithm for 1-FTP to obtain a k -approximation algorithm.

Fractional FTP and Integrality Gap

We give the following bound on the integrality gap of FRAC-FTP.

► **Theorem 8.** *The integrality gap of FRAC-FTP is at most $k + 1$. Furthermore, there exists an infinite family of instances of FTP with integrality gap arbitrarily close to $k + 1$.*

Proof. Consider an instance $I = (D, s, t, M, k)$ of FTP. Let x^* denote an optimal solution to the corresponding FRAC-FTP instance, and let $OPT = w(x^*)$ be its cost. Define a vector $y \in \mathbb{R}^A$ as follows.

$$y_e = \begin{cases} (k+1)x_e & \text{if } e \notin M \\ \min\{1, (k+1)x_e\} & \text{otherwise.} \end{cases} \quad (2)$$

Clearly, it holds that $w(y) \leq (k+1)OPT$. We claim that every s - t cut in D with capacities y has capacity of at least $k + 1$. Consider any such cut $C \subset A$, represented as the set of edges in the cut. Let $M' = \{e \in M : x_e^* \geq \frac{1}{k+1}\}$ denote the set of faulty edges attaining high fractional values in x^* . Define $C' = C \cap M'$. If $|C'| \geq k + 1$ we are clearly done. Otherwise, assume $|C'| \leq k$. In this case consider the failure scenario $F = C'$. Since x^* is a feasible solution it must hold that $\sum_{e \in C \setminus C'} x_e^* \geq 1$. Since for every edge $e \in C \setminus C'$ it holds that $y_e = (k+1)x_e^*$ we obtain

$$\sum_{e \in C \setminus C'} y_e \geq k + 1$$

as desired. From our observations it follows that the maximum flow in D with capacities y is at least $k + 1$. Finally, consider the minimum cost $(k + 1)$ -flow z^* in D with capacities defined by

$$c_e = \begin{cases} k + 1 & \text{if } e \notin M \\ 1 & \text{otherwise.} \end{cases}$$

From integrality of c and the minimum-cost flow problem we can assume that z^* is integral. Note that $y_e \leq c_e$ for every $e \in A$, hence any feasible $(k + 1)$ -flow with capacities y is also a feasible $(k + 1)$ -flow with capacities c . From the previous observation it holds that $w(z^*) \leq w(y) \leq (k + 1)OPT$. From Lemma 2 we know that the support of z^* is a feasible solution to the FTP instance. This concludes the proof of the upper bound of $k + 1$ for the integrality gap.

To prove the same lower bound we provide an infinite family of instances, containing instances with integrality gap arbitrarily close to $k + 1$. Consider a graph with $p \gg k$ parallel edges with unit cost connecting s and t , and let $M = A$. An optimal solution to this FTP instance chooses any subset of $k + 1$ edges. At the same time, the optimal solution to FRAC-FTP assigns a capacity of $\frac{1}{p-k}$ to every edge. This solution is feasible, since in every failure scenario, the number of edges that survive is at least $p - k$, hence the maximum s - t flow is at least one. The cost of this solution is $\frac{p}{p-k}$. Taking p to infinity yields instances with integrality gap arbitrarily close to $k + 1$. ◀

The proof of Theorem 8 leads to a simple $(k + 1)$ -approximation algorithm for FTP. However, simply creating k copies of each vulnerable arc and finding a minimum-cost s - t flow of value at least $k + 1$ gives a $(k + 1)$ -approximation as well.

► **Proposition 9.** *FTP admits a polynomial-time $(k + 1)$ -approximation algorithm.*

A k -Approximation Algorithm

We propose an LP-based k -approximation algorithm for FTP that is a refinement of the LP-based approximation algorithm of Proposition 9. Intuitively, the reason why the algorithm of Proposition 9 gives an approximation ratio of $k + 1$ is that the capacity of the edges in $A \setminus M$ is set to $k + 1$. Therefore, if an s - t flow z^* uses such an edge to its full capacity, the cost incurred is $k + 1$ times the cost of the edge. Hence, the best possible lower bound on the cost $w(z^*)$ is $(k + 1)OPT_{FRAC}$, where OPT_{FRAC} denotes the optimal value of the corresponding FRAC-FTP instance. To improve the algorithm we observe that each edge which carries a flow of $k + 1$ according to z^* is a cut-edge in the obtained solution.

Let $I = (D, s, t, M)$ be an instance of FTP. We begin our analysis by considering a certain canonical flow defined by minimal feasible solutions.

► **Definition 10.** *Consider an inclusion-wise minimal feasible solution $S \subseteq A$ to I . A flow f^S induced by S is any integral s - t $(k + 1)$ -flow in D respecting the capacity vector*

$$c_e^S = \begin{cases} 1 & \text{if } e \in S \cap M \\ k + 1 & \text{if } e \in S \setminus M \\ 0 & \text{if } e \in A \setminus S. \end{cases}$$

Consider an optimal solution $X^* \subseteq A$ to I and a corresponding induced flow f^* . Define

$$X_{PAR} = \{e \in X^* : f^*(e) \leq k\} \text{ and } X_{BRIDGE} = \{e \in X^* : f^*(e) = k + 1\} .$$

As we argued before, every edge in X_{BRIDGE} must be a bridge in $H = (V, X^*)$ disconnecting s and t . Let e_u denote the tail vertex of an edge $e \in A$. Since every edge $e \in X_{BRIDGE}$ constitutes an s - t cut in H , it follows that the vertices in $U = \{e_u : e \in X_{BRIDGE}\} \cup \{s, t\}$ can be unambiguously ordered according to the order in which they appear on any s - t path in H , traversed from s to t . Let $s = u_1, \dots, u_q = t$ be this order. Except for s and t , every vertex in U constitutes a cut-vertex in H . Divide H into $q - 1$ subgraphs H^1, \dots, H^{q-1} by letting $H^i = (V, Y_i)$ contain the union of all u_i - u_{i+1} paths in H . We observe the following:

► **Proposition 11.** *For every $i \in \{1, 2, \dots, q - 1\}$ the set $Y_i \subseteq A$ is an optimal solution to the FTP instance $I_i = (G, u_i, u_{i+1}, M)$.*

Consider some $i \in \{1, 2, \dots, q - 1\}$ and let f_i^* denote the flow f^* , restricted to edges in H^i . Note that f_i^* can be viewed as a u_i - u_{i+1} $(k + 1)$ -flow. Exactly one of the following cases can occur. Either H^i contains a single edge $e \in A \setminus M$, or $\max_{e \in Y_i} f_i^*(e) \leq k$. In the former case, the edge e is the shortest u_i - u_{i+1} path in $(V, A \setminus M)$. In the latter case we use an algorithm that is similar to the one of Proposition 9 to obtain a k -approximation of the optimal FTP solution on instance I_i . Concretely, the algorithm defines the capacity vector $c'(e) = k$ if $e \notin M$ and $c'(e) = 1$, otherwise, and finds an integral minimum-cost u_i - u_{i+1} $(k + 1)$ -flow Y^* in D , and returns the support $Y \subseteq A$ of the flow as the solution. The existence of the flow f_i^* guarantees that $w(y^*) \leq w(f_i^*)$, while the fact that the maximum capacity in the flow problem is bounded by k gives $w(Y) \leq kw(y^*)$. It follows that this algorithm approximates the optimal solution to the FTP instance I_i to within a factor k .

The final algorithm uses the algorithm for 1-FTP as a blueprint. However, instead of finding two edge-disjoint u - v paths, the new algorithm solves the aforementioned flow problem. We summarize the main result of this section as follows.

► **Theorem 12.** *FTP admits a polynomial-time k -approximation algorithm.*

2.3 Approximation Hardness

We complement our algorithmic results by showing approximation hardness for FTP. An instance of the problem *Directed m -Steiner Tree* (m -DST) is given by a weighted directed graph $D = (V, A)$, a source node $s \in V$, a set $T \subseteq V$ of terminals and an integer $m \leq |T|$. The goal is to find a minimum-cost arborescence $X \subseteq A$ rooted at s that connects s to m terminals. Halperin and Krauthgamer [15] showed that m -DST cannot be approximated within a factor $\log^{2-\epsilon} m$ for every $\epsilon > 0$, unless $\text{NP} \subseteq \text{ZTIME}(n^{\text{polylog}(n)})$. We show that the problem m -DST is a special case of FTP.

Given an m -DST instance we construct an instance of FTP as follows. The graph D is augmented by $|T|$ new arcs A' of cost 0 connecting every terminal to a new node t . Finally, we let $M = A'$ and $k = m - 1$. It is readily verified that any fault-tolerant s - t path in the graph so obtained corresponds to a feasible solution to the m -DST instance of the same cost (we may assume that all arcs in A' are in some solution to the FTP instance). This implies the following conditional approximation lower bound for FTP.

► **Proposition 13.** *FTP admits no polynomial-time approximation algorithms with ratio $\log^{2-\epsilon} k$ for every $\epsilon > 0$, unless $\text{NP} \subseteq \text{ZTIME}(n^{\text{polylog}(n)})$.*

The reduction can be easily adapted to yield a k^ϵ -approximation algorithm for FTP for the special case that $M \subseteq \{e \in A : t \in e\}$ using the algorithm of Charikar et. al. [8].

We end this discussion by showing that FTP contains as a special case a more general Steiner problem, which we call *Simultaneous Directed m -Steiner Tree* (m -SDST). An input to m -SDST specifies two arc-weighted digraphs $D_1 = (V, A_1, w_1)$ and $D_2 = (V, A_2, w_2)$ on

the same set V of vertices, a source s , a set $T \subseteq V$ of terminals, and an integer $m \leq |T|$. The goal is to find a subset $U \subseteq T$ of m terminals and two arborescences $S_1 \subseteq A_1$ and $S_2 \subseteq A_2$ connecting s to U in the respective graphs, so as to minimize $w_1(S_1) + w_2(S_2)$. m -SDST is seen to be a special case of FTP via the following reduction. Given an instance of m -SDST, construct a graph $D = (V', A)$ as follows. Take a disjoint union of D_1 and D_2 , where the direction of every arc in D_2 is reversed. Connect every copy of a terminal $u \in T$ in D_1 to its corresponding copy in D_2 with an additional zero-cost arc e_u . Finally, set $M = \{e_u : u \in T\}$ and $k = m - 1$. A fault-tolerant path connecting the copy of s in D_1 to the copy of s in D_2 corresponds to a feasible solution to the m -SDST instance with the same cost, and vice-versa.

3 Fault-Tolerant Flows

In this section we present our results on the problem FTF. We first give an approximation hardness result and then investigate the complexity of FTF for fixed flow values ℓ . Our main result is a polynomial-time algorithm for the corresponding augmentation problem, which we use to obtain a 2-approximation for Fault-Tolerant ℓ -Flow. We conclude by showing that a polynomial-time algorithm for Fault-Tolerant ℓ -Flow implies polynomial-time algorithms for two problems whose complexity status is open.

3.1 Approximation Hardness of FTF

We show that FTF is as hard to approximate as Directed Steiner Forest by using an approximation hardness result from [16] for the problem Weighted Robust Matching Augmentation. The problem Weighted Robust Matching Augmentation asks for the cheapest edge-set (assuming non-negative costs) to add to a bipartite graph such that the resulting graph is bipartite and contains a perfect matching after a-posteriori removing any single edge. The idea of our reduction is similar to that of the classical reduction from the Bipartite Maximum Matching problem to the Max s - t Flow problem. Note that we may assume that both parts of the input graph have the same size. We add to the graph (U, W, E) on n vertices of a Weighted Robust Matching Augmentation instance I two terminal vertices s and t , and connect s to each vertex of U and each vertex of W to t by an arc of cost 0. Now we add all possible arcs from U to W , marking those as vulnerable that correspond to an edge in E ; the costs are according to I . Observe that a fault-tolerant $n/2$ -flow corresponds to a feasible solution to the Weighted Robust Matching Augmentation instance after deleting s and t .

► **Lemma 14.** *A polynomial-time $f(\ell)$ approximation algorithm for FTF implies a polynomial-time $f(n/2)$ -approximation algorithm for Weighted Robust Matching Augmentation, where n is the number of vertices in the Weighted Robust Matching Augmentation instance.*

Proof. In the following it will be convenient to denote by \overline{E} the edge-set of the bipartite complement of a bipartite graph with edge-set E . Let $I = (G, c)$ be an instance of Weighted Robust Matching Augmentation where $G = (U, W, E)$ is a balanced bipartite graph on n vertices and $c \in \mathbb{Z}_{\geq 0}^{\overline{E}}$. Our reduction is similar to the classical reduction from the perfect matching problem in bipartite graphs to the Max s - t Flow problem. We construct in polynomial-time an instance $I' = (D', c', s, t, M)$ of FTF as follows. To obtain the digraph $D' = (V, A)$, we add to the vertex set of G two new vertices s and t and add all arcs from s to U and from W to t . Furthermore, we add all arcs from U to W and consider those that correspond to an edge in E as vulnerable. That is, we let $M := \{uw : u \in U, w \in W, uw \in E\}$. To complete the construction of I' , we let $\ell = n/2$, and let the arc-costs c' be given by

$$c'_{uw} := \begin{cases} c_{uw} & \text{if } uw \in E(G), \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

For $X \subseteq E \cup \overline{E}$ we write $q(X)$ for the corresponding set of arcs of D' . Similarly, for a set $Y \subseteq A$ of arcs we write $q^{-1}(Y)$ for the corresponding set of undirected edges of G . Observe that for a feasible solution X to I , the arc set $q(X) \cup A_s \cup A_t$ is feasible for I' , where A_s (resp., A_t) is the set of arcs leaving s (resp., entering t). Furthermore, a feasible solution Y to I' corresponds to a feasible solution $q^{-1}(Y \setminus (A_s \cup A_t))$ to I . Also note that, by the choice of c' , we have that the cost of two corresponding solutions is the same. It follows that since $\ell = n/2$, any polynomial-time $f(\ell)$ -factor approximation algorithm for Fault-Tolerant ℓ -Flow implies a polynomial-time $f(n/2)$ -factor approximation algorithm for Weighted Robust Matching Augmentation, where $n = |U + W|$. ◀

We combine Lemma 14 with two hardness results from [16] and [15] to obtain the following approximation hardness result for FTF.

► **Theorem 15.** *FTF admits no polynomial-time $\log^{2-\varepsilon}(\ell)$ -factor approximation algorithm for every $\varepsilon > 0$, unless $\text{NP} \subseteq \text{ZTIME}(n^{\text{polylog}(n)})$.*

Proof. We give a polynomial-time cost-preserving reduction from Directed Steiner Forest to FTF via Weighted Robust Matching Augmentation. The intermediate reduction step from Directed Steiner Forest to Weighted Robust Matching Augmentation is given in [16, Prop. 6.1]. Consider an instance I of Directed Steiner Forest on a weighted digraph $D = (V, A)$ on n vertices with k terminal pairs $(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)$. According to the reduction given in the proof of [16, Prop. 18], we obtain an instance of Weighted Robust Matching Augmentation on a graph of at most $2(n+k) + 2(n-k) = 4n =: n'$ vertices. By the arguments their proof, a $f(n')$ -approximation algorithm for Weighted Robust Matching Augmentation yields a $f(4n)$ -approximation algorithm for Directed Steiner Forest. We apply Proposition 14 to conclude that an $f(\ell)$ -approximation algorithm for FTF yields a $f(2n)$ -approximation algorithm for Directed Steiner Forest. According to the result of Halperin and Krauthgamer [15], the problem Directed Steiner Forest admits no polynomial-time $\log^{2-\varepsilon} n$ -approximation algorithm for every $\varepsilon > 0$, unless $\text{NP} \subseteq \text{ZTIME}(n^{\text{polylog}(n)})$. We conclude that FTF admits no polynomial-time $\log^{2-\varepsilon}(\ell/2)$ -factor approximation algorithm under the same assumption. ◀

3.2 Approximation Algorithms

We first present a simple polynomial-time $(\ell + 1)$ -approximation algorithm for FTF, which is similar to the LP-based $(k + 1)$ -approximation for FTP. The algorithm computes a minimum-cost s - t flow of value $\ell + 1$ on the input graph with the following capacities: each vulnerable arc receives capacity 1 and any other arc capacity $1 + 1/\ell$. The solution then consists of all arcs in the support of the flow. To see that for this choice of capacities we obtain a feasible solution, recall that the value of any s - t cut upper-bounds the value of any s - t flow. Therefore, each s - t cut C has value at least $\ell + 1$, so C contains either at least ℓ safe arcs or at least $\ell + 1$ arcs. To prove the approximation guarantee, we show that any optimal solution to an FTF instance contains an s - t flow of value $\ell + 1$ and observe that we over-pay safe arcs by a factor of at most $(1 + 1/\ell)$.

► **Theorem 16.** *FTF admits a polynomial-time $(\ell + 1)$ -approximation algorithm.*

Proof. Let I be an instance of FTF on a digraph $D = (V, A)$ with weight $c \in \mathbb{Z}_{\geq 0}^A$, terminals s and t , vulnerable arcs M and desired flow value ℓ . We consider an instance $I' = (D, c, s, t, \ell + 1, g)$ of MCF, where the arc capacities g are given by

$$g_e := \begin{cases} 1 & \text{if } e \in M, \text{ and} \\ 1 + \frac{1}{\ell} & \text{otherwise} \end{cases}$$

An optimal solution to I' can be computed in polynomial-time by standard techniques. We saw in the discussion at the beginning of Section 3.2 that the set of arcs of positive flow in a solution to I' yields a feasible solution to I .

It remains to bound the approximation ratio. Let Y^* be an optimal solution to I of cost $\text{OPT}(I)$. We first show that Y^* contains $\ell + 1$ disjoint s - t paths.

▷ **Claim 1.** Y^* contains an s - t flow of value $\ell + 1$ with respect to the capacities g .

Proof. First observe that in any feasible solution to I , every s - t cut contains either at least ℓ safe arcs or at least $\ell + 1$ arcs. Now, an s - t cut Z in Y^* having at least ℓ safe arcs satisfies $g(Z) \geq (1 + \frac{1}{\ell}) \cdot \ell = \ell + 1$. On the other hand, an s - t cut Z' in Y^* containing at least $\ell + 1$ arcs satisfies $g(Z') \geq \ell + 1$. Hence, each s - t cut in Y^* has capacity at least $\ell + 1$. By the max-flow-min-cut theorem there is an s - t flow of value at least $\ell + 1$. ◁

The theorem now follows from the next claim.

▷ **Claim 2.** An optimal solution to I' has cost at most $(\ell + 1) \cdot \text{OPT}(I)$.

Proof. Let $f^* \in \mathbb{Q}^A$ be an optimal s - t flow with respect to the capacities g . Furthermore, let Y be the set of arcs of positive flow, that is $Y := \{e \in A \mid f_e^* > 0\}$. Let $Y_M = Y \cap M$ be the vulnerable arcs in Y and let $Y_S = Y \setminus Y_M$ be the safe arcs. First, we may assume that each arc $e \in Y$ has flow value at least $f_e^* \geq 1/\ell$, since each arc has capacity either 1 or $1 + \frac{1}{\ell}$. This is true since we could scale the arc capacities g by a factor ℓ , which allows us to compute (in polynomial time) an integral optimal solution with respect to the scaled capacity function, using any augmenting paths algorithm for MCF. In addition, observe that we may pay a factor of at most $1 + \frac{1}{\ell}$ too much for each safe arc since the capacity of the safe arc is $1 + \frac{1}{\ell}$. Therefore, we may bound the cost of a safe arc $e \in Y_S$ by $\ell \cdot (1 + \frac{1}{\ell}) \cdot c_e \cdot f_e$ and the cost of each vulnerable arc $e \in Y_M$ by $\ell \cdot c_e \cdot f_e$, where f_e is the flow-value of arc e according to the solution Y . Hence, we obtain

$$\begin{aligned} c(Y) &= c(Y_S) + c(Y_M) \\ &\leq \ell \cdot \left(\left(1 + \frac{1}{\ell}\right) \cdot \sum_{e \in Y_S} c_e \cdot f_e^* + \sum_{e \in Y_M} c_e \cdot f_e^* \right) \\ &\leq \ell \cdot \left(1 + \frac{1}{\ell}\right) \cdot \left(\sum_{e \in Y_S} c_e \cdot f_e^* + \sum_{e \in Y_M} c_e \cdot f_e^* \right) \\ &\leq (\ell + 1) \cdot \text{OPT}(I) , \end{aligned}$$

where the first inequality follows from the two arguments above and the last inequality follows from Claim 1. ◁

Note that we cannot simply use the dynamic programming approach as in the algorithm for 1-FTP to obtain an ℓ -approximation for FTF, since a solution to FTF in general does not have cut vertices, which are essential for the decomposition approach for the k -approximation for FTP. ◀

A 2-approximation for Fault-Tolerant ℓ -Flow

We now show that for a fixed number ℓ of disjoint paths a much better approximation guarantee can be obtained. That is, we give a polynomial-time 2-approximation algorithm for Fault-Tolerant ℓ -Flow (FT ℓ F) (however, its running time is exponential in ℓ). The algorithm first computes a minimum-cost s - t flow of value ℓ and then augments it to a feasible solution by solving the following *augmentation problem*.

Fault-Tolerant ℓ -Flow Augmentation

Instance: arc-weighted directed graph $D = (V, A)$, nodes $s, t \in V$, arc-set $X_0 \subseteq A$ that contains ℓ disjoint s - t paths, and set $M \subseteq A$ of vulnerable arcs.

Task: Find minimum weight set $S \subseteq A \setminus X_0$ such that for every $f \in M$ the set $(X_0 \cup S) \setminus f$ contains ℓ disjoint s - t paths.

Our main technical contribution is that Fault-Tolerant ℓ -Flow Augmentation can be solved in polynomial time for fixed ℓ . Our algorithm is based on a dynamic programming approach and it involves solving many instances of the problem Directed Steiner Forest, which asks for a cheapest subgraph connecting ℓ given terminal pairs. This problem admits a polynomial-time algorithm for fixed ℓ [11], but is W[1]-hard when parameterized by ℓ , so the problem is unlikely to be fixed-parameter tractable [14]. Roughly speaking, we traverse the ℓ disjoint s - t paths computed previously in parallel, proceeding one arc at a time. In order to deal with vulnerable arcs, at each step, we solve an instance of Directed Steiner Forest connecting the ℓ current vertices (one on each path) to ℓ destinations on the same path by using backup paths. That is, we decompose a solution to the augmentation problem into instances of Directed Steiner Forest connected by safe arcs. An optimal decomposition yields an optimal solution to the instance of the augmentation problem. We find an optimal decomposition by dynamic programming. Essentially, we give a reduction to a shortest path problem in a graph that has exponential size in ℓ .

Let us fix an instance I of Fault-Tolerant ℓ -Flow Augmentation on a digraph $D = (V, A)$ with arc-weights $c \in \mathbb{Z}_{\geq 0}^A$ and terminals s and t . Let P_1, P_2, \dots, P_ℓ be ℓ disjoint s - t paths contained in X_0 . We assume without loss of generality that X_0 is the union of P_1, P_2, \dots, P_ℓ . If X_0 contains an arc e that is not on any of the ℓ paths, we remove e from X_0 and assign to it weight 0.

We now give the reduction to the shortest path problem. We construct a digraph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$; to distinguish it clearly from the graph D of I , we call the elements in V (A) of D *vertices* (*arcs*) and elements of \mathcal{V} (\mathcal{A}) *nodes* (*links*). We order the vertices of each path P_i , $1 \leq i \leq \ell$, according to their distance to s on P_i . For two vertices x_i, y_i of P_i , we write $x_i \leq y_i$ if x_i is at least as close to s on P_i as y_i . Let us now construct the node set \mathcal{V} . We add a node v to \mathcal{V} for every ℓ -tuple $v = (x_1, \dots, x_\ell)$ of vertices in $V(X_0)$ satisfying $x_i \in P_i$, for every $i \in \{1, 2, \dots, \ell\}$. Note that the corresponding vertices of a node are not necessarily distinct, since the ℓ *edge*-disjoint paths P_1, P_2, \dots, P_ℓ may share vertices. We also define a (partial) ordering on the nodes in \mathcal{V} . From now on, let $v_1 = (x_1, \dots, x_\ell)$ and $v_2 = (y_1, \dots, y_\ell)$ be two nodes of \mathcal{V} . We write $v_1 \leq v_2$ if $x_i \leq y_i$ for every $1 \leq i \leq \ell$. Additionally, let $Q_i(x, y)$ be the sub-path of P_i from a vertex x of P_i to a vertex y of P_i .

We now construct the link set $\mathcal{A} := \mathcal{A}_1 \cup \mathcal{A}_2$ of \mathcal{D} as the union of two link-sets \mathcal{A}_1 and \mathcal{A}_2 , defined as follows. We add to \mathcal{A}_1 a link $v_1 v_2$, if v_1 precedes v_2 and the subpaths of each P_i from x_i to y_i contain no vulnerable arc. That is, we let

$$\mathcal{A}_1 := \{v_1 v_2 \mid v_1, v_2 \in \mathcal{V}, v_1 \leq v_2, Q_i(x_i, y_i) \cap M = \emptyset \text{ for } 1 \leq i \leq \ell\} .$$

■ **Algorithm 1** : Exact algorithm for Fault-Tolerant ℓ -Flow Augmentation.

Input: instance I of Fault-Tolerant ℓ -Flow Augmentation on a digraph $D = (V, A)$

- 1: Construct the graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
 - 2: Find a shortest path \mathcal{P} in \mathcal{D} from (s, \dots, s) to (t, \dots, t)
 - 3: For each link $vw \in \mathcal{P} \cap \mathcal{A}_2$ add the arcs of an optimal solution to $I(v, w)$ to Y
 - 4: **return** Y
-

We now define the link set \mathcal{A}_2 . Let $v_1, v_2 \in \mathcal{V}$ such that v_1 precedes v_2 . If there is some $1 \leq i \leq \ell$, such that $Q_i(x_i, y_i)$ contains at least one vulnerable arc, then we first need to solve an instance of Directed Steiner Forest on ℓ terminal pairs in order to compute the cost of the link v_1v_2 . We construct an instance $I(v_1, v_2)$ of Directed Steiner Forest as follows. The terminal pairs are $(x_i, y_i)_{1 \leq i \leq \ell}$. The input graph is given by $D' = (V, A')$, where $A' = (A \setminus X_0) \cup \bigcup_{1 \leq i \leq \ell} \overleftarrow{Q}_i(x_i, y_i)$, where $\overleftarrow{Q}_i(x_i, y_i)$ are the arcs of $Q_i(x_i, y_i)$ in reversed direction. The arc costs are given by

$$c'_e := \begin{cases} c_e & \text{if } e \in A \setminus X_0, \text{ and} \\ 0 & \text{if } e \in \overleftarrow{Q}_i(x_i, y_i) \text{ for some } i \in \{1, 2, \dots, \ell\}. \end{cases}$$

That is, for $1 \leq i \leq \ell$, we reverse the path $Q_i(x_i, y_i)$ connecting x_i to y_i and make the corresponding arcs available at zero cost. We then need to connect x_i to y_i without using arcs in X_0 . Since the number of terminal pairs is at most ℓ which is a constant, the Directed Steiner Forest instance $I(v_1, v_2)$ can be solved in polynomial time by the algorithm of Feldman and Ruhl given in [11]. Let $\text{OPT}(I(v_1, v_2))$ be the cost of an optimal solution to $I(v_1, v_2)$. We add a link v_1v_2 to \mathcal{A}_2 if the computed solution of $I(v_1, v_2)$ is strongly connected. This completes the construction of \mathcal{A}_2 . For a link $e \in \mathcal{A}$ we let the weight $w_e = 0$ if $e \in \mathcal{A}_1$ and $w_e = \text{OPT}(I(v_1, v_2))$ if $e \in \mathcal{A}_2$.

We now argue that a shortest path \mathcal{P} from node $s_1 = (s, \dots, s) \in \mathcal{V}$ to node $t_1 = (t, \dots, t) \in \mathcal{V}$ in \mathcal{D} corresponds to an optimal solution to I . For every link $vw \in \mathcal{P}$, we add the optimal solution to $I(v, w)$ computed by the Feldman-Ruhl algorithm to our solution Y . The algorithm runs in polynomial time for a fixed number ℓ of disjoint s - t paths, since it computes at most n^ℓ Min-cost Strongly Connected Subgraphs on ℓ terminal pairs, which can be done in polynomial time by [11]. Proving that the final algorithm is optimal is quite technical and requires another auxiliary graph and a technical lemma.

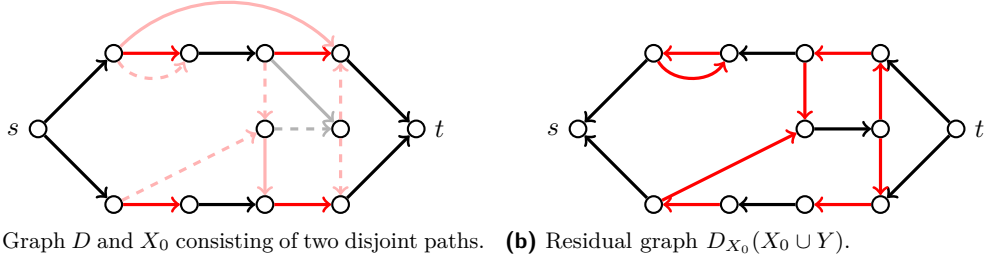
► **Theorem 17.** *The set Y computed by Algorithm 1 is an optimal solution to the instance I of Fault-Tolerant ℓ -Flow Augmentation. Furthermore, the running time is bounded by $O(|A||V|^{6\ell-2} + |V|^{6\ell-1} \log |V|)$.*

From Theorem 17 we obtain a polynomial-time 2-approximation algorithm for FT ℓ F: Let $\text{OPT}(I)$ be the cost of an optimal solution to an instance I of FT ℓ F. The algorithm first computes a minimum-cost s - t flow X_0 and then runs the algorithm for the augmentation problem using X_0 as initial arc-set. The algorithm returns the union of the arc-sets computed in the two steps. By Theorem 17 we can augment X_0 in polynomial time to a feasible solution $X_0 \cup Y$ to I . Since we pay at most $\text{OPT}(I)$ for the sets X_0 and Y , respectively, the total cost is at most $2 \text{OPT}(I)$.

► **Corollary 18.** *FT ℓ F admits a polynomial-time 2-approximation algorithm.*

The remainder of this section is devoted to sketching the proof of Theorem 17. For this purpose we need another auxiliary graph that we use as a certificate of feasibility. For a graph $H = (V, A^*)$ such that $X_0 \subseteq A^* \subseteq A$, we denote the corresponding residual graph

5:16 Fault-Tolerant Edge-Disjoint s - t Paths



■ **Figure 1** Illustration of the structure of feasible solutions to Fault-Tolerant ℓ -Flow Augmentation. Unsafe arcs are red, safe arcs are black. In Fig. 1a: edges of X_0 are black and red; edges of $A - X_0$ are light gray and light red. Dashed edges belong to Y .

by $D_{X_0}(A^*) = (V, A')$. The arc-set A' is given by $A' := \{uv \in A^* \mid uv \notin X_0\} \cup \{vu \in A^* \mid uv \in X_0\}$. An illustration of this graph is given in Figure 1. We first show that in a feasible solution $Y \subseteq A \setminus X_0$, each vulnerable arc in X_0 is contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$.

► **Lemma 19.** *Let $Y \subseteq A \setminus X_0$. Then Y is a feasible solution to I if and only if each vulnerable arc $f \in M \cap X_0$ is contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$.*

Proof. We first prove the “if” part, so let $f = uv$ be a vulnerable arc in X_0 that is contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$. Since $f \in X_0$, the arc f is reversed in $D_{X_0}(X_0 \cup Y)$ and since f is on a cycle C in $D_{X_0}(X_0 \cup Y)$, there is a path P from u to v in $D_{X_0}(X_0 \cup Y)$. Let P' be the path corresponding to P in $X_0 \cup Y$. Note that P' is not a directed path in D and that an arc e on P' is traversed forward if $e \in P' \cap Y$ and traversed backward if $e \in P' \cap X_0$. We partition P' into two disjoint parts $P'_{X_0} = P' \cap X_0$ and $P'_Y = P' \cap Y$. We now argue that $(X_0 - P'_{X_0} - f) \cup P'_Y$ contains ℓ disjoint s - t paths. Clearly, we have $(X_0 - P'_{X_0} - f) \cup P'_Y \subseteq X_0 \cup Y$. Furthermore, by our assumption that X_0 is the union of ℓ s - t edge-disjoint paths, for each vertex $v \in V - \{s, t\}$, we have $\delta^+(v) = \delta^-(v)$ and $\delta^+(s) = \delta^-(t) = \ell$. Since C is a cycle in $D_{X_0}(X_0 \cup Y)$ the degree constraints also hold for $(X_0 - P'_{X_0} - f) \cup P'_Y$. Hence $(X_0 - P'_{X_0} - f) \cup P'_Y$ is the union of ℓ disjoint s - t paths.

We now prove the “only if” part. Let $f = uv \in X_0$ be a vulnerable arc and suppose f is not contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$. Let $L \subseteq V$ be the set of vertices that are reachable from u in $D_{X_0}(X_0 \cup Y)$ and let $R = V - L$. Note that $s \in L$, since u is on some s - t path in X_0 and $t \in R$, since otherwise there is a path from u to v in $D_{X_0}(X_0 \cup Y)$ (since every arc in X_0 is reversed in $D_{X_0}(X_0 \cup Y)$). Let $L' = \{x_1, \dots, x_\ell\} \subseteq L$, $x_i \in P_i$ for $1 \leq i \leq \ell$, be the vertices of L that are closest to t in X_0 . We now claim that $\delta^+(L')$ is a cut of size ℓ in $X_0 \cup Y$ containing f . Since f is vulnerable this contradicts the feasibility of $X_0 \cup Y$. We have $f \in \delta^+(L')$ in $X_0 \cup Y$, since otherwise f is contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$. By the construction of L , we have $Y \cap \delta^+(L') = \emptyset$. Since X_0 is the union of ℓ disjoint paths, the set $\delta^+(L')$ has size at most ℓ , proving our claim, since this implies that $X_0 \cup Y$ is not feasible. ◀

Next, we sketch the proof of Theorem 17.

Proof of Theorem 17 (sketch). Let \mathcal{P} be a shortest path in the \mathcal{D} and let Y be the solution computed by Algorithm 1. Using Lemma 19 we can show the feasibility of Y .

▷ **Claim 1.** The solution Y computed by Algorithm 1 is feasible.

Let Y^* be an optimal solution to I of weight $\text{OPT}(I)$. We now show that Y is optimal. Observe that the weight of Y is equal to $c'(\mathcal{P})$, so it suffices to show that $w(\mathcal{P}) \leq \text{OPT}(I)$. To prove the inequality, we first introduce a partial ordering of the strongly connected components of $D_{X_0}(X_0 \cup Y^*)$. Using this ordering we can construct a path \mathcal{P}' in \mathcal{D} from (s, \dots, s) to (t, \dots, t) of cost $w(\mathcal{P}') = \text{OPT}(I)$. We conclude by observing that a shortest path \mathcal{P} has cost at most $w(\mathcal{P}')$.

▷ **Claim 2.** There is a path \mathcal{P}' from (s, \dots, s) to (t, \dots, t) in \mathcal{D} of cost at most $\text{OPT}(I)$.

Proof of Claim 2 (sketch). We give an algorithm that constructs a path \mathcal{P}' from (s, \dots, s) to (t, \dots, t) in \mathcal{D} such that \mathcal{P}' only uses links in \mathcal{A}_2 that correspond to strongly connected components of Y^* in $D_{X_0}(X_0 \cup Y^*)$. Starting from $s_1 = (s, \dots, s) \in \mathcal{V}$, we perform the following two steps alternatingly until we reach $(t, \dots, t) \in \mathcal{V}$.

1. From the current node u , we proceed by greedily taking links of \mathcal{A}_1 until we reach a node $v = (v_1, v_2, \dots, v_\ell) \in \mathcal{V}$ with the property that each vertex v_i , $1 \leq i \leq \ell$, is either t or part of some strongly connected component of $D_{X_0}(X_0 \cup Y^*)$.
2. From the current node v , we take a link $vw \in \mathcal{A}_2$ to some node $w \in \mathcal{V}$, where the link vw corresponds to a strongly connected component Z of $D_{X_0}(X_0 \cup Y^*)$.

It can be shown that this algorithm indeed finds the desired solution. ◁

Finally, using the algorithm in [11] for finding a cost-minimal strongly connected subgraph on ℓ terminal pairs, we obtain that Algorithm 1 runs in time $O(mn^{6\ell-2} + n^{6\ell-1} \log n)$. ◀

3.3 Relation to a Problem of Open Complexity

In the previous section we obtained a 2-approximation for Fault-Tolerant ℓ -Flow. Ideally, one would like to complement this with a hardness (of approximation) result. However, since Fault-Tolerant ℓ -Flow Augmentation admits a polynomial-time algorithm according to Theorem 17, we cannot use the augmentation problem in order to prove NP-hardness of Fault-Tolerant ℓ -Flow; an approach that has been used successfully for instance for robust paths [3], robust matchings [16] and robust spanning trees [2]. Hence, there is some hope that Fault-Tolerant ℓ -Flow might actually be polynomial-time solvable. Here, we show that a polynomial-time algorithm for Fault Tolerant 2-Flow implies polynomial-time algorithms for 1-2-connected Directed 2 Steiner Tree, a problem whose complexity is open [9].

The problem 1-2-connected Directed 2 Steiner Tree is defined as follows. Given a digraph $D = (V, E)$ with costs $c \in \mathbb{Q}^A$, a root vertex $s \in V$, and two terminals $t_1, t_2 \in V$, the goal is to find a minimum cost set $X \subseteq E$ such that X contains two disjoint s - t_1 paths and one s - t_2 path.

► **Proposition 20.** *A polynomial-time algorithm for Fault Tolerant 2-Flow implies a polynomial-time algorithm for 1-2-connected Directed 2 Steiner Tree.*

Proof. Let I be an instance of 1-2-connected Directed 2 Steiner Tree on a graph $D = (V, A)$ with edge-weights $c \in \mathbb{Q}^E$, root $s \in V$, and terminals $T = \{t_1, t_2\}$. We construct an instance I' of Fault Tolerant 2-Flow as follows. We add to D two vertices u and t and four directed edges $\hat{A} = \{(s, u), (t_1, u), (u, t), (t_2, t)\}$. Let the resulting graph be D' . The edge weights c' of D' are given by

$$c'_e := \begin{cases} c_e & \text{if } e \in A(G), \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we let $M := A \cup \{(s, u), (t_1, u)\}$, that is, the edges incident to t are safe while all other edges are unsafe.

Let X be a feasible solution to I' . We have $\hat{A} \subseteq X$, since otherwise X is not feasible. We now show that there is at least one s - t_i path and there are at least two disjoint s - t_2 paths in $(V, X \setminus \hat{A})$. Assume first that there is no path from s to t_1 in $(V, X \setminus \hat{A})$. But then $\{(s, u), (v, t)\}$ is a cut of size two in D' , where (s, u) is a vulnerable edge. This contradicts the feasibility of X . Now assume that there are no two disjoint s - t_2 paths in $(V, X \setminus \hat{A})$. It is not hard to see that then we have a contradiction to the feasibility of X . Finally, observe that there is a one-to-one correspondence between feasible solutions to I and I' . ◀

4 Conclusions and Future Work


We introduced the two problems FTP and FTF, which add a non-uniform fault model to the classical edge-disjoint s - t paths problem. This fault-model leads to a dramatic increase in the computational complexity. We gave polynomial-time algorithms for several classes of instances including the case $k = 1$ and DAGs with fixed k . Furthermore, we proved a tight bound on the integrality gap of a natural LP relaxation for FTP and obtained a polynomial k -approximation algorithm. For FTF, our main result is a 2-approximation algorithm for fixed ℓ . One of the main open problems is to see whether the approximation guarantee for FTP can be improved to the approximation guarantees of the best known algorithms for the Steiner Tree problem. Furthermore, it would be interesting to see if the methods employed in the current paper for 1-FTP and k -FTP on directed acyclic graphs can be extended to k -FTP on general graphs. Another intriguing open question is whether Fault-Tolerant ℓ -Flow is NP-hard, which is not known even for $\ell = 2$. We showed that a positive result in this direction implies polynomial-time algorithms for two Steiner problems whose complexity status is open.

References

- 1 David Adjiashvili. Non-uniform robust network design in planar graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2015.
- 2 David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible graph connectivity. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 13–26. Springer, 2020.
- 3 David Adjiashvili, Sebastian Stiller, and Rico Zenklusen. Bulk-robust combinatorial optimization. *Mathematical Programming*, 149(1-2):361–390, 2015. doi:10.1007/s10107-014-0760-6.
- 4 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Approximation complexity of min-max (regret) versions of shortest path, spanning tree, and knapsack. In *European Symposium on Algorithms*, pages 862–873. Springer, 2005.
- 5 Sylvia Boyd, Joseph Cheriyan, Arash Haddadan, and Sharat Ibrahimpur. A 2-approximation algorithm for flexible graph connectivity. *arXiv preprint*, 2021. arXiv:2102.03304.
- 6 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012.
- 7 Deeparnab Chakrabarty, Chandra Chekuri, Sanjeev Khanna, and Nitish Korula. Approximability of capacitated network design. *Algorithmica*, 72(2):493–514, 2015.
- 8 Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed Steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.
- 9 Joseph Cheriyan, Bundit Laekhanukit, Guylain Naves, and Adrian Vetta. Approximating rooted steiner networks. *ACM Transactions on Algorithms (TALG)*, 11(2):1–22, 2014.
- 10 Kedar Dhamdhere, Vineet Goyal, R Ravi, and Mohit Singh. How to pay, come what may: Approximation algorithms for demand-robust covering problems. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 367–376. IEEE, 2005.

- 11 Jon Feldman and Matthias Ruhl. The directed Steiner network problem is tractable for a constant number of terminals. *SIAM Journal on Computing*, 36(2):543–561, 2006. doi: 10.1137/S0097539704441241.
- 12 Harold N Gabow and Suzanne R Gallagher. Iterated rounding algorithms for the smallest k -edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012.
- 13 Daniel Golovin, Vineet Goyal, Valentin Polishchuk, R Ravi, and Mikko Sysikaski. Improved approximations for two-stage min-cut and shortest path problems under uncertainty. *Mathematical Programming*, 149(1-2):167–194, 2015.
- 14 Jiong Guo, Rolf Niedermeier, and Ondřej Suchý. Parameterized complexity of arc-weighted directed Steiner problems. *SIAM Journal on Discrete Mathematics*, 25(2):583–599, 2011.
- 15 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 585–594, 2003. doi: 10.1145/780542.780628.
- 16 Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. How to secure matchings against edge failures. *SIAM Journal on Discrete Mathematics*, 35(3):2265–2292, 2021.
- 17 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.
- 18 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.
- 19 Gang Yu and Jian Yang. On the robust shortest path problem. *Computers and Operations Research*, 25(6):457–468, June 1998.
- 20 Paweł Zieliński. The computational complexity of the relative robust shortest path problem with interval data. *European Journal of Operational Research*, 158(3):570–576, November 2004. doi:10.1016/s0377-2217(03)00373-4.


An Improved ε -Approximation Algorithm for Geometric Bipartite Matching

Pankaj K. Agarwal 


Duke University, Durham, NC, USA

Sharath Raghvendra 

Virginia Tech, Blacksburg, VA, USA

Pouyan Shirzadian 

Virginia Tech, Blacksburg, VA, USA

Rachita Sowle 

Virginia Tech, Blacksburg, VA, USA

Abstract

For two point sets $A, B \subset \mathbb{R}^d$, with $|A| = |B| = n$ and $d > 1$ a constant, and for a parameter $\varepsilon > 0$, we present a randomized algorithm that, with probability at least $1/2$, computes in $O(n(\varepsilon^{-O(d^3)} \log \log n + \varepsilon^{-O(d)} \log^4 n \log^5 \log n))$ time, an ε -approximate minimum-cost perfect matching under any L_p -metric. All previous algorithms take $n(\varepsilon^{-1} \log n)^{\Omega(d)}$ time. We use a randomly-shifted tree, with a polynomial branching factor and $O(\log \log n)$ height, to define a tree-based distance function that ε -approximates the L_p metric as well as to compute the matching hierarchically. Then, we apply the primal-dual framework on a compressed representation of the residual graph to obtain an efficient implementation of the Hungarian-search and augment operations.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Euclidean bipartite matching, approximation algorithms, primal dual method

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.6

Funding Work on this paper was supported by NSF under grant CCF 1909171.

1 Introduction

Let $A, B \subset \mathbb{R}^d$ be two point sets of size n each, where $d > 1$ is a constant, and $d(\cdot, \cdot)$ a metric. Let $\mathcal{G} = (A \cup B, A \times B)$ be a weighted complete bipartite graph in which the cost of an edge $(a, b) \in A \times B$ is $d(a, b)$. A *matching* in \mathcal{G} is a set of vertex-disjoint edges in \mathcal{G} . A *perfect matching* in \mathcal{G} is a matching of size n . The cost of a matching M , denoted by $c(M)$, is $c(M) = \sum_{(a,b) \in M} d(a, b)$. The *minimum-cost perfect matching* in \mathcal{G} , denoted by M^* , is a perfect matching in \mathcal{G} of the minimum cost. For any $\varepsilon > 0$, a perfect matching M in \mathcal{G} is called an ε -approximate matching if $c(M) \leq (1 + \varepsilon)c(M^*)$. We consider the case where the cost $d(a, b)$ is the L_p distance denoted by $\|a - b\|_p$. The optimal transport (OT) distance between two (possibly continuous) distributions can be estimated by taking n samples from both distributions and then computing their minimum-cost perfect matching. The wide applicability of OT in Machine Learning and Computer Vision [17, 20, 4] has motivated the design of fast exact and approximation algorithms that compute a minimum-cost perfect matching. In this paper, for the L_p -norm, we present a new algorithm to computing an ε -approximate matching.

Related work. For an arbitrary weighted bipartite graph with n vertices and m edges, the Kuhn-Munkres algorithm [13] computes a minimum-weight bipartite matching in a weighted bipartite graph in $O(mn + n^2 \log n)$ time. For bipartite graphs with non-negative integer costs bounded by C , Gabow and Tarjan [9] gave an $O(m\sqrt{n} \log(nC))$ -time algorithm. Both



© Pankaj K. Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 6; pp. 6:1–6:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the Hungarian and Gabow-Tarjan algorithms are combinatorial algorithms that iteratively find an augmenting path and augment the matching along this path. The augmenting paths are chosen such that the increase in the matching cost after each augmentation is minimized. This cost increase is referred to as the *net-cost* of the path. Alternate approaches such as the electrical flow [21] method and the matrix multiplication based methods [15] can be used to obtain fast matching algorithms. The current best known execution time is $\tilde{O}(m + n^{1.5})$.

When $A \cup B$ is a 2-dimensional point set in the Euclidean space, a Euclidean minimum-weight matching (EMWM) can be computed in $O(n^2 \text{polylog } n)$ time [12], and in $O(n^{3/2} \text{polylog } n)$ time when the points have integer coordinates [19, 18]. For this case, it is easy to compute a $O(\log n)$ -approximate matching in expectation using a randomly shifted quad-tree [6, 7]. Agarwal and Varadarajan [1] used the shifted quad-tree to compute an $O(\log 1/\delta)$ -approximate solution in $O(n^{1+\delta})$ time. Following this, there were several results that used such a decomposition; see for instance [11, 3, 8]. The current best-known approximation algorithm for computing EMWM is by Raghvendra and Agarwal [16], which computes an ε -approximate matching with high probability in $n(\varepsilon^{-1} \log n)^{O(d)}$ time. In their algorithm, each cell \square of a randomly shifted quad-tree Q is decomposed by a uniform grid into $(\log n/\varepsilon)^{O(d)}$ subcells. The Euclidean distance between any pair of points u, v with \square as their least common ancestor in Q is ε -approximated by the distance between the subcells of \square that contain u and v respectively. Their algorithm uses Q to compute a minimum net-cost augmenting path P with respect to the new distance and augment the matching along this path, both in time $O(|P| \text{polylog } n)$. They obtain a near-linear execution time by bounding the total length of all augmenting paths by $O(\varepsilon^{-1} n \log n)$. To compute these paths quickly, they compress the residual graph inside \square into a graph of $(\log n/\varepsilon)^{O(d)}$ size and execute Bellman-Ford algorithm on this graph. Lahn and Raghvendra [14] extended this framework to approximate the 2-Wasserstein distance of planar point sets, i.e., an approximate minimum-cost matching when $d(u, v)$ is $\|u - v\|_2^2$. Unlike Euclidean distance, approximating the squared-Euclidean distance using Q results in a polynomial sized compressed residual graph at each cell. Since using Bellman-Ford algorithm on such a compressed graph can be prohibitively expensive, they introduce a novel primal-dual framework and define *compressed feasibility* on the compressed residual graph. Using this framework, they are able to find an augmenting path as well as augment it along this path in sub-linear time. Consequently, they achieve an $O(n^{5/4} \text{poly}(\log n, 1/\varepsilon))$ time algorithm for the 2-Wasserstein distance between planar point sets. Recently, Agarwal et al. [2] have designed a deterministic algorithm that uses multiple quadtrees to compute a $(1 + \varepsilon)$ -approximate Euclidean matching in $n(\varepsilon^{-1} \log n)^{O(d)}$ time.

Our result. The following theorem states our main result.

► **Theorem 1.** *Let A, B be two point sets in \mathbb{R}^d of size n each, for a constant $d > 1$, and let $0 < \varepsilon \leq 1$ be a parameter. With probability at least $1/2$, an ε -approximate matching under any L_p -metric can be computed in $O(n(\varepsilon^{-O(d^3)} \log \log n + \varepsilon^{-O(d)} \log^4 n \log^5 \log n))$ time.*

For the sake of simplicity, we describe the algorithm for the Euclidean metric. It can be extended to other L_p -metrics in a straight forward manner. For any two points a and b , we use $\|a - b\|$ to denote the Euclidean distance between them. Using standard techniques [16, 14], we can preprocess the input points in $O(n \log n)$ time so that the point sets A and B satisfy the following conditions: (P1) All input points have integer coordinates bounded by $n^{O(1)}$. (P2) No integer grid point contains points of both A and B . (P3) $c(M^*) \in \left[\frac{3\sqrt{dn}}{\varepsilon}, \frac{9\sqrt{dn}}{\varepsilon} \right]$. Details of how we preprocess A and B can be found in the Appendix (Section A). Assuming A and

B satisfy (P1)–(P3), we present an algorithm that, with probability $1/2$, computes an $(\varepsilon/2)$ -approximate matching in $O(n(\varepsilon^{-O(d^3)} + \varepsilon^{-O(d)} \log^4 n \log^4 \log n))$ time. The preprocessing step adds an additional $\log \log n$ factor to the running time of the algorithm, resulting in the running time mentioned in Theorem 1. In the following, we provide an overview of our approach and its comparison with existing work.

As in [16, 14], we also define a tree based distance $d_T(\cdot, \cdot)$ that approximates the Euclidean distance (Section 2). Unlike [16, 14] that use a quad-tree of height $O(\log n)$, we build a tree T of height $O(\log \log n)$ (see Section 2.1). Each cell of T at level i (root is assigned level 0) with a side length of ℓ_i is partitioned using a randomly-shifted grid into children whose side-length $\ell_{i+1} = \ell_i^c$ where $c < 1$ is a constant that depends only on d . Given that the point set have integer coordinates bounded by $n^{O(1)}$ (from (P1)), the height of T is $h = O(\log \log n)$. For any pair of points (u, v) with a cell \square of level i as its least common ancestor, let \square_u and \square_v be the children of \square that contain u and v respectively. As in the case of a randomly shifted-quadtree where we get an $O(h) = O(\log n)$ approximation, one can show that the distance between the centers of \square_u and \square_v is a $O(h) = O(\log \log n)$ approximation of the Euclidean distance (in expectation). We obtain a refined $(1 + \varepsilon)$ -approximation of the Euclidean distance by partitioning \square_u and \square_v into finer subcells and then using the distance between the centers of those sub-cells that contain u and v . As in [16, 14], one can divide each cell into $O(h^d)$ many subcells and obtain a $(1 + \varepsilon)$ -approximation of the Euclidean distance. With $h = \log \log n$, this will result in an execution time of $\Omega(n \log^4 n (\varepsilon^{-1} \log \log n)^{d^3})$. Instead, we partition a cell into subcells more carefully (See the definition of subcells in Section 2.2). Intuitively, we make the number of subcells a function of the height of the cell, i.e., smaller cells have significantly fewer than $\log^{O(d)} \log n$ subcells. As a result, we are able to improve the dependence of our algorithm from $\log^{O(d^3)} \log n$ to $\log^{O(d)} \log n$. Interestingly, we show that the expected distortion is higher for cells that are closer to the leaves. Nonetheless, we are able to bound the expected error of our distance between any two points u and v by $\varepsilon \|u - v\|$ (See Lemma 3).

Similar to [14], our algorithm compactly stores the residual graph (Section 5.2) as well as the dual weights (Section 5.3) and uses this compact representation to efficiently find augmenting paths. The size of the compressed residual graph inside any cell is bounded by the side-length of its child, i.e., smaller cells have a smaller compressed graph (Lemma 14). As a result, finding augmenting paths in smaller cells is significantly faster than that in larger ones. In our analysis, we show that most of the augmenting paths in the algorithm are found in smaller cells which can be computed quickly. In particular, only $O(\frac{n}{\varepsilon^{\ell_{i+1}}})$ augmenting paths are found inside a compressed graph at level i , each of which can be found in $O(\ell_{i+1} \log^2 n)$ time. Combining across all $O(\log \log n)$ levels, we get a near-linear execution time.

Typical matching algorithms that are based on a compressed residual graph modify the dual weights and find an augmenting path with respect to current matching M . The algorithms presented in [16, 14] classify edges into *local* and *non-local* which they use critically in computing a minimum net-cost augmenting path. We remove the need for this classification and make our algorithm and its analysis simpler. Instead of using the classification, our algorithm carefully updates the dual weights, possibly modifies a matching M to another matching M' of the same size and cost, and finds an augmenting path with respect to the new matching M' .

2 Hierarchical Partition and the Distance Function

In this section, we present a randomized hierarchical partitioning of space, used to define a new distance function $d_T(\cdot, \cdot)$ that approximates the Euclidean metric (in expected sense) as well as to guide the construction of matching (in a hierarchical manner).

2.1 Hierarchical Partitioning

For a value $\ell > 0$, let $G[\ell]$ be the d -dimensional uniform grid with cell side-length ℓ , i.e., $G[\ell] = (\ell\mathbb{Z})^d + [0, \ell]^d$. For a point $x \in \mathbb{R}^d$, we use $G[\ell] + x$ to denote the translate of $G[\ell]$ by x . For any rectangle R , let $A_R = A \cap R$ and $B_R = B \cap R$. We say that R is *non-empty* if $A_R \cup B_R \neq \emptyset$. Given R and a grid G , let $C[R, G]$ denote the set of non-empty rectangles in the rectangular subdivision of R induced by G . If G is fixed or clear from the context, we use $C[R]$ to denote $C[R, G]$. By abusing the notation slightly, we use $C[R]$ to denote the subdivision as well as the set of non-empty rectangles in the subdivision. For a non-empty rectangle R , we designate one of the points in $A_R \cup B_R$, say r_R , as its *representative*.

Let \square_0 be the smallest axis-aligned hypercube that contains $A \cup B$. Let ℓ_0 be its side length. By property (P1), $\ell_0 = n^{O(1)}$. We construct a hierarchical partition and the associated tree T , as follows. Each node in T is associated with a non-empty rectangle which we refer to as a *cell* and we will not distinguish between the two. The *level* of a node (and the corresponding rectangle) is the length of the path in T from the root. The root of T is \square_0 and its level is 0. Set $\alpha = 1 - \frac{1}{8d+2}$. For $i > 0$, set $\ell_i = \ell_{i-1}^\alpha$. Let $h > 0$ be the smallest integer such that $\ell_h \leq (\varepsilon^{-1}d)^{\frac{1}{(1-\alpha)^2}}$. Any cell \square in T of level h is designated as a leaf node. By construction, $h = O(\log \log n)$. The choice for the condition of the leaf node will become apparent in Section 2.2. Otherwise, we choose a random point, $\xi_\square \in [0, \ell_{i+1}]^d$ and set $G = G[\ell_{i+1}] + \xi_\square$. Let $C[\square] := C[\square, G]$ be the subdivision of \square induced by G . Each rectangle $\square' \in C[\square]$ is a level $i+1$ node. We create a child of \square in T for each non-empty rectangle $\square' \in C[\square]$ and recursively construct the partition and associated sub-tree of \square' . For any $0 \leq i \leq h-1$, let $\Delta[i]$ denote all cells of T of level i .

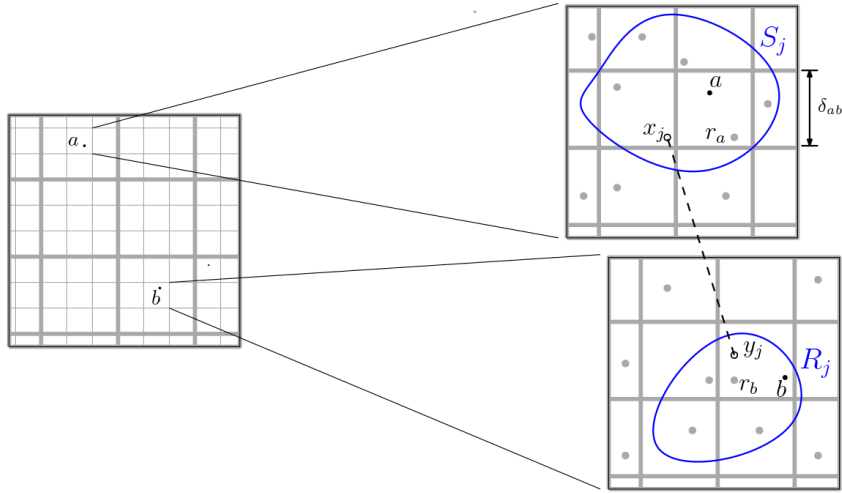
2.2 Euclidean Distance Approximation

For any pair of points $(a, b) \in A \times B$, let \square be the least common ancestor of a and b in T , i.e., the cell with the highest level that contains a and b . Let the level of \square be i . We define the *level* of (a, b) , $\text{lev}(a, b)$, to be i and refer to (a, b) as a *level i edge*. Let \square_a and \square_b be the children of \square that contain a and b respectively. We divide \square_a and \square_b into $O(\varepsilon^{-d}(h-i)^{2d})$ *subcells* and show that the distance between any two points in the subcells that contain a and b is a $(1+\varepsilon)$ -approximation of $\|a-b\|$. Note that the smaller cells, i.e., those at a higher level, have fewer subcells. Using this and the bound on the side-length of any leaf node of T , we can bound the number of subcells of the children of \square by ℓ_{i+1} (independent of $\log \log n$). In Section 5, we use this fact to compress the residual graph more efficiently.

Subcells. Any cell \square is divided into subcells as follows. For each $0 < i < h$, set

$$\mu_i = \frac{\varepsilon}{c_2 \sqrt{d}(h-i)^2} \ell_i, \quad (1)$$

where $c_2 = 24\pi^2$. We set $\mu_0 = \infty$ and $\mu_h = 0$. A *subcell* is formed by combining a subset of children of \square such that the diameter of points in these children is no more than μ_i . Set $\bar{\mu}_i = \left\lfloor \frac{\mu_i}{\sqrt{d}\ell_{i+1}} \right\rfloor \ell_{i+1}$ (By Lemma 2 below, $\left\lfloor \frac{\mu_i}{\sqrt{d}\ell_{i+1}} \right\rfloor$ is a positive integer). For any non-leaf



■ **Figure 1** Euclidean distance approximation: The grey rectangular subdivision shows the children of \square (on left) and the bold grey rectangular subdivision shows the subcells.

and non-root cell of level i in the tree T , let $G_\square = G[\bar{\mu}_i] + \xi_\square$, where ξ_\square is the random shift for the grid constructed in \square . By construction, the boundary of grid cells in G_\square are aligned with those of $G[\ell_{i+1}] + \xi_\square$ (See Figure 1). Therefore, each non-empty child cell of \square lies in exactly one subcell of subdivision $C[\square, G_\square]$. Let S_\square be this set of non-empty subcells, and let R_\square be the set of representative points of S_\square , i.e., $R_\square = \{r_R \mid R \in S_\square\}$. It is clear that the diameter of each subcell is at most μ_i . The following lemma relates the side-length of the children of a cell to the diameter of the subcells of that cell.

► **Lemma 2.** *For any $0 \leq i < h$, $\ell_{i+1} \leq \mu_i / \sqrt{d}$.*

WSPD. For any cell \square , the number of pairs of children subcells of \square can be prohibitively large. We use a well-separated pair decomposition (WSPD) to compactly store these pairs. For simplicity of the algorithm, similar to [14], we use WSPD to define our Euclidean distance approximation. For a point set X , let DIAM_X be the distance between the farthest pair of points in X . For any $\varepsilon > 0$, two point sets X and Y are called ε -well-separated if for all $x \in X$ and $y \in Y$, $\max\{\text{DIAM}_X, \text{DIAM}_Y\} \leq (\varepsilon/12)\|x - y\|$. Given a point set $X \in \mathbb{R}^d$ of size n and a parameter $\varepsilon > 0$, an ε -WSPD (or simply WSPD for brevity) of X is a set $\mathcal{W} = \{(R_1, S_1), \dots, (R_k, S_k)\}$ such that (i) each (R_i, S_i) is ε -well separated, (ii) for each pair of points $(u, v) \in X \times X$, there is a unique pair $(R_i, S_i) \in \mathcal{W}$ such that $(u, v) \in R_i \times S_i$ or $(u, v) \in S_i \times R_i$, and (iii) $k = O(\varepsilon^{-d}n)$. Also, if the spread of X , the ratio of the largest to smallest pairwise distances, is bounded by $n^{O(1)}$, then every point of X participates in $O(\varepsilon^{-d} \log n)$ pairs of \mathcal{W} . \mathcal{W} can be constructed in $O(n \log n + \varepsilon^{-d}n)$ time [5, 10]. For any $(R_i, S_i) \in \mathcal{W}$, we choose an arbitrary pair $(x_i, y_i) \in R_i \times S_i$ and make this pair its *representative pair*.

For any non-leaf cell $\square \in T$, let $X_\square = \bigcup_{\square' \in C[\square]} R_{\square'}$ be the set of representative points of all non-empty subcells of the children of \square . We build an ε -WSPD $\mathcal{W}_\square = \{(R_1, S_1), \dots, (R_k, S_k)\}$ on X_\square . For a leaf cell \square , we construct an ε -WSPD \mathcal{W}_\square on $A_\square \cup B_\square$.

Distance function. We are now ready to define the distance function $d_T : A \times B \mapsto \mathbb{R}_{\geq 0}$. For any pair of points $(a, b) \in A \times B$ of level i with \square as its least common ancestor, if $i = h$, we set $\delta_{ab} = 0$ and if $i < h$, we set $\delta_{ab} = \mu_{i+1}$. For $i = h$, we set (R_j, S_j) to be the pair in

\mathcal{W}_\square such that $(a, b) \in R_j \times S_j$. For $i < h$, (R_j, S_j) is defined as follows. Let \square_a (resp. \square_b) be the child of \square that contains a (resp. b), and let ξ_{ab}^a (resp. ξ_{ba}^b) be the subcell of \square_a (resp. \square_b) that contains a (resp. b). Let r_a and r_b be the representatives of ξ_{ab}^a and ξ_{ba}^b respectively. Let (R_j, S_j) be the unique ε -well separated pair of \mathcal{W}_\square such that $(r_a, r_b) \in R_j \times S_j$. We say that (a, b) is *covered* by (R_j, S_j) . Now let (x_j, y_j) be the representative pair of (R_j, S_j) (See Figure 1). Then, we define

$$d_\tau(a, b) = (1 + \varepsilon/4)\|x_j - y_j\| + 2\delta_{ab}. \quad (2)$$

Unlike in [16, 14], we create fewer subcells for cells that are closer to the leaves, which results in a larger distortion for the edges within these cells. However, Lemmas 3 and 4 together establish that the expected distortion on any pair of points $(a, b) \in A \times B$ is still proportional to $\varepsilon\|a - b\|$. See Appendix B for the proof.

► **Lemma 3.** *For any pair of points $(a, b) \in A \times B$, $\mathbb{E}[\delta_{ab}] \leq \frac{\pi^2}{6c_2}\varepsilon\|a - b\|$.*

Using Lemma 3, we show in Lemma 4 that our distance function, in expectation, approximates the Euclidean distance within a factor of $(1 + \varepsilon)$.

► **Lemma 4.** *For any pair of points $(a, b) \in A \times B$, $d_\tau(a, b) \geq \|a - b\|$. Furthermore, $\mathbb{E}[d_\tau(a, b)] \leq (1 + (\frac{5\pi^2}{6c_2} + \frac{11}{24})\varepsilon)\|a - b\|$.*

3 Preliminaries

We begin by presenting notations pertaining to the distance function that will help us describe our algorithm. For any subset $E \subseteq A \times B$ of edges, we use $w(E) = \sum_{(u,v) \in E} d_\tau(u, v)$ to denote the sum of the weights of the edges in E with respect to $d_\tau(\cdot, \cdot)$.

Next, we describe definitions related to matching that will assist us in presenting our algorithm. Let M be any matching in \mathcal{G} . A vertex is *free* with respect to M if it has no edges of M incident on it. An *alternating* path with respect to M is a simple path in \mathcal{G} whose edges alternate between those in M and not in M . An *augmenting* path is an alternating path whose endpoints are free. We can *augment* M along an augmenting path P by simply setting $M \leftarrow M \oplus P$. For any matched vertex $u \in A \cup B$, let $m(u)$ denote the vertex to which u is matched in M . For any edge (u, v) , we define the *net-cost* $\phi(u, v)$ of (u, v) as follows: $\phi(u, v) = d_\tau(u, v) + \delta_{uv}$ if $(u, v) \notin M$, and $\phi(u, v) = -d_\tau(u, v)$ if $(u, v) \in M$. For a set $E \subseteq A \times B$ of edges, we define its *net-cost* as $\phi(E) = \sum_{(u,v) \in E} \phi(u, v)$.

A *residual graph* \mathcal{G}_M is a directed bipartite graph that has the same set of edges as \mathcal{G} and for any matching (resp. non-matching) edge $(a, b) \in A \times B$, it is directed from a to b (resp. b to a). We refer to the *weight* of (a, b) in \mathcal{G}_M to be its net-cost. It is easy to see that any simple directed path P in \mathcal{G}_M alternates between matching and non-matching edges and therefore, P is an alternating path. For any rectangle R , let M_R be the subset of the edges of M with both endpoints inside R , and let \mathcal{G}_M^R denote the residual graph on $A_R \cup B_R$ with respect to the matching M_R .

Similar to the Hungarian algorithm, our algorithm assigns a dual weight $y(v) \geq 0$ to each vertex $v \in A \cup B$. We say that a matching M and a set of non-negative dual weights $y(\cdot)$ are *feasible* if, for every directed edge (u, v) of \mathcal{G}_M , $y(u) - y(v) \leq \phi(u, v)$. The presence of δ_{uv} in the definition of $\phi(u, v)$ makes our feasibility conditions a relaxed form of the feasibility conditions of the Hungarian algorithm. It can be shown that a feasible perfect matching is (in expectation) a $(1 + \varepsilon/2)$ -approximation of the minimum-cost Euclidean matching.

► **Lemma 5.** *Suppose a perfect matching M along with a set of dual weights $y(\cdot)$ are feasible and let M^* denote an optimal matching with respect to the Euclidean cost $c(\cdot)$. Then, $\mathbb{E}[c(M)] \leq \mathbb{E}[w(M)] \leq (1 + \varepsilon/2)c(M^*)$.*

For any directed edge (u, v) of \mathcal{G}_M , we define its *slack* as $s(u, v) = \phi(u, v) - y(u) + y(v)$. Based on the definition of feasibility, it is clear that $s(u, v) \geq 0$. An edge (u, v) of \mathcal{G}_M is called *admissible* if $s(u, v) = 0$. Given a feasible matching M , we can use the definition of slack to relate the weight of any directed path P from u to v in \mathcal{G}_M to the slack on its edges:

$$\phi(P) = \sum_{(u',v') \in P} (y(u') - y(v') + s(u', v')) = y(u) - y(v) + \sum_{(u',v') \in P} s(u', v'). \quad (3)$$

We present a slow yet simple implementation to find a $(1 + \varepsilon)$ -approximate matching, which is basically the Hungarian algorithm. Initialize for every $v \in A \cup B$, $y(v) = 0$ and $M \leftarrow \emptyset$. At each step, we find an augmenting path in the residual graph as follows. We set the edge weights in the residual graph to be their slacks. Next, starting from the free vertices of B , we execute the Dijkstra's shortest-path algorithm (also called the *Hungarian Search*) in this graph. For any $v \in A \cup B$, let κ_v be the shortest path from any free vertex of B to v . The algorithm returns the augmenting path P to a free vertex a of A that minimizes κ_a ; i.e, the minimum net-cost augmenting path. We update the dual weight of every vertex v with $\kappa_v < \kappa_a$ by setting $y(v) \leftarrow y(v) - \kappa_v + \kappa_a$, making all edges of P admissible. Finally, we set $M = M \oplus P$. Augmenting the matching along P keeps the matching feasible. In the following lemma, we state two observations of this algorithm.

- **Lemma 6.** *During the execution of the algorithm described above,*
- (i) *augmenting paths are computed in increasing order of their net-costs; and*
 - (ii) *if the net-cost of an augmenting path P is less than μ_i , then P does not contain any edge of level lower than i (Recollect that any such edge has a cost of at least μ_i).*

Lemma 6 suggests that we can search for augmenting paths in residual graph inside the cells of $\Delta[i]$ until the net-cost of the augmenting path reaches μ_i .

The implementation described above requires n executions of Dijkstra's algorithm, each taking $\Theta(n^2)$ time. In the next two sections, we use the properties of this algorithm (Lemma 6) to present an efficient implementation of a variant of the above algorithm.

4 Overview of the Algorithm

We present our algorithm assuming the existence of three procedures, namely, BUILD, HUNGARIANSEARCH, and AUGMENT procedures. The details of these procedures is deferred to Section 5.

Our algorithm computes a feasible perfect matching by processing the cells of T in decreasing order of their levels. Initially $i \leftarrow h - 1$ and $M \leftarrow \emptyset$ (no matching is computed at level h). For any cell $\square \in \Delta[i]$, the algorithm executes the following steps:

- If $i < h - 1$, the algorithm calls the BUILD(\square, M) procedure. This step builds a compact representation of the residual graph (defined in Section 5.2).
- The algorithm does the following iteratively: It calls the HUNGARIANSEARCH(\square, M) procedure. This procedure returns NULL if there is no augmenting path in \mathcal{G}_M^\square of a net-cost less than μ_i (see (1)). In this case, the algorithm stops processing \square . Otherwise, if there is an augmenting path, the procedure updates the dual weights $y(\cdot)$ and may update $M \leftarrow M'$ where M' is another matching of equal size such that $y(\cdot), M'$ is feasible and $w(M) = w(M')$. Then it returns a minimum net-cost augmenting path P with respect to the updated matching. The algorithm calls AUGMENT(\square, P, M) to augment M along P .

After all cells in $\Delta[i]$ are processed, if $i = 0$, the algorithm returns the matching M . Otherwise, it sets $i \leftarrow i - 1$ and continues.

Execution time of the procedures. The BUILD, HUNGARIANSEARCH, and AUGMENT procedures presented in Section 5 have the following execution time. For any cell \square , let $n_\square = |A_\square \cup B_\square|$. If \square has a level $i < h - 1$, the BUILD procedure takes $n_\square \log^3 n (\varepsilon^{-1} \log \log n)^{O(d)}$ time. Next, we present the running time of HUNGARIANSEARCH and AUGMENT procedures.

For any cell \square , if \square is at level $h - 1$ of T , then the HUNGARIANSEARCH and AUGMENT procedures takes $\varepsilon^{-O(d^3)}$ time. Otherwise, let $i = \text{lev}(\square) < h - 1$. For $j > i$, let k_j be the number of level j cells that contains at least one vertex of P . If HUNGARIANSEARCH returns NULL, $k_j = 0$. HUNGARIANSEARCH takes

$$O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^2 \log n + \sum_{j=i+1}^{h-1} k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$$

time and the total time taken by AUGMENT is $O(\sum_{j=i+1}^{h-1} k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$.

Invariants. In Section 5, we also show that the three procedures maintain the following two invariants while processing cells at level i . At any point, for the matching M , there is a set of dual weights $y(\cdot)$ such that

(J1) $M, y(\cdot)$ is feasible, and,

(J2) For any vertex $u \in B$, $y(u) \leq \mu_i$. Furthermore, if u is a free vertex of B , its dual weight $y(u) \geq \mu_{i+1}$. If u is a free vertex of A , its dual weight $y(u) = 0$.

Our procedures will not maintain dual weights $y(\cdot)$ explicitly but only guarantee the existence of dual weights that satisfy (J1) and (J2). From (J1), (J2), and (3), we get the following:

► **Corollary 7.** *For any $i \geq 0$, while the algorithm is processing level i cells, the net-cost of any augmenting path in \mathcal{G}_M is at least μ_{i+1} .*

4.1 Analysis of the algorithm

Note that at the root cell \square_0 , $\mu_0 = \infty$ and therefore, the second step of the algorithm terminates only when there are no more augmenting paths in \mathcal{G}_M ; i.e, M is perfect. Since $M, y(\cdot)$ is also feasible, from Lemma 5, it is (in expectation) a $(1 + \varepsilon)$ -approximate matching. Let W be the cost of the matching returned by our algorithm. From (P3) and the fact that $\varepsilon \leq 1$, we get that, with probability at least $1/2$,

$$W = \Theta(n/\varepsilon). \tag{4}$$

Next, using the execution time of the procedures and the invariants they maintain, we bound the execution time of our algorithm.

We introduce a few notation that helps us analyze our algorithm. Recollect that $n_\square = |A_\square \cup B_\square|$ and M^* denotes the optimal matching of $A \cup B$ with respect to the Euclidean costs. Let $\mathbb{P} = \langle P_1, \dots, P_n \rangle$ be the sequence of augmenting paths computed by the algorithm in the order in which they were computed. Let $M_0 = \emptyset$ and let M_i be the matching after augmenting along the path P_i .

Efficiency analysis. We begin by bounding the time taken by $\text{BUILD}(\square, M)$ across all $O(\log \log n)$ levels of T by

$$\sum_{i=0}^{h-1} \sum_{\square \in \Delta[i]} n_{\square} \log^3 n_{\square} (\varepsilon^{-1} \log \log n)^{O(d)} = O(n \log^3 n (\varepsilon^{-1} \log \log n)^{O(d)}). \quad (5)$$

This equality follows from the fact that $\sum_{\square \in \Delta[i]} n_{\square} = n$ and $h = O(\log \log n)$. Next, we bound the execution time of HUNGARIANSEARCH and AUGMENT procedures. For any cell \square of level $h - 1$, the HUNGARIANSEARCH and AUGMENT procedures take $(1/\varepsilon)^{O(d^3)}$ time. The total time taken across all level $h - 1$ cells is $n/\varepsilon^{O(d^3)}$. Next, we bound the running time for cell at levels less than $h - 1$.

► **Lemma 8.** *For $i < h$, the number of free vertices after processing level i cells is $O\left(\frac{W}{\mu_i}\right)$.*

Therefore, there are $O\left(\frac{W}{\mu_{i+1}}\right)$ executions of HUNGARIANSEARCH on level i cells. The time taken by all HUNGARIANSEARCH executions that return a NULL is at most

$$\left(\frac{W}{\mu_{i+1}}\right) \times O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^2 \log n) = O(W \varepsilon^{-O(d)} \log^3 n \log^2 \log n).$$

Otherwise, the HUNGARIANSEARCH procedure returns a minimum net-cost augmenting path P and the AUGMENT procedure augments the matching along P . The time taken by each such execution of HUNGARIANSEARCH and AUGMENT procedure is

$$\begin{aligned} & O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^2 \log n) + \sum_{j=i+1}^h k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n \\ &= O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^4 \log n) + \sum_{j=i+1}^h (k_j - 1) \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n. \end{aligned} \quad (6)$$

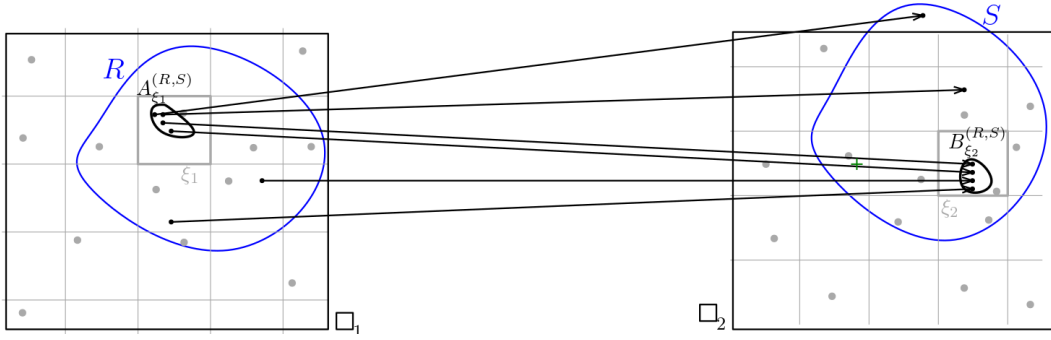
The equality follows from the fact that $\mu_{i+1} > \mu_{j+1}$. While processing $\Delta[i]$, $O(W/\mu_{i+1})$ augmenting paths are found (see Lemma 8), so the first term in the RHS of (6) is $O(W \varepsilon^{-O(d)} \log^3 n \log^5 \log n)$ over all augmenting paths. Next, we bound the second term over all augmenting paths.

Any augmenting path P has at least $k_j - 1$ edges of level at most $j - 1$. Furthermore, for any $j' \geq j$, every level $j - 1$ edge on P will contribute at most two new cells to $k_{j'}$. Suppose γ_j is the number of level $j - 1$ edges across all augmenting paths. The second term of the RHS of (6), when summed across all augmenting paths, can be written as $O(\sum_{j=1}^h \gamma_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$. Lemma 9 (See Appendix B for a proof) establish that $\gamma_j = O\left(\frac{W \log n}{\mu_{j+1}}\right)$.

► **Lemma 9.** *For any $1 \leq j \leq h - 1$, $\gamma_j = O(W \log n / \mu_{j+1})$.*

Therefore, the second term of the RHS of (6) over all augmenting paths is $O(W \varepsilon^{-O(d)} \log^4 n \log^4 \log n)$ and the total running time is $O(W(\varepsilon^{-O(d^3)} + \varepsilon^{-O(d)} \log^4 n \log^4 \log n))$. Since $W = \Theta(n/\varepsilon)$ with probability at least $1/2$, we get the following:

► **Lemma 10.** *Let $A, B \in \mathbb{R}^d$ be two point sets of size n each and a parameter $0 < \varepsilon \leq 1$ that satisfy (P1) – (P3). With probability at least $1/2$, an ε -approximate matching of A, B can be computed in time $O(n(\varepsilon^{-O(d^3)} + \varepsilon^{-O(d)} \log^4 n \log^4 \log n))$.*



■ **Figure 2** Illustrates two boundary clusters $A_{\xi_1}^{(R,S)}$ and $B_{\xi_2}^{(R,S)}$. \square_1 and \square_2 are siblings. Grey subdivision represents the subcells and matching edges are shown as solid black.

5 Algorithm Details

In this section, we present the details of the BUILD, HUNGARIANSEARCH and AUGMENT procedures for some level i . For any cell, we cluster the points inside the cell into $O(\mu_{i+1}^{1/4} \varepsilon^{-d} \log n \log \log n)$ clusters (Section 5.1). We use these clusters to compress the residual graph (Section 5.2) and feasibility conditions (Section 5.3). Next, we describe an efficient implementation of the BUILD, HUNGARIANSEARCH, and AUGMENT procedures using the compressed graph and feasibility conditions (Section 5.4). Our compressed graph is different from [16, 14] as it does not recursively expand an augmenting path in this compressed graph to an admissible augmenting path with respect to M . Instead, our algorithm may modify the matching M to another matching M' of the same cost and size and returns an admissible path with respect to M' .

We begin by introducing a few notation that will assist in describing the procedures. For any cell \square and $\xi \in \mathcal{S}_\square$, let $K \subseteq \mathcal{C}[\square]$ be the subset of children of \square that are contained inside ξ . Let $D(\xi) = \bigcup_{\square'' \in K} \mathcal{S}_{\square''}$ be the set of subcells of the children cells of \square that lie inside ξ . For any cell \square with $\text{lev}(\square) < h - 1$ and for any $j < \text{lev}(\square)$, let \square^j be the ancestor of \square at level j . Consider any $\square' \in \mathcal{C}[\square]$ and a subcell $\xi \in \mathcal{S}_{\square'}$ of \square' . Then, it can be shown that each level- j edge $(a, b) \in A_{\square^j} \times B_{\square^j}$ with one endpoint in ξ is covered by at least one WSPD pair from a subset $\mathcal{W}_\xi^j \subseteq \mathcal{W}_{\square^j}$, with $|\mathcal{W}_\xi^j| = O(\varepsilon^{-1} \log n)$.

► **Lemma 11.** *For any cell \square with $\text{lev}(\square) < h - 1$, for any $j < \text{lev}(\square)$, and for any subcell $\xi \in \mathcal{S}_\square$, $|\mathcal{W}_\xi^j| = O(\varepsilon^{-1} \log n)$. Furthermore, for any non-empty subcell $\xi' \in D(\xi)$, $\mathcal{W}_{\xi'}^j = \mathcal{W}_\xi^j$.*

5.1 Clustering points

For any cell \square of level $k \in [i, h - 1]$, we partition $A_\square \cup B_\square$ into a set of clusters denoted by V_\square . For a subcell ξ of a child \square' of \square (i.e., $\xi \in \mathcal{S}_{\square'}$), we partition $A_\xi \cup B_\xi$ into three types of clusters. We create one *free cluster* A_ξ^F (resp. B_ξ^F) for all free points of A_ξ (resp. B_ξ) and one *internal cluster* A_ξ^I (resp. B_ξ^I) for all points $a \in A_\xi$ (resp. $b \in B_\xi$) such that $m(a) \in \square'$ (resp. $m(b) \in \square'$). Additionally, we create *boundary clusters* as follows: For any level $j \in [i, k]$, recall that \mathcal{W}_ξ^j is the set of WSPD pairs that cover all level- j edges with at least one endpoint in ξ . For every pair $(R, S) \in \mathcal{W}_\xi^j$, we create one cluster $A_\xi^{(R,S)}$ (resp. $B_\xi^{(R,S)}$) of A (resp. B) that contains all points $a \in A_\xi$ (resp. $b \in B_\xi$) whose matching edge $(a, m(a))$ (resp. $(m(b), b)$) is a level- j edge that is covered by the well-separated pair (R, S) (See Figure 2). For each level j , there are $O(\varepsilon^{-d} \log n)$ WSPD pairs in \mathcal{W}_ξ^j and there are $O(\log \log n)$ levels. Therefore, there are $O(\varepsilon^{-d} \log n \log \log n)$ many clusters of this type.

For any cell \square of level $k \in [i, h-1)$, and for any child $\square' \in \mathcal{C}[\square]$, every subcell $\xi \in \mathcal{S}_{\square'}$ is formed by combining the children of \square' . For any subcell $\xi \in \mathcal{S}_{\square}$, the free and internal clusters of ξ are formed as in Equation (7).

$$\begin{aligned} A_{\xi}^F &= \bigcup_{\xi' \in D(\xi)} A_{\xi'}^F, & B_{\xi}^F &= \bigcup_{\xi' \in D(\xi)} B_{\xi'}^F. \\ A_{\xi}^I &= \bigcup_{\xi' \in D(\xi)} \{(A_{\xi'}^I \cup \bigcup_{(R,S) \in \mathcal{W}_{\xi'}^{k+1}} A_{\xi'}^{(R,S)})\}, & B_{\xi}^I &= \bigcup_{\xi' \in D(\xi)} \{(B_{\xi'}^I \cup \bigcup_{(R,S) \in \mathcal{W}_{\xi'}^{k+1}} B_{\xi'}^{(R,S)})\}. \end{aligned} \quad (7)$$

For any $i \leq j \leq k$ and any $(R, S) \in \mathcal{W}_{\xi}^j$, the boundary cluster of ξ corresponding to (R, S) is formed as follows.

$$A_{\xi}^{(R,S)} = \bigcup_{\xi' \in D(\xi)} A_{\xi'}^{(R,S)}, \quad B_{\xi}^{(R,S)} = \bigcup_{\xi' \in D(\xi)} B_{\xi'}^{(R,S)}. \quad (8)$$

Based on these relations, we extend the definition of $D(\cdot)$ for any cluster X to denote the clusters that combine to form X as $D(X)$. We refer to each cluster $X' \in D(X)$ as a *child-cluster* of X .

If \square is a level $h-1$ node, for every child $\square' \in \mathcal{C}[\square]$, we add $A_{\square'} \cup B_{\square'}$ to V_{\square} and appropriately classify them as free, internal, or boundary cluster depending on whether they are free, matched inside \square' , or outside \square' .

We present an upper bound on the number of subcells of the children of any cell in the following lemma. See Appendix B for a proof.

► **Lemma 12.** *For any cell \square at level $i < h-1$, $\sum_{\square' \in \mathcal{C}[\square]} |\mathcal{S}_{\square'}| = O(\mu_{i+1}^{1/4})$.*

For each cell \square at level $i < h-1$ and each child $\square' \in \mathcal{C}[\square]$, any subcell $\xi \in \mathcal{S}_{\square'}$ contributes $O(\varepsilon^{-d} \log n \log \log n)$ clusters to V_{\square} . Therefore, by Lemma 12,

► **Corollary 13.** *For any cell \square at level $i < h-1$, $|V_{\square}| = O(\mu_{i+1}^{1/4} \varepsilon^{-d} \log n \log \log n)$.*

5.2 Compressed residual graph

At every non-leaf node \square in the tree T , we create a *compressed residual graph* \mathcal{AG}_{\square} of \mathcal{G}_M^{\square} with V_{\square} being its vertex set. For any non-leaf node \square , the vertex set of \mathcal{AG}_{\square} consists of one vertex for each cluster in V_{\square} . For any cell \square and its child $\square' \in \mathcal{C}[\square]$, we use $V_{\square}(\square')$ to denote the clusters of V_{\square} that are inside \square' . Next, we define E_{\square} , the set of edges of \mathcal{AG}_{\square} , and a weight $\Phi(X, Y)$ for every edge $(X, Y) \in E_{\square}$.

If $\text{lev}(\square) = h-1$, we simply set the edges of \mathcal{AG}_{\square} to be the edges of \mathcal{G}_M^{\square} and use its net-cost as the weight, i.e., for any edge (u, v) in \mathcal{G}_M^{\square} , $\Phi(u, v) = \phi(u, v)$. If $\text{lev}(\square) < h-1$, then we define *internal* and *bridge* edges between vertices of V_{\square} as follows:

Bridge edges: For any two children $\square_1 \neq \square_2 \in \mathcal{C}[\square]$, let $X_1 \in V_{\square}(\square_1)$ and $X_2 \in V_{\square}(\square_2)$ be two clusters in those children, such that X_1 (resp. X_2) is a cluster of type *A* (resp. *B*). If there is at least one non-matching edge $(b, a) \in X_2 \times X_1$, we add a directed edge from X_2 to X_1 and assign it a weight equal to $\Phi(X_2, X_1) = \phi(b, a)$. We refer to this edge as a *non-matching arc*. If there is a matching edge $(a, b) \in X_1 \times X_2$, we add an edge from X_1 to X_2 and set its cost to be $\Phi(X_1, X_2) = \phi(a, b)$. We call this edge from X_1 to X_2 a *matching arc*.

We classify clusters as entry and exit clusters, and define internal edges from an entry to an exit cluster: The free cluster B_{ξ}^F , the internal cluster A_{ξ}^I , and every boundary cluster $B_{\xi}^{(R,S)}$, for $i \leq j \leq k$ and $(R, S) \in \mathcal{W}_{\xi}^j$, is designated as an *entry cluster*. The free cluster A_{ξ}^F ,

the internal cluster B_ξ^I , and every boundary cluster $A_\xi^{(R,S)}$, for $i \leq j \leq k$ and $(R,S) \in \mathcal{W}_\xi^j$, is designated as an *exit cluster*. For any cell \square and its child $\square' \in \mathcal{C}[\square]$, we use $V_\square^\downarrow(\square')$ and $V_\square^\uparrow(\square')$ to denote the entry and exit clusters of $V_\square(\square')$.

We classify entry and exit clusters in this way for the following reason. Consider any augmenting path P . For any cell \square' , consider any edge (u,v) of P such that (u,v) is in \mathcal{G}_M^\square but the edge preceding (u,v) (resp. succeeding (u,v)) is not. Then u has to be a point in an entry (resp. exit) cluster.

Internal edges: Let \square' be any child of \square . For any pair of clusters $(X_1, X_2) \in V_\square^\downarrow(\square') \times V_\square^\uparrow(\square')$, we create an *internal edge* (X_1, X_2) in \mathcal{AG}_\square . Let $E_\square(\square')$ denote the set of these edges. For any $(X'_1, X'_2) \in V(\square') \times V(\square')$, define $P_{\square'}(X'_1, X'_2)$ to be the minimum-weight path between X'_1 and X'_2 in $\mathcal{AG}_{\square'}$. Define $P(X_1, X_2) = \arg \min_{X'_1 \in D(X_1), X'_2 \in D(X_2)} \Phi(P_{\square'}(X'_1, X'_2))$. We set $\Phi(X_1, X_2)$ to be the weight of the path $P(X_1, X_2)$ in $\mathcal{AG}_{\square'}$.

For consistency, if \square is a cell of level $h-1$, any edge that lies completely inside a child of \square becomes an internal edge and edges that go between two points in two different children is referred to as a bridge edge.

We abuse notation and refer to any directed path P between two free clusters in the compressed residual graph \mathcal{AG}_\square as an *augmenting path*. For efficiency reasons, we only store the internal edges of E_\square . To compute the bridge edges and their costs efficiently, we construct an ε -WSPD as described in Section 2. In the following lemma, we bound the total size of all compressed residual graphs across all cells. See Appendix B for a proof.

► **Lemma 14.** *The total size of all compressed residual graphs across all cells of T is $O(n \log n (\varepsilon^{-1} \log \log n)^{O(d)})$.*

► **Lemma 15.** *For any cell \square and for any augmenting path P from u to v in \mathcal{G}_M^\square , there is an augmenting path P' in \mathcal{AG}_\square that goes from the cluster of u to the cluster of v and $\Phi(P') \leq \phi(P)$.*

5.3 Compressed Feasibility

We use the compressed residual graph to compute an augmenting path. To assist us with the computation of this path, we describe a set of dual weights of V_\square and a set of feasibility conditions for the edges of the compressed graph. Let \square be a cell of level i . For every $X \in V_\square$, we assign a dual weight $y(X)$. We say that a matching and dual weights are *compressed feasible* with respect to \mathcal{AG}_\square if, for any directed edge $(X, Y) \in E_\square$,

$$y(X) - y(Y) \leq \Phi(X, Y). \quad (9)$$

Next, we define slack on any compressed edge (X, Y) to be $s(X, Y) = \Phi(X, Y) - y(X) + y(Y)$. Note that $s(X, Y) \geq 0$ for a compressed feasible matching. We say that an edge (X, Y) is *admissible* if $s(X, Y) = 0$. Similar to (3), one can express the weight of any path P in \mathcal{AG}_\square from X to Y using the slacks on its edges as follows.

$$\Phi(P) = \sum_{(X', Y') \in P} (y(X') - y(Y') + s(X', Y')) = y(X) - y(Y) + \sum_{(X', Y') \in P} s(X', Y'). \quad (10)$$

After any execution of BUILD, HUNGARIANSEARCH, or AUGMENT procedures at \square within our algorithm, in addition to (J1) and (J2), our algorithm also satisfies a third invariant. Let \square' be either \square or any descendant of \square in T .

- (J3) There exists a set of dual weights $y(\cdot)$ on $V_{\square'}$ that satisfy compressed feasibility conditions for edges in $\mathcal{AG}_{\square'}$. In addition, for any $X \in V_{\square'}$, if X is a free cluster of A , then $y(X) = 0$ and if X is a free cluster of B , then $y(X) = \max_{X' \in V_{\square'}} y(X')$. Furthermore, for any cluster $X \in V_{\square'}$,
- (a) if X is an internal entry (resp. exit) cluster, $y(X) \leq \min_{X' \in D(X)} y(X')$ (resp. $y(X) \geq \max_{X' \in D(X)} y(X')$), and
 - (b) if X is a free or a boundary cluster then for every cluster $X' \in D(X)$, $y(X') = y(X)$.

5.4 Details of the Procedures

Assume that the algorithm has executed until level $i + 1$ and (J1)–(J3) hold at the end. We describe the implementation of BUILD, HUNGARIANSEARCH and AUGMENT procedures and show that (J1)–(J3) continue to hold after their executions.

5.4.1 Build procedure

For any cell \square of level i and for every $\square' \in \mathcal{C}[\square]$, we have a set of compressed feasible dual weights on $V_{\square'}$. The BUILD procedure creates a cluster X at \square by simply combining the clusters $D(X)$ of its children and set its dual weight $y(X)$ to $\min_{X' \in D(X)} y(X')$ (resp. $\max_{X' \in D(X)} y(X')$) provided X is an entry (resp. exit) cluster.

In order to compute the weight of any internal edge $(X, Y) \in V_{\square}^{\downarrow}(\square') \times V_{\square}^{\uparrow}(\square')$, we observe that from (10), the minimum-weight path $P(X, Y)$ is also the path with the smallest total slack between any two clusters $X', Y' \in D(X) \times D(Y)$ in $\mathcal{AG}_{\square'}$. For every entry cluster $X \in V_{\square}^{\downarrow}(\square')$, this can be found in a straight-forward way using an execution of Dijkstra's shortest-path algorithm. More specifically, we add a source s to $\mathcal{AG}_{\square'}$, connect s to all $X' \in D(X)$ with an edge of weight of $y(X')$, and replace the weight of every other edge in $\mathcal{AG}_{\square'}$ with its slack. Then, we execute Dijkstra's algorithm in $\mathcal{AG}_{\square'}$ from s to find the distance of each cluster Y' from s , denoted by $\kappa_{Y'}$. For any exit cluster $Y \in V_{\square}^{\uparrow}(\square')$, we set the weight of the edge (X, Y) in \mathcal{AG}_{\square} , to be $\Phi(X, Y) = \min_{Y' \in D(Y)} \{\kappa_{Y'} - y(Y')\}$.

The BUILD procedure does not affect the invariants (J1) and (J2). The following lemma states that the invariant (J3) holds after the execution of BUILD procedure.

► **Lemma 16.** *The dual weights assigned to the clusters of V_{\square} by the BUILD procedure satisfy (J3).*

Efficiency of the Build procedure. To compute the internal edges incident on the entry cluster X , instead of using an $O(|V_{\square'}|^2)$ time Dijkstra's algorithm, as in [14], we use the WSPDs in a straight-forward way to compute the shortest path in $O(|E_{\square'}| \log |E_{\square'}|)$ time. Given that the number of entry clusters in each cell is $\log n (\varepsilon^{-1} \log \log n)^{O(d)}$ and since $|E_{\square'}| = O(\varepsilon^{-O(d)} n_{\square'} \log n \log^{O(d)} \log n)$ (from Lemma 14), the total running time of the BUILD procedure is

$$O \left(\sum_{\square' \in \mathcal{C}[\square]} \sum_{X \in V_{\square'}^{\downarrow}(\square')} \varepsilon^{-O(d)} n_{\square'} \log^2 n \log^{O(d)} \log n \right) = O(n_{\square} \log^3 n_{\square} (\varepsilon^{-1} \log \log n)^{O(d)}).$$

5.4.2 HungarianSearch procedure

Let \square be a cell of level i . The HUNGARIANSEARCH procedure on \square consists of two parts. In the first part, the algorithm modifies the dual weights of V_{\square} and make the edges on the minimum-weight augmenting path in \mathcal{AG}_{\square} admissible (Search step). Then, the algorithm

updates the dual weights of the clusters and points and may modify the matching M to another feasible matching M' with $w(M') = w(M)$. Then, it returns an admissible augmenting path in $\mathcal{G}_{M'}^\square$ with respect to M' (Update step). This path is also the minimum net-cost augmenting path with respect to M' .

Search Step: From (10), the path from a free cluster F' to a free cluster F with the smallest total slack is also the minimum-weight path between F' and F in \mathcal{AG}_\square . To compute this path, we replace the cost of every edge in \mathcal{AG}_\square with its slack and then execute a Dijkstra's algorithm that starts at the free clusters of B . Let P_X be the shortest path from a free cluster of B to any cluster X and κ_X be its cost. Let F be a free cluster of A that has the smallest shortest path. If there are no free clusters of A in \mathcal{AG}_\square , then the HUNGARIANSEARCH returns NULL. Let the path P_F start at some free cluster F' of B . P_F is the minimum-weight augmenting path in \mathcal{AG}_\square and $y(F) = 0$ (from (J3)). Therefore, from (10), the augmenting path P_F has a weight of $\Phi(P_F) = y(F') + \kappa_F$. If $\Phi(P_F) \leq \mu_i$, let $U \subseteq V_\square$ be the subset of clusters whose shortest-path distances from s is less than κ_F . We update the dual weights of any cluster $X \in U$ by setting $y(X) \leftarrow y(X) - \kappa_X + \kappa_F$. If $\Phi(P_F) > \mu_i$, the algorithm sets $\kappa_F = \mu_i - y(F')$, updates the dual weights as described above and then returns NULL. The dual updates to the clusters of V_\square in the search step ensures that the dual weights of free clusters of B do not exceed μ_i .

For any cell \square_1 and its child $\square_2 \in \mathcal{C}[\square_1]$, we say that the dual weights of V_{\square_1} *dominates* the dual weights of V_{\square_2} if for each exit cluster $X \in V_{\square_1}^\uparrow(\square_2)$, $y(X) \geq \max_{X' \in D(X)} y(X')$. During the search step, the dual weight of any cluster $X \in V_\square$ is non-decreasing. Therefore, after the search step, for each child $\square' \in \mathcal{C}[\square]$, the dual weights of V_\square dominates the dual weights of $V_{\square'}$. Furthermore, the updated dual weights are compressed feasible and the edges of P_F is admissible.

► **Lemma 17.** *For any cell \square , after the execution of the search step of the HUNGARIANSEARCH procedure, the updated dual weights of V_\square are compressed feasible and every edge on the minimum-weight path computed by the search step is admissible. Furthermore, for any child $\square' \in \mathcal{C}[\square]$, the dual weights of V_\square dominate the dual weights of $V_{\square'}$.*

After the search step, the updated dual weights of V_\square remain compressed feasible and the edges of P_F are admissible. In the Update Step, we expand the path P_F into an admissible augmenting path in the residual graph. We describe a procedure called SYNC that assists in expanding this path. In particular, consider any admissible edge (X, Y) on P_F where (X, Y) is an internal edge for some child $\square' \in \mathcal{C}[\square]$. The SYNC procedure updates the dual weight of $V_{\square'}$ so that the path $P(X, Y)$ becomes admissible.

Sync Procedure. The SYNC procedure takes any cell \square_1 and its child $\square_2 \in \mathcal{C}[\square_1]$ along with a set of compressed feasible dual weights of V_{\square_1} and V_{\square_2} such that the dual weights of V_{\square_1} dominates the dual weights of V_{\square_2} . This procedure then generates a set of compressed feasible dual weights of V_{\square_2} such that the dual weights of V_{\square_1} and V_{\square_2} satisfy conditions (a) and (b) of (J3). Furthermore, if any internal edge $(X, Y) \in E_{\square_1}(\square_2)$ is admissible, then the path $P(X, Y)$ is also admissible with respect to the dual weights of V_{\square_2} . The description of the SYNC procedure is provided in Appendix C. Let \square_1 be a cell of level j of T . The execution of the SYNC procedure on \square_2 requires an execution of Dijkstra's algorithm on \mathcal{AG}_{\square_2} and takes a total of $O(\mu_{j+1}\varepsilon^{-O(d)} \log^3 n \log^2 \log n)$ time. The following lemma states the important properties of the SYNC procedure.

► **Lemma 18.** *For any cell \square_1 and any child $\square_2 \in \mathcal{C}[\square_1]$, suppose the set of dual weights of V_{\square_1} and V_{\square_2} are compressed feasible and the set of dual weight of $V_{\square_1}(\square_2)$ dominates the set of dual weights of V_{\square_2} . After applying the SYNC procedure on \square_2 , we have that:*

- (i) The updated dual weights on \mathcal{AG}_{\square_2} are compressed feasible,
- (ii) For any cluster $X \in V_{\square_1}(\square_2)$, the dual weight of X and clusters in $D(X)$ satisfies the conditions (a) and (b) of (J3), and,
- (iii) If \square_2 is not a leaf cell, for any $\square_3 \in \mathcal{C}[\square_2]$, the dual weights of $V_{\square_2}(\square_3)$ dominates the set of dual weights of V_{\square_3} .

Using Lemma 18 part (iii), it is clear that one can recursively apply the SYNC procedure on all descendants of a cell \square . The following lemma shows that after the search step, if we apply the SYNC procedure on all descendants of every cell $\square \in \Delta[i]$, the dual weights assigned to all clusters and points satisfy (J1)–(J3).

► **Lemma 19.** *After executing the search step of HUNGARIANSEARCH on \mathcal{AG}_{\square} , for every $\square \in \Delta[i]$, suppose we apply SYNC to \square and all its descendants. The resulting up-to-date dual weights satisfy (J1)–(J3).*

The following lemma helps in converting the minimum-weight path obtained by the search step into an admissible augmenting path.

► **Lemma 20.** *For any admissible internal edge $(X, Y) \in E_{\square}(\square')$, let $X' \in D(X)$ and $Y' \in D(Y)$ be the clusters containing the first and the last vertex of some $P(X, Y)$. Then, after calling the SYNC procedure on \square' , the path $P(X, Y)$ is admissible, $y(X') = y(X)$, and $y(Y') = y(Y)$.*

Using the SYNC procedure, the Update step converts the augmenting path P_F returned by the Search step to an admissible augmenting path \tilde{P}_F in the residual graph. In this process, the Update step might change the matching M to another matching M' with the same weight and size. We describe the Update step as a recursive procedure that initially takes P_F as the input.

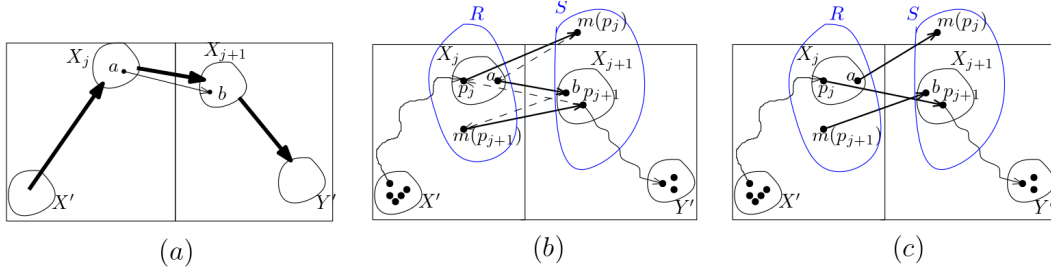
Update step: For any cell \square' , the Update step takes any admissible path $P = \langle X = X_1, X_2, \dots, X_m = Y \rangle$ in $\mathcal{AG}_{\square'}$ as input and returns an admissible alternating path from a point p to p' in $\mathcal{G}_M^{\square'}$ with the property that $y(p) = y(X)$ and $y(p') = y(Y)$.

If \square' is a cell of level $h - 1$, then P is also an admissible path in $\mathcal{G}_M^{\square'}$ and the procedure returns this path. Otherwise, let $k = \text{lev}(\square') < h - 1$ be the level of \square' . Let \mathcal{I} denote the set of all internal edges on the path P . Note that \mathcal{I} is a set of vertex-disjoint edges. Let \mathcal{B} be the set of all clusters X_t on P that do not participate in any edge of \mathcal{I} . It is easy to see that \mathcal{B} is a set of boundary clusters.

For any internal edge $(X_j, X_{j+1}) \in \mathcal{I}$, let $\square_j \in \mathcal{C}[\square']$ such that $(X_j, X_{j+1}) \in E_{\square'}(\square_j)$. We execute the SYNC procedure on \square_j . Since (X_j, X_{j+1}) is an admissible edge, the path $P(X_j, X_{j+1})$ is admissible with $y(X'_j) = y(X_j)$ and $y(X'_{j+1}) = y(X_{j+1})$ (From Lemma 20). We recursively apply the Update step on $P(X_j, X_{j+1})$.

Assume that for each internal edge $(X_j, X_{j+1}) \in \mathcal{I}$, this recursive call has returned an admissible path Π_j in the residual graph from a point $p_j \in X_j$ to a point $p_{j+1} \in X_{j+1}$ with $y(p_j) = y(X_j)$ and $y(p_{j+1}) = y(X_{j+1})$. We select the point p_j for the cluster X_j and the point p_{j+1} for the cluster X_{j+1} . For any boundary cluster $X_t \in \mathcal{B}$, we select an arbitrary point p_t . Let Z be the set of all ancestors of p_t in T of level greater than k . First, we iteratively apply SYNC on every cell in Z in increasing order of their level. After all executions of the SYNC procedure, from Lemma 18 (ii), $y(p_t) = y(X_t)$. Thus, from every cluster X_j on P , we have selected one point p_j with $y(p_j) = y(X_j)$.

Next, we construct the admissible alternating path \tilde{P} corresponding to the path P as follows. For every internal edge $(X_j, X_{j+1}) \in \mathcal{I}$, we replace (X_j, X_{j+1}) with the path Π_j . For every bridge edge (X_j, X_{j+1}) , if (X_j, X_{j+1}) is a non-matching arc, we simply add an edge from p_j to p_{j+1} to the path. If (X_j, X_{j+1}) is a matching arc, then both X_j and X_{j+1}



■ **Figure 3** (a) Compact minimum-weight path P in \mathcal{AG}_{\square} . (b) and (c) show how the matching M is modified to a matching M' to obtain an augmenting path.

are boundary clusters. While there may not be a matching edge between p_j and p_{j+1} , we know that there is some matching edge (a, b) such that $a \in X_j$ and $b \in X_{j+1}$. Let Z (resp. Z') be the set of all ancestors of a (resp. b) of level greater than k . We apply the SYNC procedure on the cells in Z (resp. Z') in increasing order of their level. Then, we modify our matching M to M' as follows: Add matching edges (p_j, p_{j+1}) , $(a, m(p_j))$, and $(b, m(p_{j+1}))$ to the matching and remove the edges (a, b) , $(p_j, m(p_j))$, and $(p_{j+1}, m(p_{j+1}))$ from the matching (See Figure 3). The new matching continues to be feasible since the dual weights of a (resp. b) and p_j (resp. p_{j+1}) are identical. This is because X_j (resp. X_{j+1}) is a boundary cluster and therefore, from Lemma 18 (ii), the application of the SYNC procedure on the ancestors of p_j (resp. p_{j+1}) and a (resp. b) will make $y(a) = y(p_j)$ (resp. $y(b) = y(p_{j+1})$). Note that, for every internal edge (X_j, X_{j+1}) , the path Π_j consists of only admissible edges. Furthermore, for every bridge edge (X_j, X_{j+1}) , the edge (p_j, p_{j+1}) added to the path is admissible. This follows from the fact that $y(p_j) = y(X_j)$, $y(p_{j+1}) = y(X_{j+1})$, and (X_j, X_{j+1}) is admissible. Finally, the dual weight of the first (resp. last) point p_1 (resp. p_m) is equal to $y(X)$ (resp. $y(Y)$) as desired. This completes the description of the Update step. Let \tilde{P}_F be the admissible augmenting path in \mathcal{G}_{\square} returned by the Update step with P_F as input.

► **Lemma 21.** *The matching M can be modified to another matching M' so that, $w(M) = w(M')$, $M', y(\cdot)$ is feasible and the compressed residual graph at each node remains unchanged. Furthermore, there is an admissible augmenting path in the residual graph $\mathcal{G}_{M'}$.*

Recollect that, we only require the existence of a set of dual weights that satisfy the conditions in (J1)–(J3). For efficiency reasons, the Update step does not maintain the up-to-date dual weights explicitly. Instead, it computes the up-to-date dual weights for all cells \square' whose $V_{\square'}$ or $M_{\square'}$ may change after augmenting M along \tilde{P}_F . For every other cell \square'' , from Lemma 22 and Lemma 19, we can always retrieve the up-to-date dual weights for these cells satisfying (J1)–(J3) by recursively applying SYNC on \square'' and all its descendants.

► **Lemma 22.** *During the execution of our algorithm, consider a sequence of consecutive applications of the SYNC procedure on a cell \square . If M_{\square} and the clusters in V_{\square} remain unchanged, then, this sequence of SYNC executions can be replaced with the last one while producing the same set of dual weights of V_{\square} .*

Efficiency of the HungarianSearch procedure. The search step requires execution of a single Dijkstra's algorithm on \mathcal{AG}_{\square} which takes $O(\mu_{i+1}\varepsilon^{-O(d)}\log^3 n \log^2 \log n)$ time. Applying SYNC procedure for a cell \square' of level j requires an execution of Dijkstra's algorithm on $\mathcal{AG}_{\square'}$, which takes $O(\mu_{j+1}\varepsilon^{-O(d)}\log^3 n \log^2 \log n)$ time. Recall that for any level j of T and an

augmenting path \tilde{P}_F on the residual graph, k_j denotes the number of level j cells containing at least one point of \tilde{P}_F . In the Update step, For each level $j > i$, we execute the SYNC procedure on the cell of level j containing at least one point of \tilde{P}_F . Furthermore, during the Update step, for each bridge matching arc (X_j, X_{j+1}) the algorithm may also apply the SYNC procedure on an additional $O(\log \log n)$ cells. This is done before the algorithm modifies the matching. These executions of the SYNC procedure can be charged to the $O(\log \log n)$ SYNC procedures executed for the ancestors of points $p_j \in X_j$ and $p_{j+1} \in X_{j+1}$. Therefore, there are at most $O(k_j \log \log n)$ executions of the SYNC procedure during the execution of the Update step. The execution time of HUNGARIANSEARCH procedure, therefore, is $O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^2 \log n + \sum_{j=i+1}^{h-1} k_j \mu_{j+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$.

5.4.3 Augment procedure

Given an augmenting path $P = \langle b_0, a_0, b_1, \dots, b_k, a_k \rangle$ with respect to the matching M , the AUGMENT procedure will simply update $M \leftarrow M \oplus P$. After augmentation, any edge (a_i, b_i) is a matching edge and any edge (b_{i+1}, a_i) is a non-matching edge (Note that the direction of the edges are reversed after the augmentation). For a new matching edge (a_i, b_i) after an augmentation, suppose \square_i is the least common ancestor of a_i and b_i . Let $X, Y \in V_{\square_i}$ be the pair of boundary clusters such that $(a_i, b_i) \in X \times Y$. If there exists another matching edge $(a'_i, b'_i) \in (X \times Y) \setminus P$, then a_i and b_i inherit their dual weights, i.e., $y(a_i) \leftarrow y(a'_i)$ and $y(b_i) \leftarrow y(b'_i)$. Otherwise, their dual weight remains unchanged. This completes the description of the AUGMENT procedure. For every new matching edge (a_i, b_i) , the procedure may inherit the dual weights from another matching edge (a'_i, b'_i) that did not participate in P . Since (J2) holds prior to augmentation, $y(a'_i)$ and $y(b'_i)$ satisfy (J2). Therefore, post augmentation, $y(a_i)$ and $y(b_i)$ also satisfy (J2).

The following lemma shows that the dual weights of the points after the AUGMENT procedure remains feasible and (J1) holds.

► **Lemma 23.** *After augmenting the matching and updating the dual weights by the AUGMENT procedure, the dual weights of the points are feasible with respect to the new matching.*

The vertex and the edge sets of the compressed residual graph change after augmentation. The AUGMENT procedure creates the new clusters at all ancestors of every point in P in a straight-forward way. In a bottom-up fashion, for any cluster $X \in V_{\square}$, if X is an exit (resp. entry) cluster, it assigns $\max_{X' \in D(X)} y(X')$ (resp. $\min_{X' \in D(X)} y(X')$) as $y(X)$. For each edge on P , the procedure will update the bridge edges in \mathcal{AG}_{\square} in a straight-forward way. The following lemma shows that the updated dual weights are compressed feasible with respect to \mathcal{AG}_{\square} .

► **Lemma 24.** *After augmenting the matching, the new set of clusters and their dual weights $y(\cdot)$ satisfy (J3).*

Next, for any $\square' \in C[\square]$, in order to update the weights on the internal edges $E_{\square}(\square')$ in \mathcal{AG}_{\square} , we apply the BUILD procedure. From Corollary 13, if \square is a cell of level i , then $|V_{\square}| = O(\mu_{i+1}^{1/4} \varepsilon^{-d} \log n \log \log n)$. Since there at most $O(|V_{\square}|)$ entry clusters in V_{\square} and the BUILD procedure executes that many Dijkstra's algorithm to construct the internal edges, the total time taken is bounded by $O(\mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n)$.

The AUGMENT procedure executes BUILD procedure on all ancestors of any vertex of P in a bottom-up fashion. Therefore, the total time taken is $O(\sum_{j=1}^i (k_j \mu_{i+1} \varepsilon^{-O(d)} \log^3 n \log^3 \log n))$.

References

- 1 Pankaj Agarwal and Kasturi Varadarajan. A near-linear constant-factor approximation for Euclidean bipartite matching? In *Proceedings of the twentieth annual symposium on Computational geometry*, page 247, 2004.
- 2 Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. Deterministic, near-linear ε -approximation algorithm for geometric bipartite matching. *arXiv preprint arXiv:2204.03875*, 2022.
- 3 A. Andoni, K. D. Ba, P. Indyk, and D. P. Woodruff. Efficient sketches for earth-mover distance, with applications. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 324–330, 2009.
- 4 Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proc. 34th International Conference on Machine Learning*, pages 214–223, 2017.
- 5 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, January 1995.
- 6 Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 380–388, 2002.
- 7 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *In Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.
- 8 Kyle Fox and Jiashuai Lu. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. In *Proc. 36th Annual Symposium on Computational Geometry*, pages 45:1–45:18, 2020.
- 9 H. N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18:1013–1036, 1989.
- 10 Sariel Har-Peled. Geometric approximation algorithms, 2011.
- 11 Piotr Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *SODA 2007*, page 4, 2007.
- 12 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2495–2504, 2017.
- 13 Harold Kuhn. Variants of the hungarian method for assignment problems. *Naval Research Logistics*, 3(4):253–258, 1956.
- 14 Nathaniel Lahn and Sharath Raghvendra. An $O(n^{5/4})$ time ε -approximation algorithm for RMS matching in a plane. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pages 869–888, 2021.
- 15 Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255, 2004.
- 16 Sharath Raghvendra and Pankaj K. Agarwal. A near-linear time ε -approximation algorithm for geometric bipartite matching. *Journal of the ACM*, 67(3):1–19, June 2020.
- 17 Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover’s distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.
- 18 R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *29th International Symposium on Computational Geometry*, pages 9–16, 2013. doi:10.1145/2462356.2480283.
- 19 R. Sharathkumar and P. K. Agarwal. Algorithms for transportation problem in geometric settings. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 306–317, 2012.
- 20 Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics*, 34(4):66, 2015.

- 21 Jan van den Brand, Danupon Nanongkai Yin-Tat Lee, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. *IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 919–930, 2020.

A Preprocessing Step

Similar to some of the earlier algorithms [16, 14], we perform the following preprocessing step that makes the input “well-conditioned” at a slight increase in the cost of the optimal matching. Using a quad-tree based greedy algorithm [6], we compute a $c_1 \log n$ -approximate matching M_0 of A and B , in $O(n \log n)$ time, for some constant $c_1 \geq 0$ [7]. Let $w_0 = c(M_0)$. Then $\frac{w_0}{c_1 \log n} \leq c(M^*) \leq w_0$.

For any integer $i \in [0, \log(c_1 \log n)]$, define $\beta_i = w_0/2^i$; there is an i such that $\beta_i \leq c(M^*) \leq \beta_{i-1}$. Set $t(n) = c_0 n(\varepsilon^{-O(d^3)} + \log^3 n(\varepsilon^{-1} \log \log n)^{O(d)} + \varepsilon^{-O(d)} \log^4 n \log \log^4 n)$ for some sufficiently large constant c_0 . We run the algorithm described in Section 4 for at most $t(n)$ steps on each choice of β_i . In the i -th iteration, either the algorithm returns a perfect matching of A and B or terminates without computing a perfect matching. Among the perfect matchings computed by the algorithm, we return the one with the smallest cost. Theorem 10 ensures that if $\beta_i \leq c(M^*) \leq \beta_{i-1}$ then with probability at least $1/2$, the algorithm returns an ε -approximate matching within $t(n)$ time. Now forward, we assume that we have computed a value $\beta > 0$ such that $c(M^*) \leq \beta \leq 2c(M^*)$. As in [16, 14], by scaling $A \cup B$ and snapping points to an integer grid, we can assume A and B satisfy the following conditions: (P1) All input points have integer coordinates bounded by $n^{O(1)}$. (P2) No integer grid point contains points of both A and B . (P3) $c(M^*) \in \left[\frac{3\sqrt{dn}}{\varepsilon}, \frac{9\sqrt{dn}}{\varepsilon} \right]$. We compute an $(\varepsilon/2)$ -approximate matching of A and B satisfying (P1) – (P3) in $t(n)$ time.

B Missing Proofs

Proof of Lemma 3. If the edge (a, b) intersects the boundary of a cell at level $i + 1$, then, $\text{lev}(a, b) \leq i$. Therefore, $\Pr[\text{lev}(a, b) = i] \leq \frac{\sqrt{d}\|a-b\|}{\ell_{i+1}}$. As a result,

$$\mathbb{E}[\delta_{ab}] \leq \sum_{i=0}^{h-1} \Pr[\text{lev}(a, b) = i] \cdot \mu_{i+1} \leq \sum_{i=0}^{h-1} \frac{\sqrt{d}\|a-b\|}{\ell_{i+1}} \cdot \frac{\varepsilon \ell_{i+1}}{c_2 \sqrt{d}(h-i)^2} = \frac{\varepsilon}{c_2} \|a-b\| \sum_{i=0}^{h-1} \frac{1}{(h-i)^2}.$$

$\sum_{i=1}^{\infty} \frac{1}{i^2} = \zeta(2) = \frac{\pi^2}{6}$, where $\zeta(\cdot)$ is the *Riemann zeta function*. Therefore, $\mathbb{E}[\delta_{ab}] \leq \frac{\pi^2}{6c_2} \varepsilon \|a-b\|$.

Proof of Lemma 9. To prove this lemma, first, we bound the total additive error across all augmenting paths computed during the execution of our algorithm.

► **Lemma 25.** $\sum_{P_i \in \mathbb{P}} \sum_{(u,v) \in P_i \cap M_i} \delta_{uv} = O(W \log n)$.

Every level $j - 1$ edge in γ_j appears as a matching or a non-matching edge. Furthermore, any matching edge will first appear as a non-matching edge in an augmenting path and carry an additive error of μ_{j+1} . Therefore, we charge at most two level $j - 1$ edges in γ_j to the additive error of any non-matching edge. So, the total additive error on all non-matching edges of γ_j is at least $\gamma_j \mu_{j+1}/2$. Combining with Lemma 25, we get $\gamma_j = O(W \log n / \mu_{j+1})$.

Proof of Lemma 12. By construction, the number of subcells of a cell \square' is bounded by the number of children of \square' ; i.e, $|\mathbf{S}_{\square'}| \leq |\mathbf{C}[\square']|$. Furthermore, for any cell \square' at level j , $|\mathbf{C}[\square']| \leq (\frac{\ell_j}{\ell_{j+1}})^d = \ell_j^{d(1-\alpha)}$. As a result,

$$\sum_{\square' \in \mathbf{C}[\square]} |\mathbf{S}_{\square'}| \leq \sum_{\square' \in \mathbf{C}[\square]} |\mathbf{C}[\square']| \leq \ell_i^{d(1-\alpha)} \ell_{i+1}^{d(1-\alpha)} \leq \ell_i^{\frac{2d}{8d+2}}. \quad (11)$$

From Lemma 2, $\mu_{i+1} \geq \ell_{i+2}$. Furthermore, from the construction of the tree and since $d \geq 2$, $\ell_{i+2} = \ell_i^{\left(\frac{8d+1}{8d+2}\right)^2} \geq \ell_i^{\frac{8d}{8d+2}}$. Plugging this in (11), we get $\sum_{\square' \in \mathbf{C}[\square]} |\mathbf{S}_{\square'}| = O(\mu_{i+1}^{1/4})$.

Proof of Lemma 14. By construction, for any cell \square and any child $\square' \in \mathbf{C}[\square]$ and for each pair of clusters $(X, Y) \in V_{\square}^{\downarrow}(\square') \times V_{\square}^{\uparrow}(\square')$, there exists at most one internal edge in E_{\square} . Therefore, each cluster $X \in V_{\square}(\square')$ has degree at most $|V_{\square}(\square')| = O(|\mathbf{S}_{\square'}| \varepsilon^{-d} \log n \log \log n) = O(\varepsilon^{-O(d)} \log n \log^{O(d)} \log n)$, where the last equality is resulted since $|\mathbf{S}_{\square'}| = O(\varepsilon^{-d} \log^{2d} \log n)$. Summing over all clusters in V_{\square} ,

$$|E_{\square}| = O(\varepsilon^{-O(d)} |V_{\square}| \log n \log^{O(d)} \log n).$$

Summing across all cells in T , we get

$$\sum_{\square \in T} |E_{\square}| = \sum_{\square \in T} |V_{\square}| \times O(\varepsilon^{-O(d)} \log n \log^{O(d)} \log n). \quad (12)$$

Since each point participates in a single cluster per level and $O(\log \log n)$ clusters overall, we get $\sum_{\square \in T} |V_{\square}|$ by $O(n \log \log n)$. Plugging this in (12), we can conclude that the total size of the compressed residual graph across all cells is $O(n \log n (\varepsilon^{-1} \log \log n)^{O(d)})$.

C Details of the Sync procedure

For any entry cluster $X \in V_{\square}^{\downarrow}$ and any cluster $X' \in D(X)$, the dual weight $y(X')$ needs to be no less than the updated $y(X)$ (In the case of free or boundary clusters, it should be equal). We define $\text{inc}(X')$ to be the value by which $y(X')$ should be increased, i.e., $\text{inc}(X') = y(X) - y(X')$. Note that $\text{inc}(X')$ can be negative. Let $\rho_X = \max_{X' \in D(X)} \text{inc}(X')$ and let $\rho = \max\{0, \max_{X \in V_{\square}^{\downarrow}(\square')} \rho_X\}$. The value ρ corresponds to the largest increase in dual weights we desire across all child-clusters in each entry cluster. So, for any cluster $X \in V_{\square}^{\downarrow}(\square')$ and $X' \in D(X)$, $-\infty < \text{inc}(X') \leq \rho$.


Let \mathcal{AG}' be an augmented compressed residual network that is created by adding a vertex s to $\mathcal{AG}_{\square'}$ and connecting s to every $X' \in D(X)$ for any entry cluster $X \in V_{\square}^{\downarrow}(\square')$. We set the weight of (s, X') to be $\rho - \text{inc}(X')$. Since $\text{inc}(X') \leq \rho$, the weight on the edge will be non-negative. For every other edge (U, V) , we set its weight to be the slack $s(U, V)$. We then execute Dijkstra's algorithm on \mathcal{AG}' from the source s . For any cluster V , let κ_V denote the weight of the shortest-path distance from s to V . The dual updates are done in an identical fashion to the HUNGARIANSEARCH. Let U denote the set of all clusters $V \in V_{\square'}$ with $\kappa_V < \rho$. For any $V \in U$, we update the dual weight $y(V) \leftarrow y(V) - \kappa_V + \rho$.

After updating these dual weights, for every boundary and free exit cluster $X \in V_{\square}^{\uparrow}(\square')$ and any $X' \in D(X)$, we set $y(X') \leftarrow y(X)$. This step will not decrease the dual weight of any cluster. This completes the description of SYNC procedure.

On the Visibility Graphs of Pseudo-Polygons: Recognition and Reconstruction

Safwa Ameer ✉

Department of Computer Science, The University of Texas at San Antonio, TX, USA

Matt Gibson-Lopez ✉ 

Department of Computer Science, The University of Texas at San Antonio, TX, USA

Erik Krohn ✉

Department of Computer Science, The University of Wisconsin – Oshkosh, WI, USA

Qing Wang ✉

Department of Computer Science, University of Tennessee at Martin, TN, USA

Abstract

We give polynomial-time algorithms that solve the pseudo-polygon visibility graph recognition and reconstruction problems. Our algorithms are based on a new characterization of the visibility graphs of pseudo-polygons.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Pseudo-Polygons, Visibility Graph Recognition, Visibility Graph Reconstruction

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.7

Funding Supported by the National Science Foundation under Grant No. 1733874.

1 Introduction

Geometric covering problems have been a focus of research for decades. Here we are given a set of points P and a set S where each $s \in S$ can cover some subsets of P . The subset of P is generally induced by some geometric object. For example, P might be a set of points in the plane, and s consists of the points contained within some disk in the plane. For most variants, the problem is NP-hard and can easily be reduced to an instance of the combinatorial set cover problem which has a polynomial-time $O(\log n)$ -approximation algorithm, which is the best possible approximation under standard complexity assumptions [5]. The main question therefore is to determine for which variants of geometric set cover we can obtain polynomial-time approximation algorithms with approximation ratio $o(\log n)$, as any such algorithm must exploit the geometry of the problem to achieve the result. This area has been studied extensively, see for example [2, 14, 1], and much progress has been made utilizing algorithms that are based on solving the standard linear programming relaxation.

Unfortunately this technique has severe limitations for some variants of geometric set cover, and new ideas are needed to make progress on these variants. In particular, the techniques are lacking when the points P we wish to cover is a simple polygon, and we wish to place the smallest number of points in P that collectively “see” the polygon. This problem is classically referred to as the *art gallery problem* as an art gallery can be modeled as a polygon and the points placed by an algorithm represent cameras that can “guard” the art gallery. This has been one of the most well-known problems in computational geometry for many years, yet still to this date the best polynomial-time approximation algorithms for this problem have approximation ratios that are $\omega(1)$. The key issue is a fundamental lack of understanding of the combinatorial structure of visibility inside simple polygons. It seems that in order to develop powerful approximation algorithms for this problem, the community first needs to better understand the underlying structure of such visibility.



© Safwa Ameer, Matt Gibson-Lopez, Erik Krohn, and Qing Wang;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 7; pp. 7:1–7:13

Leibniz International Proceedings in Informatics



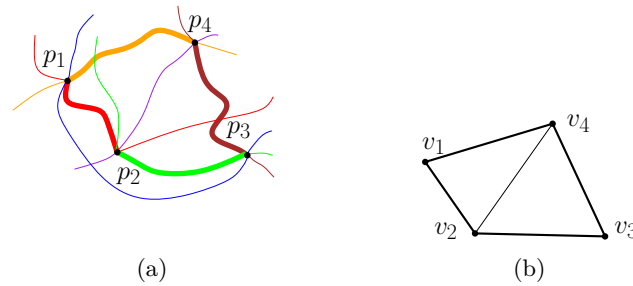
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Visibility Graphs. A very closely related issue which has received a lot of attention in the community is the *visibility graph* (VG) of a simple polygon. Given a simple polygon P , the VG $G = (V, E)$ of P has the following structure. For each vertex $p \in P$, there is a vertex in V , and there is an edge connecting two vertices in G if and only if the corresponding vertices in P “see” each other (i.e., the line segment connecting the points does not go outside the polygon). The VG of a simple polygon must contain a Hamiltonian cycle that corresponds with the boundary of the P , and it generally is assumed that the input G comes with a labeled Hamiltonian cycle. Three major open problems regarding VGs of simple polygons are the VG characterization problem, the VG recognition problem, and the VG reconstruction problem. The *VG characterization* problem seeks to define a set of properties that all VGs satisfy. The *VG recognition* problem is the following. Given a graph G , determine if there exists a simple polygon P such that G is the VG of P in polynomial time. The *VG reconstruction* problem seeks to construct a simple polygon P such that a given VG G is the VG of P .

The problems of characterizing and recognizing the VGs of simple polygons have had partial results given dating back to over 25 years ago [6] and remain open to this day with only a few special cases being solved. Characterization and recognition results have been given in the special cases of “spiral polygons” [4] and “tower polygons” [3]. There have been several results [7, 4, 12] that collectively have led to four necessary conditions (NCs) that a simple polygon VG must satisfy. That is, if the graph G does not satisfy all four of the conditions then we know that G is not the VG for *any* simple polygon, and moreover it can be determined if a graph G satisfies all of the NCs in polynomial time. Streinu, however, has given an example of graph that satisfies all of the NCs but is not a VG for any simple polygon [13], implying that the set of conditions is not sufficient and therefore a strengthening of the NCs is needed. Unfortunately it is not even known if simple polygon VG recognition is in NP. See [8] for a nice survey on these problems and other related visibility problems.

Pseudo-polygons. Given the difficulty of understanding simple polygon VGs, O’Rourke and Streinu [10] considered the VGs for a special case of polygons called *pseudo-polygons* which we will now define. An arrangement of *pseudo-lines* \mathcal{L} is a collection of simple curves, each of which separates the plane, such that each pair of pseudo-lines of \mathcal{L} intersects at exactly one point, where they cross. Let $P = \{p_0, p_2, \dots, p_{n-1}\}$ be a set of points in \mathbb{R}^2 , and let \mathcal{L} be an arrangement of $\binom{n}{2}$ pseudo-lines such that every pair of points p_i and p_j lie on exactly one pseudo-line in \mathcal{L} , and each pseudo-line in \mathcal{L} contains exactly two points of P . The pair (P, \mathcal{L}) is called a *pseudo-configuration of points* (pcp) in general position.

Intuitively a pseudo-polygon is determined similarly to a standard Euclidean simple polygon except using pseudo-lines instead of straight line segments. Let $L_{i,j}$ denote the pseudo-line through the points p_i and p_j . We view $L_{i,j}$ as having three different components. The subsegment of $L_{i,j}$ connecting p_i and p_j is called the *segment*, and we denote it $p_i p_j$. Removing $p_i p_j$ from $L_{i,j}$ leaves two disjoint *rays*. Let $r_{i,j}$ denote the ray starting from p_i and moving away from p_j , and we let $r_{j,i}$ denote the ray starting at p_j and moving away from p_i . Consider the pseudo-line $L_{i,i+1}$ in a pcp (indices taken modulo n and are increasing in counterclockwise order throughout the paper). We let e_i denote the segment of this line. A *pseudo-polygon* is obtained by taking the segments e_i for $i \in \{0, \dots, n-1\}$ if (1) the intersection of e_i and e_{i+1} is only the point p_{i+1} for all i , and (2) for any i and j such that $j > i+1$, the segments e_i and e_j do not intersect. We call the segments e_i the *boundary edges*. A pseudo-polygon separates the plane into two regions: “inside” the pseudo-polygon and “outside” the pseudo-polygon, and any two points p_i and p_j see each other if the segment $p_i p_j$ does not go outside of the pseudo-polygon. See Fig. 1 for an illustration. Pseudo-polygons



■ **Figure 1** (a) A pcp and pseudo-polygon. (b) The corresponding VG.

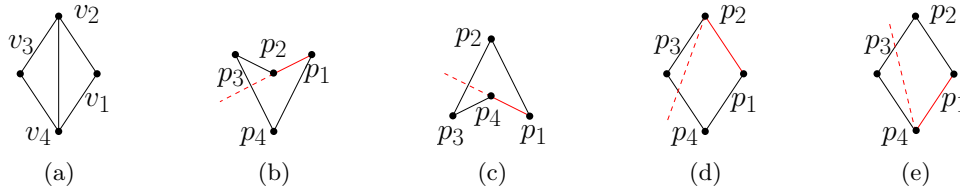
can be viewed as a combinatorial abstraction of simple polygons. Note that every simple polygon is a pseudo-polygon (simply allow each $L_{i,j}$ to be the straight line through p_i and p_j), and Streinu showed that there are pseudo-polygons that cannot be “stretched” into a simple polygon [13].

O’Rourke and Streinu [10] give a characterization of *vertex-edge* VGs of pseudo-polygons. In this setting, for any vertex v we are told which *edges* v sees rather than which *vertices* it sees. Unfortunately, O’Rourke and Streinu showed that vertex-edge VGs encode more information about a pseudo-polygon than a regular VG [11]. Gibson, Krohn, and Wang [9] gave a characterization of the VGs of pseudo-polygons. Unfortunately this characterization did not directly lead to a polynomial-time recognition or reconstruction algorithm.

Our Results. In this paper, we give results for the VGs of both pseudo-polygons and simple polygons. First, we settle the remaining two open questions for the VGs of pseudo-polygons: recognition and reconstruction. First, we present a polynomial-time algorithm that can decide if a given graph G (with a labeled Hamiltonian cycle) is the VG for some pseudo-polygon, settling the recognition problem for pseudo-polygons. To obtain the result, we give a slightly different characterization of pseudo-polygon VGs than the one given in [9]. We then show that we can extend the recognition algorithm to obtain a polynomial-time reconstruction algorithm for pseudo-polygons. Our algorithm computes a vertex-edge VG that can then be reconstructed into a pseudo-polygon using the technique described in [10].

2 Preliminaries

We begin with some definitions that were relied upon heavily in the characterization of [9] that will be used in this paper as well. Note that the visibility graph G of a pseudo-polygon P must contain a Hamiltonian cycle because each p_i must see p_{i-1} and p_{i+1} . Since determining if a graph contains a Hamiltonian cycle is NP-hard, previous research has assumed that G does have such a cycle C and the vertices are labeled in counterclockwise order according to this cycle. So now suppose we are given an arbitrary graph $G = (V, E)$ with the vertices labeled p_0 to p_{n-1} such that G contains a Hamiltonian cycle $C = (p_0, p_2, \dots, p_{n-1})$ in order according to their indices. We are interested in determining if G is the visibility graph for some pseudo-polygon P where C corresponds with the boundary of P . For any two vertices p_i and p_j , we let $\partial(p_i, p_j)$ denote the vertices and boundary edges encountered when walking counterclockwise around C from p_i to p_j (inclusive). For any edge $\{p_i, p_j\}$ in G , we say that $\{p_i, p_j\}$ is a *visible pair*, as their points in P must see one another. If $\{p_i, p_j\}$ is not an edge in G , then we call (p_i, p_j) and (p_j, p_i) *invisible pairs*. Note that visible pairs are unordered, and invisible pairs are ordered (for reasons described below).

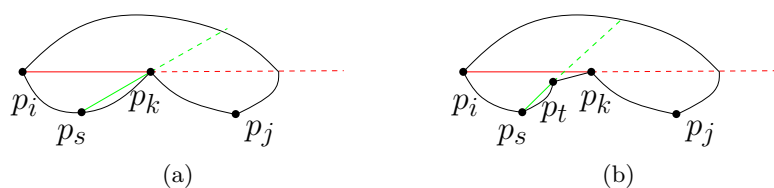


■ **Figure 2** (a) A visibility graph G . (b) A simple polygon using p_2 to block p_1 and p_3 . (c) A simple polygon using p_4 to block p_1 and p_3 . (d) A pseudo-polygon using p_2 to block p_1 and p_3 . Note that if the segment p_1p_3 does not exit the polygon then it would have to intersect $L_{1,2}$ at least twice (once at p_1 and once in the dashed ray $r_{2,1}$). (e) A pseudo-polygon using p_4 to block p_1 and p_3 .

Consider any invisible pair (p_i, p_j) . If G is the visibility graph for a pseudo-polygon P , the segment $p_i p_j$ must exit P . For example, suppose we want to construct a polygon P such that the graph in Fig. 2 (a) is the visibility graph of P . Note that p_1 should not see p_3 , and thus if there exists such a polygon, it must satisfy that $p_1 p_3$ exits the polygon. In the case of a simple polygon, we view this process as placing the vertices of P in convex position and then contorting the boundary of P to block p_1 from seeing p_3 . We can choose p_2 or p_4 to block p_1 from seeing p_3 (see (b) and (c)). Note that as in Fig. 2 (b) when using $p_2 \in \partial(p_1, p_3)$ as the blocker in a simple polygon, the line segment $p_1 p_2$ does not go outside P and the ray $r_{2,1}$ first exits P through a boundary edge in $\partial(p_3, p_1)$. Similarly as in Fig. 2 (c) when using $p_4 \in \partial(p_3, p_1)$ as the blocker, the line segment $p_1 p_4$ does not go outside of the polygon and the ray $r_{4,1}$ first exits the polygon through a boundary edge in $\partial(p_1, p_3)$. The situation is similar in the case of pseudo-polygons, but since we do not have to use straight lines to determine visibility, instead of bending the boundary of P to block the invisible pair we can instead bend the pseudo-line. See Fig. 2 (d) and (e). Note that the combinatorial structure of the pseudo-line shown in part (d) (resp. part (e)) is the same as the straight line in part (b) (resp. in part (c)). The following definition plays an important role in our characterization. Consider a pseudo-polygon P , and let p_i and p_j be two vertices of P that do not see each other. We say a vertex $p_k \in \partial(p_i, p_j)$ of P is a *designated blocker* for the invisible pair (p_i, p_j) if p_i sees p_k (i.e. the segment $p_i p_k$ is inside the polygon) and the ray $r_{k,i}$ first exits the polygon through an edge in $\partial(p_j, p_i)$. The definition for $p_k \in \partial(p_j, p_i)$ to be a designated blocker for (p_i, p_j) is defined similarly. Intuitively, a designated blocker is a canonical vertex that prevents the points in an invisible pair from seeing each other.

The key structural lemma proved in [9] that led to their characterization was to show that every invisible pair (p_i, p_j) in a pseudo-polygon must have exactly one designated blocker. Moreover, the designated blocker must be one of at most two *candidate blockers*. There is at most one candidate blocker in $\partial(p_i, p_j)$ and there is at most one candidate blocker in $\partial(p_j, p_i)$. We will define the candidate blocker in $\partial(p_i, p_j)$, and the other case is handled symmetrically. Starting from p_j , walk clockwise towards p_i until we reach the first point p_k such that $\{p_i, p_k\}$ is a visible pair (clearly there must be such a point since $\{p_i, p_{i+1}\}$ is a visible pair). We say that p_k is a *candidate blocker* for (p_i, p_j) if there are no visible pairs $\{p_s, p_t\}$ such that $p_s \in \partial(p_i, p_{k-1})$ and $p_t \in \partial(p_{k+1}, p_j)$. If there is such a visible pair $\{p_s, p_t\}$, then there is no candidate blocker (and therefore no designated blocker) for (p_i, p_j) in $\partial(p_i, p_j)$. Note that a vertex may be a candidate blocker for (p_i, p_j) but not for (p_j, p_i) , and we view invisible pairs as ordered pairs for this reason. The formal statement of the lemma proved in [9] is as follows.

► **Lemma 1.** *For any invisible pair (p_i, p_j) in a pseudo-polygon P , there is exactly one designated blocker p_k . Moreover, p_k is a candidate blocker for the invisible pair (p_i, p_j) in the visibility graph of P .*



■ **Figure 3** (a) If p_k is the designated blocker for (p_i, p_j) and p_s sees p_k then p_k is the designated blocker for (p_s, p_j) . (b) If p_s does not see p_k , and p_t is the designated blocker for (p_s, p_k) then p_t is also the designated blocker for (p_s, p_j) .

2.1 The Characterization of [9]

We now state the characterization of pseudo-polygon VGs given in [9]. The main idea is that if G is the VG of some pseudo-polygon P , then each invisible pair must be able to be assigned exactly one of its candidate blockers to serve as the designated blocker in P because of Lemma 1. However, we cannot simply arbitrarily pick a candidate blocker to serve as the designated blocker, as some choices may cause pseudo-lines to violate the pseudo-line properties. That is, some assignments may force a pair of pseudo-lines to intersect more than once and/or they may intersect at a vertex but not cross. The proof for each of the following assignment properties (APs) from the characterization in [9] showed that if the AP was violated then we would violate such a pseudo-line property. They later proved that if one can assign a designated blocker to each invisible pair that satisfies all of these properties, then G is in fact the VG for some pseudo-polygon (i.e., the properties are necessary and sufficient). We remark that the first four properties will also be used in the characterization in this paper, but the fifth property will be replaced with a new property to obtain a different characterization that will better fit within the framework of our reconstruction algorithm.

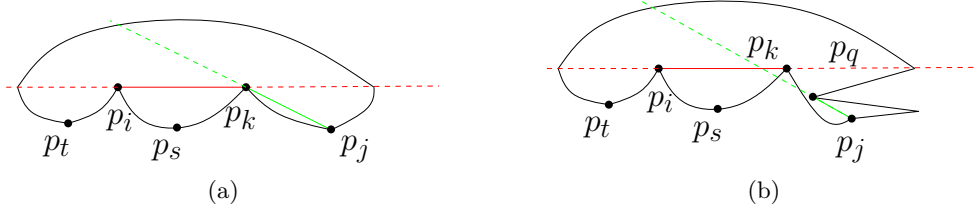
Let (p_i, p_j) be an invisible pair, and let p_k be the candidate blocker assigned to it. The first AP uses the definition of pseudo-lines and designated blockers to provide additional constraints on p_i and p_k . Note that while the condition is stated for $p_k \in \partial(p_i, p_j)$, a symmetric condition for when $p_k \in \partial(p_j, p_i)$ clearly holds.

► **Assignment Property 1.** *If $p_k \in \partial(p_i, p_j)$ is the candidate blocker assigned to invisible pair (p_i, p_j) then both of the following must be satisfied: (1) p_k is assigned to the invisible pair (p_i, p_t) for every $p_t \in \partial(p_{k+1}, p_j)$ and (2) if (p_k, p_j) is an invisible pair then p_i is not the candidate blocker assigned to it.*

Again let p_k be the candidate blocker assigned to an invisible pair (p_i, p_j) such that $p_k \in \partial(p_i, p_j)$. Since p_k is a candidate blocker, we have that (p_s, p_j) is an invisible pair for every $p_s \in \partial(p_i, p_{k-1})$. The next AP is a constraint on the location of designated blockers for (p_s, p_j) . In particular, if $\{p_s, p_k\}$ is a visible pair, then p_k must be the designated blocker for (p_s, p_j) . See Fig. 3 (a). If (p_s, p_k) is an invisible pair, then it must be assigned a designated blocker p_t . In this case, p_t must also be the designated blocker for (p_s, p_j) . See Fig. 3 (b).

► **Assignment Property 2.** *Let (p_i, p_j) denote an invisible pair, and suppose p_k is the candidate blocker assigned to this invisible pair. Without loss of generality, suppose $p_k \in \partial(p_i, p_j)$, and let p_s be any vertex in $\partial(p_i, p_{k-1})$. Then exactly one of the following two cases holds: (1) $\{p_s, p_k\}$ is a visible pair, and the candidate blocker assigned to the invisible pair (p_s, p_j) is p_k , or (2) (p_s, p_k) is an invisible pair. If the candidate blocker assigned to (p_s, p_k) is p_t , then (p_s, p_j) is assigned the candidate blocker p_t .*

The next AP is somewhat similar to AP 2, except instead of introducing constraints on



■ **Figure 4** (a) If p_k is the designated blocker for (p_i, p_j) and p_j sees p_k then p_k is the designated blocker for (p_j, p_s) , (p_j, p_i) , and (p_j, p_t) . (b) If p_j does not see p_k , and p_q is the designated blocker for (p_j, p_k) then p_q is the designated blocker for (p_j, p_s) , (p_j, p_i) , and (p_j, p_t) . Moreover, (p_i, p_q) is an invisible pair and p_k is its designated blocker.

the designated blockers for (p_s, p_j) , it introduces constraints on the designated blockers for (p_j, p_s) (where the order is reversed). Similar to the previous case, if p_j sees p_k then p_k must block p_j from seeing every $p_s \in \partial(p_i, p_{k-1})$, but we can also see that p_k must block p_j from any point p_t such that p_i is the designated blocker for (p_k, p_t) . See Fig. 4 (a). If p_j does not see p_k , then there must be a designated blocker p_q for (p_j, p_k) . See Fig. 4 (b). We show that in this case, p_q must be the designated blocker for all (p_j, p_s) and (p_j, p_t) . Also, (p_i, p_q) must be an invisible pair with designated blocker p_k .

► **Assignment Property 3.** Let (p_i, p_j) denote an invisible pair, and suppose p_k is the candidate blocker assigned to this invisible pair. Without loss of generality, suppose $p_k \in \partial(p_i, p_j)$. Then exactly one of the following two cases holds:

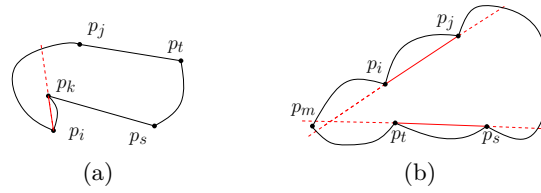
1. (a) $\{p_j, p_k\}$ is a visible pair. (b) For all $p_s \in \partial(p_i, p_{k-1})$, the candidate blocker assigned to the invisible pair (p_j, p_s) is p_k . (c) If p_t is such that p_i is the candidate blocker assigned to the invisible pair (p_k, p_t) , then (p_j, p_t) is an invisible pair and is assigned the candidate blocker p_k .
2. (a) (p_j, p_k) is an invisible pair. Let p_q denote the candidate blocker assigned to (p_j, p_k) . (b) (p_i, p_q) is an invisible pair, and p_k is the candidate blocker assigned to it. (c) For all $p_s \in \partial(p_i, p_k)$, the candidate blocker assigned to the invisible pair (p_j, p_s) is p_q . (d) If p_t is such that p_i is the candidate blocker assigned to the invisible pair (p_k, p_t) , then (p_j, p_t) is an invisible pair and is assigned the candidate blocker p_q .

Suppose p_k is a candidate blocker for an invisible pair (p_i, p_j) (or (p_j, p_i)), and suppose without loss of generality that $p_i \in \partial(p_j, p_k)$. If p_k is also a candidate blocker for an invisible pair (p_s, p_t) such that $p_s, p_t \in \partial(p_k, p_j)$ then we say that the two invisible pairs are a *separable invisible pair*. We have the following condition which is the same as Necessary Condition 3 for simple polygons in [8]. See Fig. 5 (a).

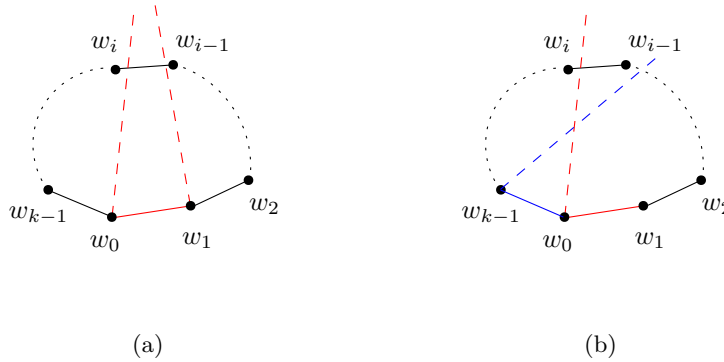
► **Assignment Property 4.** Suppose (p_i, p_j) and (p_s, p_t) are a separable invisible pair with respect to a candidate blocker p_k . If p_k is assigned to (p_i, p_j) then it is not assigned to (p_s, p_t) .

We now give the final AP. Let p_j, p_i, p_t , and p_s be four vertices of G in “counter-clockwise order” around the Hamiltonian cycle C . We say that they are $\{p_i, p_t\}$ -pinched if there is a $p_m \in \partial(p_i, p_t)$ such that p_i is the designated blocker for the invisible pair (p_j, p_m) and p_t is the designated blocker for the invisible pair (p_s, p_m) . See Fig. 5 (b). The notion of $\{p_j, p_s\}$ -pinched is defined symmetrically.

► **Assignment Property 5.** Let p_i, p_j, p_s , and p_t be four vertices of G in counter-clockwise order around the Hamiltonian cycle C that are $\{p_i, p_t\}$ -pinched. Then they are not $\{p_j, p_s\}$ -pinched.



■ **Figure 5** (a) If p_k blocks one invisible pair of a separable invisible pair then it cannot block the other one as well. (b) p_i, p_j, p_s , and p_t are $\{p_i, p_t\}$ -pinched. If p_j blocks p_i from seeing some point, then p_s cannot also block p_t from seeing that point.



■ **Figure 6** Illustrations for the proof of NC 1. (a) If w_0 is the designated blocker for (w_1, w_i) and w_1 is the designated blocker for (w_0, w_{i-1}) then $L_{0,1}$ will intersect the segment $w_{i-1}w_i$ twice. (b) If w_0 is the designated blocker for (w_1, w_i) and w_{k-1} is the designated blocker for (w_0, w_{i-1}) then $L_{0,1}$ intersects $L_{0,k-1}$ twice (once at w_0 and again in the dashed rays).

3 A New Characterization of Pseudo-Polygon VGs

In this section, we prove a different characterization of the VGs of pseudo-polygons. This characterization proves a property that must be satisfied by all VGs of pseudo-polygons. We then show that if G satisfies this property, then AP 5 is not needed. That is, if G satisfies this property and we can find an assignment of candidate blockers to invisible pairs that satisfies APs 1-4, then AP 5 must also be satisfied.

The property we prove is similar to the NC given by Ghosh [6] for simple polygon VGs; however, the proof uses geometric arguments that do not apply to general pseudo-polygons and therefore a new proof is needed. To state and prove the property, we first need some definitions. Suppose w_0, w_1, \dots, w_{k-1} form a cycle in G such that the vertices w_0, w_1, \dots, w_{k-1} follow the order in the Hamiltonian cycle C . Then, we say that w_0, w_1, \dots, w_{k-1} are an *ordered cycle*. Note that the Hamiltonian cycle C is an ordered cycle of all n vertices in G . An edge in G connecting two non-adjacent vertices of an ordered cycle is called a *chord*.

► **Necessary Condition 1.** *If G is the VG of a pseudo-polygon then any ordered cycle in G of length at least 4 must have at least one chord.*

Proof. Suppose G has an ordered cycle O of length $k \geq 4$, and let w_0, w_1, \dots, w_{k-1} denote the vertices around O in counterclockwise order, and for the sake of contradiction assume that O does not have any chords. This implies that any $w_i \in O$ sees no other vertices in O other than w_{i-1} and w_{i+1} (indices taken modulo k). Moreover, since $k \geq 4$, this implies that

there is at least one vertex on O that w_i does not see. Let w_j be such a vertex. Note that any candidate blocker for (w_i, w_j) must be a vertex that is on O , because any vertex v not on O is in $\partial(w_a, w_{a+1})$ for some a and w_a sees w_{a+1} since they are consecutive points on O .

So now consider w_1 . There is at least one vertex on O that w_1 does not see, and it must be that either w_0 and w_2 is the designated blocker for any such invisible pairs. Without loss of generality, assume that w_0 is a designated blocker for (w_1, w_j) for at least one $w_j \in O$. Walking counterclockwise around O starting at w_1 , let w_i be the first point we encounter such that w_0 is the designated blocker for (w_1, w_i) . Now note that $i - 1 \geq 2$, in particular i cannot be 2 because w_1 sees w_2 . Therefore (w_0, w_{i-1}) is an invisible pair and either w_{k-1} or w_1 must be its designated blocker. But if w_1 is its designated blocker, then $L_{0,1}$ will intersect $w_{i-1}w_i$ twice (see Figure 6 (a)). If w_{k-1} is its designated blocker then $L_{k-1,0}$ will intersect $L_{0,1}$ twice (see Figure 6 (b)). ◀

Ghosh [7] showed it can be determined if G satisfies this property in $O(n^2)$ time. Now suppose that G does indeed satisfy this property. We will show then that it suffices to pick an assignment of candidate blockers to invisible pairs that satisfies only APs 1-4. In order to prove this, we prove two lemmas which will be used to prove the new characterization.

► **Lemma 2.** *Let p_i, p_j, p_k be three vertices of a pseudo-polygon VG in counterclockwise order. If p_j is a candidate blocker for invisible pair (p_i, p_k) , then $(p_k, p_{k'})$ is an invisible pair for any $p_{k'} \in \partial(p_k, p_i)$ such that p_i is a candidate blocker for $(p_j, p_{k'})$.*

Proof. We will show that if p_k sees $p_{k'}$ then G violates NC 1. Suppose p_k sees $p_{k'}$. If p_k sees p_j and $p_{k'}$ sees p_i then $p_i, p_j, p_k, p_{k'}$ is an ordered cycle of length four with zero chords, a violation of NC 1. So now suppose that p_k does not see p_j . We will pick a “chain” C_1 of vertices (c_1, c_2, \dots, c_m) in $\partial(p_j, p_k)$ such that: 1) the order of the vertices in C_1 is in clockwise order, and 2) a vertex c_i only sees vertices c_{i-1} and c_{i+1} (if they exist) in C_1 . Initially we let $c_1 = p_k$. Now suppose c_i is the last vertex in C_1 we have found and we wish to find c_{i+1} . We start at p_j and walk counterclockwise until we find the first vertex that sees c_i (it may be p_j itself). If this vertex sees $p_{k'}$, then we throw out the current chain restart a new chain with this vertex as c_1 , and otherwise we let this vertex be c_{i+1} in the current chain. We continue this process until p_j is added to C_1 . Note that p_j cannot be c_1 since it does not see $p_{k'}$, and therefore the length of C_1 is at least 2.

We repeat a symmetric process to obtain a chain $C_2 = (c'_1, c'_2, \dots, c'_{m'})$ in $\partial(p_{k'}, p_i)$, except we compute C_2 with respect to the vertices of C_1 . That is, when we have computed c'_i and wish to compute c'_{i+1} , we consider if it sees any of the vertices in C_1 . If it doesn't, then we add it to C_2 as vertex c'_{i+1} . If it does see some vertex of C_1 , then we update *both* chains. We restart C_2 with this vertex as c'_1 , and we will remove a “prefix” of C_1 depending on what c'_1 sees. Let z be the maximum integer such that c'_1 sees c_z of C_1 . Then we remove all vertices from C_1 with index less than z . Note that c'_1 cannot see p_j or else p_i would not be a candidate blocker for $(p_j, p_{k'})$, and therefore C_1 still has length at least 2. We continue this until p_i gets added to C_2 . Note that when p_i is added to C_2 , C_1 is not reduced because if p_i saw any vertex in C_1 other than p_j then p_j would not be a candidate blocker for (p_i, p_k) . Therefore C_1 and C_2 both have length at least 2.

The following is an ordered cycle of length at least four that does not have any chords: $c'_1, c'_2, \dots, c'_{m'}, c_m, c_{m-1}, \dots, c_1$. Indeed we have that the only visible pairs that has one vertex in C_1 and the other in C_2 are $\{c_1, c'_1\}$ and $\{c'_{m'}, c_m\}$ by construction. Note $c'_{m'} = p_i$ and $c_m = p_j$. $\{p_i, p_j\}$ must be a visible pair because they are candidate blockers for each other. And finally there are no chords connecting two vertices of C_1 or two vertices of C_2 by construction of the chains. It follows that if p_k sees $p_{k'}$, then G violates NC 1. ◀

► **Lemma 3.** *Let p_i, p_j, p_s, p_t be four vertices of a pseudo-polygon VG G in counterclockwise order such that $\{p_i, p_j\}$, $\{p_i, p_s\}$, $\{p_j, p_t\}$, and $\{p_s, p_t\}$ are visible pairs. Suppose we have an assignment of candidate blockers to invisible pairs that satisfies AP 1-4. Let B be the set of all vertices in $\partial(p_j, p_s)$ such that for each $p_z \in B$, p_j is the designated blocker for (p_i, p_z) and p_s is the designated blocker for (p_t, p_z) . If $|B| \geq 1$, then there is at least one vertex $p_k \in B$ such that satisfies one property of set 1 and one property of set 2:*

1. a. $\{p_k, p_j\}$ is a visible pair, or
b. (p_k, p_j) is an invisible pair blocked by a designated blocker in $\partial(p_j, p_k)$
2. a. $\{p_k, p_s\}$ is a visible pair, or
b. (p_k, p_s) is an invisible pair blocked by a designated blocker in $\partial(p_k, p_s)$.

Proof. We first show a fact about any point p_z in B that does not satisfy the conditions of the lemma. Suppose for p_z at least one of the two sets of properties has both properties not satisfied. Without loss of generality, assume it is the first set. Then (p_z, p_j) is an invisible pair blocked by a candidate blocker $p_b \in \partial(p_z, p_j)$. We will show that p_b must be in B . To prove this, we first show that p_b must be in $\partial(p_{z+1}, p_{s-1})$. p_z cannot see any point in $\partial(p_i, p_{j-1})$ or else p_j would not be a candidate blocker for (p_i, p_z) . If p_b were in $\partial(p_s, p_i)$, AP 3 case 2 (b) implies that p_j must be the designated blocker for (p_i, p_b) . But if this is true then AP 1 case 1 implies that p_j must be the designated blocker for (p_i, p_s) , but $\{p_i, p_s\}$ is a visible pair. Therefore it must be $p_b \in \partial(p_{z+1}, p_{s-1})$. We now show that $p_b \in B$. That is, it must be that (p_b, p_i) and (p_b, p_t) are both invisible pairs. If $\{p_b, p_t\}$ were a visible pair then p_s would not be a candidate blocker for (p_t, p_z) . Also (p_i, p_b) must be an invisible pair with designated blocker p_j by AP 3 case 2 (b). Therefore p_b is indeed in B .

Now we show how to find a point in B that satisfies the conditions of the lemma. Walk counterclockwise starting from p_j until we find the first vertex that is in B , and let us call this vertex p_{x_1} . If this vertex does not satisfy a property from each set, then it must be that there is a blocker p_{x_2} that blocks p_{x_1} from p_j (it must also block it from p_s by AP 1). From the above analysis $p_{x_2} \in B$, and therefore p_{x_2} must be in $\partial(p_{x_1+1}, p_{s-1})$ since p_{x_1} is the first point encountered in B . Likewise, if p_{x_2} does not satisfy a property from each set, then there must be a blocker $p_{x_3} \in B$ that blocks p_{x_2} from both p_s and p_j . Again we can see that $p_{x_3} \in \partial(p_{x_2+1}, p_{s-1})$. Indeed, it cannot be in $\partial(p_j, p_{x_1-1})$ since none of these points are in B , and it cannot be in $\partial(p_{x_1}, p_{x_2-1})$ as that would contradict that p_{x_2} is a candidate blocker for (p_{x_1}, p_s) . Inductively we repeat this until we find a point that satisfies the condition. This process must terminate since if p_{x_i} does not satisfy a property from each set, it must be blocked by a point in $\partial(p_{x_i+1}, p_{s-1})$. Eventually we will run out of points in B and there will be no more points to be the blocker, and therefore we will find the desired point. ◀

We are now ready to prove our new characterization that removes the need to satisfy AP 5 if G satisfies NC 1.

► **Theorem 4.** *A graph G with a labeled Hamiltonian cycle is the visibility graph of some pseudo-polygon if and only if it satisfies NC 1 and there is an assignment of candidate blockers to the invisible pairs of G that satisfies APs 1-4.*

Proof. Assuming that G satisfies NC 1, we will prove that if an assignment violates AP 5, then it also violates one of the other APs. Assume AP 5 is violated. That is, let p_i, p_j, p_s, p_t be four vertices of G in counterclockwise order around C such that there is a $p_k \in \partial(p_j, p_s)$ and a $p_{k'} \in \partial(p_t, p_i)$ satisfying: 1) (p_i, p_k) is an invisible pair that has been assigned p_j , 2)

(p_t, p_k) is an invisible pair that has been assigned p_s , 3) $(p_j, p_{k'})$ is an invisible pair that has been assigned p_i , and 4) $(p_s, p_{k'})$ is an invisible pair that has been assigned p_t . Here, we pick p_k and $p_{k'}$ to be points that satisfy the conditions of Lemma 3.

Now note that $(p_k, p_{k'})$ must be an invisible pair by Lemma 2, and therefore there must be a candidate blocker assigned to this invisible pair. Since p_j is the candidate blocker assigned to (p_i, p_k) and p_i is the candidate blocker assigned to $(p_j, p_{k'})$, AP 3 implies that the candidate blocker assigned to $(p_k, p_{k'})$ must be in $\partial(p_{k'}, p_k)$ (i.e., the blocker is p_j if p_k sees p_j or whatever point is blocking p_k from p_j which must be in $\partial(p_j, p_k)$ since p_k satisfies Lemma 3). However, we can apply the same argument symmetrically to p_s and p_t to see that $(p_k, p_{k'})$ must be assigned a blocker in $\partial(p_k, p_{k'})$. Therefore no matter which candidate blocker we assign to $(p_k, p_{k'})$, it must be that AP 3 is violated. ◀

4 Recognition and Reconstruction Algorithms

In this section, we give a polynomial-time algorithm to determine whether or not a given graph G with a labeled Hamiltonian cycle C is the VG for some pseudo-polygon. We then extend the algorithm to obtain a polynomial-time algorithm that can reconstruct a pseudo-polygon P such that G is the VG of P if the recognition algorithm returns YES.

Algorithm 1 Recognition Algorithm.

```

1: if  $G$  does not satisfy NC 1 then
2:   Return NO
3: for all invisible pairs  $(p_i, p_j)$  do
4:   Compute their (at most 2) candidate blockers.
5:   Add  $(p_i, p_j)$  to the set of all invisible pairs  $I$ .
6: while there is an invisible pair (IP)  $(p_i, p_j) \in I$  such that  $(p_i, p_j)$  has  $< 2$  remaining
   feasible candidate blockers do
7:   if some IP has 0 remaining candidate blockers then
8:     Return NO
9:   else
10:    if some IP has exactly 1 feasible candidate blocker then
11:      Assign this candidate blocker to the IP.
12:      Remove the IP from  $I$ .
13:      For every other IP in  $I$ , remove any remaining candidate blocker if its selection
        would violate APs 1-4.
14: Return YES

```

The recognition algorithm is stated formally in Algorithm 1. The algorithm itself is fairly simple. We first check that G satisfies NC 1, and if it does not then we return NO. If it satisfies this property, then for each invisible pair, we compute the set of at most two candidate blockers for this invisible pair. If some invisible pair has 0 candidate blockers, we return NO. If some invisible pair has only 1 candidate blocker, then it must be the designated blocker so we assign it to the candidate blocker. We then remove any candidate blocker for any other invisible pair if that candidate blocker would violate one of the APs. We then repeat this until every invisible pair has been assigned a candidate blocker, or until every remaining invisible pair has two candidate blockers remaining. In either case, we return YES.

The algorithm runs in polynomial time. Checking NC 1 can be done in $O(n^2)$ time [7]. There are at most $O(n^2)$ invisible pairs of G . Checking a violation of an AP involves only 2 invisible pairs, and this check can be done in constant time. The algorithm is clearly

correct if we return NO or if we return YES because every invisible pair was assigned a candidate blocker. We must prove that the algorithm is correct when we return YES because every remaining invisible pair has two candidate blockers. We prove this constructively by assigning a candidate blocker to each remaining invisible pair in I . The algorithm for this is formally stated in Algorithm 2.

■ **Algorithm 2** Candidate Blocker Assignment.

-
- 1: Let I denote the invisible pairs of G that are not assigned a candidate blocker by Algorithm 1.
 - 2: Let p_x be any arbitrarily chosen vertex of G .
 - 3: **for all** invisible pairs $(p_i, p_j) \in I$ **do**
 - 4: Walk counterclockwise around the Hamiltonian cycle C starting at p_x . Let p_a denote which vertex of (p_i, p_j) is encountered first and let p_b denote the other vertex.
 - 5: Assign to (p_i, p_j) the candidate blocker in $\partial(p_a, p_b)$.
-

We now prove that the combination of candidate blocker assignments in Algorithm 1 and in Algorithm 2 satisfies APs 1-4, thereby proving the correctness of Algorithm 1.

► **Lemma 5.** *The combination of candidate blocker assignments in Algorithm 1 and in Algorithm 2 assigns a valid candidate blocker to every invisible pair of G .*

Proof. Each of the APs regards the feasibility of a pair of assigned candidate blockers. Since we removed any candidate blocker that would have created a violation with a choice made in Algorithm 1, we only need to consider candidate blockers assigned in Algorithm 2. We go through each AP and show that any assignment we make will not violate the AP, which completes the proof of the lemma.

AP 1. Case 1 states that if $p_k \in \partial(p_i, p_j)$ is the candidate blocker assigned to invisible pair, (p_i, p_j) then it must also be assigned (p_i, p_t) for every $p_t \in \partial(p_{k+1}, p_j)$. Since $p_k \in \partial(p_i, p_j)$, it must be that $p_i = p_a$ and $p_j = p_b$ in Algorithm 2 implying that $p_x \in \partial(p_{j+1}, p_i)$. Then it also must mean that $p_i = p_a$ and $p_t = p_b$ when considering (p_i, p_t) , and therefore p_k will be assigned to (p_i, p_t) . Case 2 states that if (p_k, p_j) is an invisible pair then p_i is not assigned to it, but since $p_x \in \partial(p_{j+1}, p_i)$ it must be that $p_k = p_a$ and $p_j = p_b$ when considering (p_k, p_j) and therefore we will assign to (p_k, p_j) the candidate blocker in $\partial(p_k, p_j)$ which is not p_i .

AP 2. This AP applies when $p_k \in \partial(p_i, p_j)$ is assigned to invisible pair (p_i, p_j) and then considers the invisible pairs (p_s, p_j) for each $p_s \in \partial(p_i, p_{k-1})$. If we choose $p_k \in \partial(p_i, p_j)$, then it must be that $p_x \in \partial(p_{j+1}, p_i)$, and therefore we will assign to (p_s, p_j) the candidate blocker in $\partial(p_s, p_j)$. Case 1 of AP 2 states that if $\{p_s, p_k\}$ is a visible pair, then p_k must be assigned to (p_s, p_j) , and indeed this is what we do if $\{p_s, p_k\}$ is a visible pair because no point in $\partial(p_s, p_{k-1})$ can see any point in $\partial(p_{k+1}, p_j)$ or else p_k would not be a candidate blocker for (p_i, p_j) . Case 2 says that if (p_s, p_k) is an invisible pair, then the candidate blocker assigned to (p_s, p_j) must be the same as the candidate blocker assigned to (p_s, p_k) . But given the location of p_x , it must be that for both invisible pairs we have $p_s = p_a$ in Algorithm 2, and therefore we will assign the same candidate blocker to both invisible pairs.

AP 3. This AP applies when $p_k \in \partial(p_i, p_j)$ is assigned to invisible pair (p_i, p_j) and then considers the invisible pairs (p_j, p_i) as well as (p_j, p_t) for any p_t such that p_i is the candidate blocker assigned to (p_k, p_t) . First we consider (p_j, p_i) . If $p_k \in \partial(p_i, p_j)$ is assigned to invisible pair (p_i, p_j) , then it must be that $p_x \in \partial(p_{j+1}, p_i)$. This implies

that when considering (p_j, p_i) , we will assign the candidate blocker in $\partial(p_i, p_j)$ to (p_j, p_i) . Case 1 states that if $\{p_j, p_k\}$ is a visible pair that this blocker must be p_k , and in fact this the candidate blocker in $\partial(p_i, p_j)$ because no point in $\partial(p_i, p_{k-1})$ can see any point in $\partial(p_{k+1}, p_j)$ or else p_k would not be a candidate blocker for (p_i, p_j) . Case 2 states that if (p_j, p_k) is an invisible pair, then whatever blocker is assigned to (p_j, p_k) must be assigned to (p_j, p_i) , but here we have $p_j = p_b$ in both scenarios, and therefore we will assign the same blocker to both (p_j, p_k) and (p_j, p_i) .

Now consider (p_j, p_t) for any p_t such that p_i is the candidate blocker assigned to (p_k, p_t) . The analysis is very similar to the previous case. If $p_k \in \partial(p_i, p_j)$ is assigned to invisible pair (p_i, p_j) and $p_i \in \partial(p_t, p_k)$ is assigned to invisible pair (p_k, p_t) , then it must be that $p_x \in \partial(p_{j+1}, p_t)$. This implies that $p_t = p_a$ and $p_j = p_b$ when considering (p_j, p_t) . For the same reasons as in Case 1 for AP 3, we will assign the correct candidate blocker to (p_j, p_t) .

AP 4. Here we have (p_i, p_j) and (p_s, p_t) which are a separable invisible pair with respect to candidate blocker p_k . If p_k is assigned to (p_i, p_j) , then that implies that $p_j = p_a$ and $p_i = p_b$ which means that $p_x \in \partial(p_{i+1}, p_j)$. This means that when we consider (p_s, p_t) we will have $p_s = p_a$ and $p_t = p_b$, and therefore we will assign it the candidate blocker in $\partial(p_s, p_t)$ which is not p_k . ◀

This gives us the following theorem.

► **Theorem 6.** *There is a polynomial-time algorithm that can determine whether a given graph G with a labeled Hamiltonian cycle C is the visibility graph for some pseudo-polygon P such that C corresponds with the boundary of P .*

We can then combine our Algorithms 1 and 2 into a reconstruction algorithm by building a vertex-edge VG based on our computed assignment (details in [9]) and then reconstruct the pseudo-polygon from this graph using the technique described in [10].

► **Theorem 7.** *There is a polynomial-time algorithm that can construct a pseudo-polygon P such that a given visibility graph G of a pseudo-polygon with a labeled Hamiltonian cycle C is the visibility graph of P where C corresponds to the boundary of P .*

References

- 1 Greg Aloupis, Jean Cardinal, Sébastien Collette, Stefan Langerman, David Orden, and Pedro Ramos. Decomposition of multiple coverings into more parts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 302–310, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496804>.
- 2 Boris Aronov, Esther Ezra, and Micha Sharir. Small-size epsilon-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, July 2010. doi:10.1137/090762968.
- 3 Seung-Hak Choi, Sung Yong Shin, and Kyung-Yong Chwa. Characterizing and recognizing the visibility graph of a funnel-shaped polygon. *Algorithmica*, 14(1):27–51, 1995. doi:10.1007/BF01300372.
- 4 Hazel Everett and Derek G. Corneil. Negative results on characterizing visibility graphs. *Comput. Geom.*, 5:51–63, 1995. doi:10.1016/0925-7721(95)00021-Z.
- 5 Uriel Feige, Magnús M. Halldórsson, Guy Kortsarz, and Aravind Srinivasan. Approximating the domatic number. *SIAM J. Comput.*, 32(1):172–195, January 2003. doi:10.1137/S0097539700380754.
- 6 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. In *SWAT*, pages 96–104, 1988.

- 7 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997. doi:10.1007/BF02770871.
- 8 Subir Kumar Ghosh and Partha P. Goswami. Unsolved problems in visibility graphs of points, segments, and polygons. *ACM Comput. Surv.*, 46(2):22, 2013. doi:10.1145/2543581.2543589.
- 9 Matt Gibson, Erik Krohn, and Qing Wang. A characterization of visibility graphs for pseudo-polygons. In *ESA*, pages 607–618, 2015.
- 10 Joseph O’Rourke and Ileana Streinu. Vertex-edge pseudo-visibility graphs: Characterization and recognition. In *Symposium on Computational Geometry*, pages 119–128, 1997. doi:10.1145/262839.262915.
- 11 Joseph O’Rourke and Ileana Streinu. The vertex-edge visibility graph of a polygon. *Computational Geometry*, 10(2):105–120, 1998. doi:10.1016/S0925-7721(97)00011-4.
- 12 G. Srinivasaraghavan and Asish Mukhopadhyay. A new necessary condition for the vertex visibility graphs of simple polygons. *Discrete & Computational Geometry*, 12:65–82, 1994. doi:10.1007/BF02574366.
- 13 Ileana Streinu. Non-stretchable pseudo-visibility graphs. *Comput. Geom.*, 31(3):195–206, 2005. doi:10.1016/j.comgeo.2004.12.003.
- 14 Kasturi R. Varadarajan. Epsilon nets and union complexity. In *Symposium on Computational Geometry*, pages 11–16, 2009. doi:10.1145/1542362.1542366.

Recognizing Map Graphs of Bounded Treewidth

Patrizio Angelini ✉ 

Department of Mathematics, Natural, and Applied Sciences, John Cabot University, Rome, Italy

Michael A. Bekos ✉ 

Department of Mathematics, University of Ioannina, Greece

Giordano Da Lozzo ✉ 


Department of Engineering, Roma Tre University, Rome, Italy

Martin Gronemann ✉ 

Algorithms and Complexity Group, Technische Universität Wien, Austria

Fabrizio Montecchiani ✉ 

Department of Engineering, University of Perugia, Italy

Alessandra Tappini ✉ 

Department of Engineering, University of Perugia, Italy

Abstract

A map graph is one admitting a representation in which vertices are nations on a spherical map and edges are shared curve segments or points between nations. We present an explicit fixed-parameter tractable algorithm for recognizing map graphs parameterized by treewidth. The algorithm has time complexity that is linear in the size of the graph and, if the input is a yes-instance, it reports a certificate in the form of a so-called witness. Furthermore, this result is developed within a more general algorithmic framework that allows to test, for any k , if the input graph admits a k -map (where at most k nations meet at a common point) or a hole-free k -map (where each point is covered by at least one nation). We point out that, although bounding the treewidth of the input graph also bounds the size of its largest clique, the latter alone does not seem to be a strong enough structural limitation to obtain an efficient time complexity. In fact, while the largest clique in a k -map graph is $\lfloor 3k/2 \rfloor$, the recognition of k -map graphs is still open for any fixed $k \geq 5$.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Map graphs, Recognition, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.8

Funding *Michael A. Bekos*: Partially supported by DFG grant KA812/18-1.

Giordano Da Lozzo: Partially supported by MSCA-RISE project “CONNECT”, N° 734922, and by MIUR, grant 20174LF3T8 “AHeAD: efficient Algorithms for HARnessing networked Data”.

Fabrizio Montecchiani: Partially supported by MIUR, grant 20174LF3T8 “AHeAD: efficient Algorithms for HARnessing networked Data”, and by Dipartimento di Ingegneria, University of Perugia, grants RICBA20ED and RICBA21LG.

Alessandra Tappini: Partially supported by MIUR, grant 20174LF3T8 “AHeAD: efficient Algorithms for HARnessing networked Data”, and by Dipartimento di Ingegneria, University of Perugia, grants RICBA20ED and RICBA21LG.

Acknowledgements We thank the anonymous reviewers of a previous version of this paper for pointing out that the map recognition problem admits an MSO_2 formulation.



© Patrizio Angelini, Michael A. Bekos, Giordano Da Lozzo, Martin Gronemann, Fabrizio Montecchiani, and Alessandra Tappini;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 8; pp. 8:1–8:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

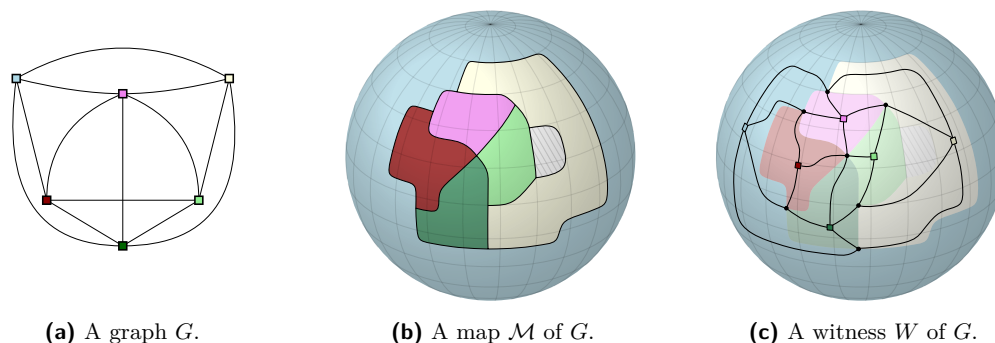
1 Introduction

Planarity is one of the most influential concepts in Graph Theory. Inspired by topological inference problems and by intersection graphs of planar curves, in 1998, Chen, Grigni and Papadimitriou [8] suggested the study of map graphs as a generalized notion of planarity. A *map* of a graph G is a function \mathcal{M} that assigns each vertex v of G to a region $\mathcal{M}(v)$ on the sphere homeomorphic to a closed disk such that no two regions share an interior point, and any two distinct vertices v and w are adjacent in G if and only if the boundaries of $\mathcal{M}(v)$ and $\mathcal{M}(w)$ share at least one point. For each vertex v of G , the region $\mathcal{M}(v)$ is called the *nation* of v . A connected open region of the sphere that is not covered by nations is a *hole*. A graph that admits a map is a *map graph*, whereas a graph that admits a map without holes is a *hole-free map graph*; Figs. 1a and 1b show a graph and a map of it, respectively. Map graphs generalize planar graphs by allowing local non-planarity at points where more than three nations meet. In fact, the planar graphs are exactly those graphs having a map in which at most three nations share a boundary point [8, 19].

Besides their theoretical interest, the study of map graphs is motivated by applications in graph drawing, circuit board design, and topological inference problems [1, 4, 5, 11]. Map graphs are also useful to design parameterized and approximation algorithms for several optimization problems that are NP-hard on general graphs [6, 16, 21, 22, 23].

A natural and central algorithmic question regards the existence of efficient algorithms for recognizing map graphs. Towards an answer to this question, Chen et al. [8, 9] first gave a purely combinatorial characterization of map graphs: A graph is a map graph if and only if it admits a witness, formally defined as follows; see Fig. 1c. A *witness* of a graph $G = (V, E)$ is a bipartite planar graph $W = (V \cup I, A)$ with $A \subseteq V \times I$ and such that $W^2[V] = G$, where the graph $W^2[V]$ is the *half-square* of W , that is, the graph on the vertex set V in which two vertices are adjacent if and only if their distance in W is 2. Here, the vertices in I are meant to represent the adjacencies among nations. Since W can always be chosen to have linear size in the number of vertices of G [9], the problem of recognizing map graphs is in NP. In 1998, Thorup [31] proposed a polynomial-time algorithm to recognize map graphs. However, the extended abstract by Thorup does not contain a complete proof of the result and, to the best of our knowledge, a full version has not appeared yet. Moreover, the proposed algorithm has two drawbacks. First, the time complexity is not specified explicitly (the exponent of the polynomial bounding the time complexity is estimated to be about 120 [10]; see also [5, 28]). Second, it does not report a certificate in the positive case; a natural one would be a witness.

Hence, the problem of finding a simple and efficient recognition algorithm for map graphs remains open. In the last years, several authors focused on graphs admitting restricted types of maps. Aside from the already defined hole-free maps, another notable example consists of the *k-maps*, in which at most k nations meet at a common point; observe that, when $k \geq n - 1$, map graphs and k -map graphs trivially coincide. For instance, Chen studied the density of k -map graphs [7], while in a recent milestone paper on linear layouts Dujmović et al. [20] proved that the queue number of k -map graphs is cubic in k . Note that the algorithm by Thorup [31] cannot be directly used to recognize k -map graphs (unless $k \geq n - 1$). Chen et al. [10] focused on hole-free 4-map graphs and gave a cubic-time recognition algorithm for this graph family. Later, Brandenburg [5] gave a cubic-time recognition algorithm for general (i.e., not necessarily hole-free) 4-map graphs, by exploiting an alternative characterization of these graphs closely related to maximal 1-planarity. Notably, a polynomial-time recognition algorithm for the family of (general or hole-free) k -map graphs with $k > 4$ is still missing. In particular, for $k > 4$, the only result we are aware of is a characterization of 5-map graphs in



■ **Figure 1** (a) A graph G , (b) a map of G - the striped region is a hole, and (c) a witness of G .

terms of forbidden crossing patterns [4]. A different approach for the original problem is the one by Mnich, Rutter, and Schmidt [28], who proposed a linear-time algorithm to recognize the map graphs with an outerplanar witness, which also reports a certificate witness, if any.

We remark that the size of the largest clique in a k -map graph is $\lfloor 3k/2 \rfloor$ (see, e.g., [9]), thus bounding the size of the largest clique does not seem to be a strong enough structural limitation of the input to obtain an efficient time complexity. Despite the notable amount of work, no prior research focuses on further structural parameters of the input graph to design efficient recognition algorithms. In this paper, we address precisely this challenge.

Our contribution. Our main result is a novel algorithmic framework that can be used to recognize map graphs, as well as variants thereof; in particular, hole-free k -map graphs and k -map graphs. Recall that, by setting $k = n - 1$, our algorithm also recognizes (hole-free) map graphs. In fact, we can also compute the minimum value of k within the same asymptotic running time. The proposed algorithm is parameterized by the treewidth [18, 29] of the n -vertex input graph G and its time complexity has a linear dependency in n , while it does not depend on the natural parameter k . Notably, for graphs of bounded treewidth, our algorithm improves over the existing literature [5, 10, 31] in three ways: it solves the problem for *any* fixed k , it can deal with *both* scenarios where holes are or are not allowed in the sought map, and it exhibits an asymptotically *optimal* running time in the input size. The following theorem summarizes our main contribution.

► **Theorem 1.** *Given an n -vertex graph G and a tree-decomposition of G of width t , there is a $O(t^{O(t)} \cdot n)$ -time algorithm that computes the minimum k , if any, such that G admits a (hole-free) k -map. In the positive case, the algorithm returns a certificate in the form of a witness of G within the same time complexity.*

We remark that the problem of recognizing map graphs can be expressed by using MSO_2 logic. Thus the main positive result behind Theorem 1 can be alternatively achieved by Courcelle's theorem [13]. However, with this approach, the dependency of the time complexity on the treewidth is notoriously very high. As a matter of fact, Courcelle's theorem is generally used as a classification tool, while the design of an explicit ad-hoc algorithm remains a challenging and valuable task [15].

To prove Theorem 1, we first solve the decision version of the problem. For a fixed k , we use a dynamic-programming approach, which can deal with different constraints on the desired witness. While we exploit such flexibility to check whether at most k nations intersect at any point and whether holes can be avoided, other constraints could be plugged into the framework such as, for example, the outerplanarity of the witness (as in [28]). In view of this versatility, future applications of our tools may be expected.

Proof strategy. We exploit the characterization in [9] and test for the existence of a suitable witness of the input graph. The crux of our technique is in the computation of suitable records that represent equivalent witnesses and contain only vertices of a tree-decomposition bag. Each such record must carry enough information, in terms of embedding, so to allow testing whether it can be extended with a new vertex or merged with another witness. Moreover, we need to check whether any such witness yields a k -map and, if required, a hole-free one. To deal with the latter property, we provide a strengthening of the characterization in [9], which we believe to be of independent interest, that translates into maintaining suitable counters on the edges of our records. Additional checks on the desired witness can be plugged in the presented algorithmic framework, provided that the records store enough information. One of the main difficulties is hence “sketching” irrelevant parts of the embedded graph without sacrificing too much information. (A similar challenge is faced in the context of different planarity and beyond-planarity problems [17, 25, 27].) Also, when creating such sketches, multiple copies (potentially linearly many) of the same edge may appear, which we need to simplify to keep our records small. The formalization of such records then allows us to exploit a dynamic-programming approach on a tree-decomposition.

Paper structure. Section 2 contains preliminary definitions. Section 3 illustrates basic properties of map graphs that will be used throughout the paper. Section 4 introduces the concept of “sketching” an embedding of a witness, the key ingredient of the algorithmic framework, which we present in Section 5. Section 6 contains open problems raised by our work. The proofs of the statements marked as \star have been omitted.

2 Preliminaries

We only consider finite, undirected, and simple graphs, although some procedures may produce non-simple graphs. In such a case the presence of self-loops or multiple edges will be clearly indicated. Let $G = (V, E)$ be a graph; for a vertex $v \in V$, we denote by $N(v)$ the set of neighbors of v in G , and by $\deg(v)$ the *degree* of v , i.e., the cardinality of $N(v)$.

Embeddings. A *topological embedding* of a graph G on the sphere Σ is a representation of G on Σ in which each vertex of G is associated with a point and each edge of G with a simple arc between its two endpoints in such a way that any two arcs intersect only at common endpoints. A topological embedding of G subdivides the sphere into topologically connected regions, called *faces*. If G is connected, the *boundary* of a face f is a closed walk, that is, a circular list of alternating vertices and edges; otherwise, the boundary of f is a *set* of closed walks. Note that a cut-vertex of G may appear multiple times in any such walk. A topological embedding of G uniquely defines a *rotation system*, that is, a cyclic order of the edges around each vertex. If G is connected, the boundary defining each face can be reconstructed from a rotation system; otherwise, to reconstruct the boundary of every face f , we also need to know which connected components are incident to f . We call the incidence relationship between closed walks of different components and faces the *position system* of G . A *combinatorial embedding* of G is an equivalence class of topological embeddings that define the same rotation and position systems. An *embedded graph* G is a graph along with a combinatorial embedding. A pair of parallel edges e and e' of G with end-vertices v and w is *homotopic* if there is a face of G whose boundary consists of a single closed walk $\langle v, e, w, e' \rangle$.

Tree-decompositions. Let (\mathcal{X}, T) be a pair such that $\mathcal{X} = \{X_1, X_2, \dots, X_\ell\}$ is a collection of subsets of vertices of a graph G , called *bags*, and T is a tree whose nodes are in one-to-one correspondence with the elements of \mathcal{X} . When this creates no ambiguity, X_i will denote both a bag of \mathcal{X} and the node of T whose corresponding bag is X_i . The pair (\mathcal{X}, T) is a *tree-decomposition* of G if: (i) for every edge (u, v) of G , there exists a bag X_i that contains both u and v , and (ii) for every vertex v of G , the set of nodes of T whose bags contain v induces a non-empty (connected) subtree of T .

The *width* of (\mathcal{X}, T) is $\max_{i=1}^{\ell} |X_i| - 1$, while the *treewidth* of G is the minimum width over all tree-decompositions of G . For an n -vertex graph of treewidth t , a tree-decomposition of width t can be found in FPT time [2].

► **Definition 2.** A tree-decomposition (\mathcal{X}, T) of a graph G is called *nice* if T is a rooted tree with the following properties [3].

- (P.1) Every node of T has at most two children.
- (P.2) If a node X_i of T has two children whose bags are X_j and $X_{j'}$, then $X_i = X_j = X_{j'}$. In this case, X_i is a *join bag*.
- (P.3) If a node X_i of T has only one child X_j , then $X_i \neq X_j$ and there exists a vertex $v \in G$ such that either $X_i = X_j \cup \{v\}$ or $X_i \cup \{v\} = X_j$. In the former case X_i is an *introduce bag*, while in the latter case X_i is a *forget bag*.
- (P.4) If a node X_i is a leaf of T , then X_i contains exactly one vertex, and X_i is a *leaf bag*.

Note that, given a tree-decomposition of width t , a nice tree-decomposition can be computed in $O(t \cdot n)$ time (see, e.g., [26]).

3 Basic Properties of Map Graphs and Their Witnesses

The following statements (in a weaker or different form) have already been discussed in the work by Chen et al. [9] and their proofs are omitted here.

Let $G = (V, E)$ be a map graph and let $W = (V \cup I, A)$ be a witness of G , i.e., W is a planar bipartite graph such that $W^2[V] = G$. A vertex $u \in I$ is an *intersection vertex* of W , while a vertex $v \in V$ is a *real vertex* of W . Also, we let $n_V = |V|$, $n_I = |I|$, and $n = n_V + n_I$.

► **Property 3.** A graph is a k -map graph if and only if it admits a witness such that the maximum degree of every intersection vertex is k .

► **Property 4.** A graph G admits a map if and only if each of its biconnected components admits a map. Also, if G admits a hole-free map, then G is biconnected.



■ **Figure 2** (a) Inessential intersection vertices, and (b) a twin-pair.

Let $W = (V \cup I, A)$ be an embedded witness (i.e., with a prescribed combinatorial embedding). An intersection vertex $u \in I$ is *inessential* if $\deg(u) = 2$ and there exists $u' \in I$ such that $N(u) \subset N(u')$; see Fig. 2a. Furthermore, a pair of intersection vertices $u_1, u_2 \in I$ is a *twin-pair* if $N(u_1) = N(u_2) = \{v, w\}$, for some $v, w \in V$, and W contains a face whose

boundary consists of a single closed walk with exactly four edges with end-vertices v, u_1, w, u_2 ; see Fig. 2b. Note that removing an inessential vertex or one vertex of a twin-pair from W does not modify $W^2[V]$.

► **Definition 5.** *An embedded witness of a map graph is compact if it contains neither inessential intersection vertices nor twin-pairs.*

We remark that a compact witness is not necessarily minimal, i.e., it may contain intersection vertices of degree greater than 2 whose removal does not modify its half-square; see also [9]. However, in our setting, removing further information from a witness would have an impact on the proof of Theorem 7 and on the recognition algorithm (Section 5).

The next lemma shows that focusing on compact witnesses is not restrictive.

► **Lemma 6** (\star). *A graph $G = (V, E)$ is a map graph if and only if it admits a compact witness. Also, G is a k -map graph if and only if it admits a compact witness whose intersection vertices have degree at most k .*

In [9], it is observed (without a formal argument) that a map graph is hole-free if and only if it admits a witness whose faces have 4 or 6 edges each. The next characterization improves over this observation and hence can be of independent interest. A connected embedded graph is a *quadrangulation* if each face boundary consists of a single closed walk with 4 edges.

► **Theorem 7** (\star). *A graph is a hole-free map graph if and only if it admits a compact witness that is a biconnected quadrangulation.*

► **Lemma 8** (\star). *A (hole-free) map graph G admits a compact witness with $n \leq 6n_V - 10$ (respectively, $n \leq 3n_V - 4$) vertices.*

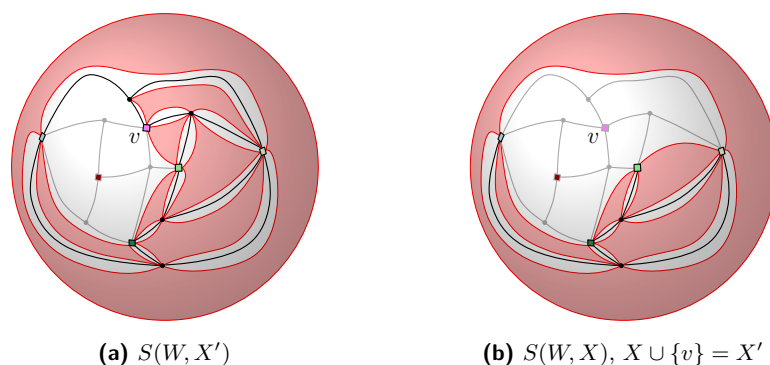
Based on Lemma 8, we can make the following remark.

► **Remark 9.** Without loss of generality, we assume in the following that any compact witness W of G has $n \leq 3n_V - 4$ vertices if G is hole-free, or $n \leq 6n_V - 10$ vertices otherwise.

4 Embedding Sketches

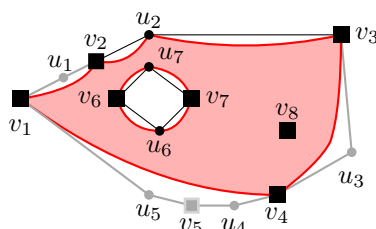
Let G be an input graph. Property 4 allows us to assume that G is biconnected, and thus every witness of G , if any, is connected. Also, by Lemma 6, it suffices to consider compact witnesses.

Let (\mathcal{X}, T) be a nice tree-decomposition of G of width $t = \omega - 1$, i.e., each bag contains at most ω vertices. Given a bag $X \in \mathcal{X}$, we denote by T_X the subtree of T rooted at X , and by $G_X = (V_X, E_X)$ the subgraph of G induced by all the vertices in all bags of T_X . Let $W_X = (V_X \cup I_X, A_X)$ be a compact witness of G_X (in particular, $W_X^2[V_X] = G_X$). Note that, although G is connected, G_X may have multiple connected components. However, since G is connected, each connected component of G_X must contain at least one vertex of X . Moreover, for each connected component C of G_X , there is a connected component C' of W_X such that C' is a witness of C . A vertex of W_X is an *anchor vertex* if it is either a real vertex of X or an intersection vertex whose neighbors in W_X all belong to X . Observe that if an intersection vertex u has a neighbor v in $V_X \setminus X$, then no real vertex in $V \setminus V_X$ is adjacent to v , and therefore there is no way to add further edges to u without creating a false adjacency involving v .



■ **Figure 3** (a) A sketch $S(W, X')$ computed from the witness W of Fig. 1 with respect to a bag X' ($V_{X'} = V$). The anchor vertices of X' are opaque, while the non-anchor vertices are faded. The active boundaries are red and the background of the active faces is light red. (b) A sketch $S(W, X)$, where $X \cup \{v\} = X'$ computed from $S(W, X')$ by applying the deletion operation (Section 5).

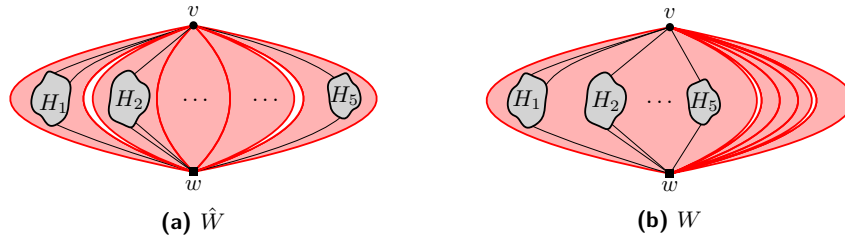
We will exploit anchor vertices to reduce the size of W_X from $O(|V_X|)$ to $O(\omega)$, by “sketching” parts of the embedding that are not relevant¹. The idea of sketching an embedded graph is inspired by a previous work about orthogonal planarity [17]; applying such idea to our problem requires the development of several new tools and concepts, described in the remainder of this section (and partly in Section 2). A face f of W_X is *active* either if its boundary contains only one vertex v (which implies $W_X = (\{v\}, \emptyset)$ and v is an anchor vertex, or if its boundary contains more vertices among which there are at least two anchor vertices; refer to Fig. 3a. The *active boundary* of f (red in Fig. 3a) is obtained by shortcutting all non-anchor vertices of f , where the *shortcut operation* is defined as follows. For a closed walk π and a vertex v in π , shortcutting v consists of removing each occurrence of v (if more than one), together with the edge (u, v) that precedes it in π , and the edge (v, u') that follows it in π , and of adding the edge (u, u') between u and u' in π . Fig. 4 illustrates a single face f



■ **Figure 4** An active boundary (red) made of three closed walks (edges are omitted): $\langle v_1, v_2, u_2, v_3, u_3, v_4, v_1 \rangle$, $\langle u_6, v_6, u_7, v_7 \rangle$, $\langle v_8 \rangle$; vertices u_1, u_3, u_4, u_5, u_5 have been shortcutted.

and the corresponding active boundary. The *embedding sketch* (for short the *sketch*) of W_X with respect to X is the embedded graph $S(W_X, X)$ formed by all the vertices and edges that belong to the active boundaries of W_X . For each active boundary B_f of an active face f of W_X , $S(W_X, X)$ has an *active face* f^* (light red in Figs. 3a and 4). Note that $S(W_X, X)$ also has faces that are not active (white in Figs. 3a and 4). Also, the position system of

¹ In the database and data engineering fields, sketching algorithms form a powerful toolkit to compress data in a way that supports answering various queries [12]. Our idea of sketching has some similarities with this concept but serves a different purpose.



■ **Figure 5** Illustrations for the proof of Lemma 10. Modifying the rotation system of \hat{W} such that each H_i lies in B_1 and all other non-extensible active boundaries become empty.

W_X yields a position system for $S(W_X, X)$, since if two closed walks of distinct components of W_X were incident to the same active face f , then the two corresponding closed walks of $S(W_X, X)$ are also incident to the same active face f^* . However, $S(W_X, X)$ may not be bipartite any longer (as in Fig. 4) and it may contain multiple edges (but no self-loops). It is worth noting that the embedding sketch of W_X can be defined with respect to any bag X' as long as $V_{X'} = V_X$ (see Fig. 3a).

We now further refine $S(W_X, X)$ to avoid active boundaries that are not useful for our purposes. Namely, an active boundary is *non-extensible* if it consists of two homotopic parallel edges. Given a witness W of G , the *restriction* of W to G_X is the compact witness $W[G_X]$ of G_X obtained from W by removing all the real vertices not in G_X , all the intersection vertices that are isolated (due to the removal of some real vertices) or inessential, as well as a vertex for each twin-pair until the graph contains none of them. The next lemmas allow us to bound the size of a sketch.

► **Lemma 10.** *If G is a map graph, then it admits a compact witness W with the following property. If $S(W[G_X], X)$ contains $h > 1$ non-extensible active boundaries that share the same pair of end-vertices, then the vertices of W lie in at most one of these h active boundaries.*

Proof. Refer to Fig. 5. Let \hat{W} be a compact witness of G , and suppose $S(\hat{W}[G_X], X)$ contains $h > 1$ non-extensible active boundaries B_1, B_2, \dots, B_h with common end-vertices v, w . Let H_i be the subgraph of \hat{W} that lies inside B_i (if any), for $1 \leq i \leq h$. Since each B_i consists of two parallel edges, v and w separate H_i and $S(\hat{W}[G_X], X) \setminus H_i$. We obtain a new compact witness W of G by modifying the rotation system of \hat{W} so that each H_i lies inside B_1 . ◀

► **Remark 11.** By Lemma 10, we assume in the following that for any compact witness W of G such that, for some $X \in \mathcal{X}$, the sketch $S(W[G_X], X)$ contains $h > 1$ non-extensible active boundaries, the vertices of W lie in at most one of such active boundaries. Therefore, in $S(W[G_X], X)$, we keep only one of the corresponding h pairs of homotopic parallel edges.

► **Lemma 12.** *A sketch $S(W_X, X)$ contains $O(\omega)$ vertices and edges.*

Proof. With a similar argument as in the proof of Lemma 8 we can show that, in W_X , each real vertex in X is adjacent to $O(\omega)$ intersection vertices that are anchor vertices. Therefore, $S(W_X, X)$ contains $O(\omega)$ vertices in total. Concerning the number of edges, since $S(W_X, X)$ is embedded on the sphere, it contains $O(\omega)$ edges such that each pair of edges is either non-parallel or non-homotopic parallel. In addition, since each of these edges participates in at most one homotopic pair by Remark 11, it follows that $S(W_X, X)$ contains $O(\omega)$ edges. ◀

We now exploit the concept of sketch to define an equivalence relation among witnesses.

► **Definition 13.** *Two compact witnesses W_X and W'_X of G_X are X -equivalent if they have the same sketch with respect to X , i.e., $S(W_X, X) = S(W'_X, X)$.*

The next lemma deals with the size of the quotient of such a relation.

► **Lemma 14.** *The X -equivalence relation yields $\omega^{O(\omega)}$ classes for the compact witnesses of G_X .*

Proof. Let n_1 be the number of possible (abstract) graphs that can be obtained from the real vertices of X and all possible sets of intersection vertices. For each such graph, let n_2 be the maximum number of possible rotation and position systems that it can have. It follows that the number of X -equivalent classes is upper bounded by the product of n_1 and n_2 .

Given the set X of real vertices and a compact witness W_X of G_X , any sketch $S(W_X, X)$ contains $O(\omega)$ intersection vertices, as otherwise W_X would contain inessential intersection vertices or twin-pairs. Since each intersection vertex is adjacent to a set of at most ω real vertices, we can bound the number n_{int} of possible sets of intersection vertices by $a \cdot \sum_{i=2}^{\omega} \binom{\omega}{i} < a \cdot 2^\omega$, where a is the maximum number of intersection vertices in any sketch that have the same set of neighbors. Since $a \in O(\omega)$, we have that $n_{\text{int}} \in 2^{O(\omega)}$. Let I_X be one of the n_{int} possible sets of intersection vertices. The number n_{abs} of distinct abstract graphs with vertex set $X \cup I_X$ can be upper bounded by the number of possible neighborhoods of a real vertex combined for all real vertices, that is

$$n_{\text{abs}} \leq \prod_{v \in X} \omega^{\deg(v)} = \omega^{\sum_{v \in X} \deg(v)} \leq \omega^{O(\omega)}$$

holds, which yields $n_1 \leq n_{\text{int}} \cdot n_{\text{abs}} \in \omega^{O(\omega)}$.

For a fixed graph S , the number of possible rotation systems n_{rot} is upper bounded by the number of possible permutations of edges around each vertex. Thus we have

$$n_{\text{rot}} \leq \prod_{v \in S} \deg(v)! < \prod_{v \in S} \deg(v)^{\deg(v)} \leq \omega^{O(\sum_{v \in S} \deg(v))} \leq \omega^{O(\omega)}.$$

Each rotation system of S fixes the closed walk of each face of each connected component of S . Since S contains, over all its connected components, at most ω closed walks (at most one for each real vertex in X) and hence at most ω faces, for the number n_{pos} of possible position systems it holds $n_{\text{pos}} \leq \omega^\omega$. Therefore we have $n_2 \leq n_{\text{rot}} \cdot n_{\text{pos}} \in \omega^{O(\omega)}$, which yields $n_1 \cdot n_2 \in \omega^{O(\omega)}$, as desired. ◀

5 Algorithmic Framework

Let $G = (V, E)$ be an input graph, let k be an integer, and let (\mathcal{X}, T) be a nice tree-decomposition of G of width $t = \omega - 1$. We present an algorithmic framework to test whether G is a k -map graph or a hole-free k -map graph. Namely, we traverse T bottom-up and equip each bag $X \in \mathcal{X}$ with a suitably defined set of sketches, called *record* R_X . The framework can be tailored by imposing different properties for the records. The next three properties are rather general; the first two are useful to prove the correctness of our approach, as shown in Theorem 22, whereas the third property comes into play when dealing with the efficiency of the approach, and in particular in Lemma 16.

► **Definition 15.** *The record R_X is valid if the following properties hold:*

F1 *For every compact witness W_X of G_X , R_X contains its sketch $S(W_X, X)$.*

F2 *For every entry $r \in R_X$, there is a compact witness W_X of G_X such that $r = S(W_X, X)$.*

F3 *R_X contains no duplicates.*

8:10 Recognizing Map Graphs of Bounded Treewidth

► **Lemma 16.** *For every $X \in \mathcal{X}$, if R_X is valid, it contains $\omega^{O(\omega)}$ entries, each of size $O(\omega)$.*

Proof. By F1–F3, the entries of R_X are all and only the possible sketches of W_X and are all distinct. Hence, $|R_X| \in \omega^{O(\omega)}$ by Lemma 14. Each sketch has size $O(\omega)$ by Lemma 12. ◀

We now describe the additional properties that we incorporate in the framework. In order to verify that G admits a k -map we exploit Property 3, which translates into verifying that, for each sketch, the degree of any intersection vertex is at most k .

► **Definition 17.** *A record R_X is k -map valid if it is valid and it contains a non-empty subset $R_X^* \subseteq R_X$, called subrecord, for which the following additional property holds:*

F4 *For every entry $r \in R_X$, it holds $r \in R_X^*$ if and only if r contains no intersection vertex u with $\deg(u) > k$.*

It is worth observing that, since an intersection vertex of degree k implies the existence of a clique of size k in the input graph G , property F4 is trivially verified when $k \geq \omega$. On the other hand, the size of the largest clique of a k -map graph is $\lfloor 3k/2 \rfloor$ (see, e.g., [9]).

To check whether G has a hole-free k -map, we exploit Theorem 7. Namely, consider a sketch $S(W_X, X)$ and an active boundary B_f of $S(W_X, X)$. Let f be the active face of W_X corresponding to B_f . Note that any edge e that is part of B_f represents a subsequence π_e of a closed walk π in the boundary of f . Therefore, to control the number of edges on the boundary of each face of W_X , for every edge e that is part of an active boundary of $S(W_X, X)$ we also store a counter $c(e) \geq 1$, which represents the number of edges in π_e . If there is an edge e such that $c(e) > 4$, then G does not admit a compact witness W that is a quadrangulation and such that $W_X = W[G_X]$; hence we can avoid storing counters greater than 4. Moreover, for any face f of a compact witness W of G , we know there exist two bags \hat{X}' and \hat{X} in T such that \hat{X}' is the child of \hat{X} , \hat{X} is a forget bag, the active boundary representing f in \hat{X}' has more than one anchor vertex, while the one in \hat{X} has only one anchor vertex (and hence is not part of $S(W_{\hat{X}}, \hat{X})$). We call such an active boundary *complete* in \hat{X}' , as it will not be modified anymore by the algorithm. As such, for each complete active boundary, the sum of the counters of its edges in $S(W_{\hat{X}'}, \hat{X}')$ must be exactly 4, otherwise G does not admit a compact witness W that is a biconnected quadrangulation such that $W_{\hat{X}} = W[G_{\hat{X}}]$.

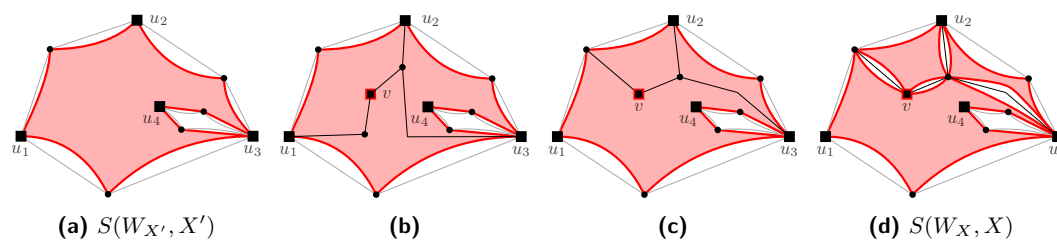
► **Definition 18.** *A record R_X is hole-free valid if it is valid and it contains a non-empty subset $R_X^\circ \subseteq R_X$, called subrecord, for which the following additional property holds:*

F5 *For every entry $r \in R_X$, it holds $r \in R_X^\circ$ if and only if r contains no intersection vertex u with $\deg(u) > k$ and each complete active boundary of r (if any) is such that its edge counters sum up to 4.*

Each leaf bag contains only one vertex v , thus its record consists of one sketch with only one active face whose active boundary is $\langle v \rangle$. Such a record can be computed in $O(1)$ time and it is trivially valid. Also, it is hole-free (and hence k -map) valid, as its unique active boundary is not complete. The next three operations are performed on a non-leaf bag X of T , based on the type of X , to compute a k -map or hole-free valid record R_X , if any.

Deletion operation. Let X be a forget bag whose child X' in T has a k -map (hole-free) valid record $R_{X'}$. Let v be the vertex forgotten by X . We generate R_X from $R_{X'}$ as follows.

For a fixed sketch $S(W_{X'}, X')$ of $R_{X'}$, let $N_I(v) \subseteq N(v)$ be the set of intersection vertices adjacent to v in $S(W_{X'}, X')$. Since v is forgotten by X , all its neighbors have already been processed, thus no vertex in $N_I(v)$ can connect vertices that will be introduced by bags visited after X . Therefore, for every vertex $y \in N_I(v) \cup \{v\}$ and for every sketch $S(W_{X'}, X')$



■ **Figure 6** Illustration for the addition of vertex v . (a) Details of a face of $S(W_{X'}, X')$ that contains all the neighbors of v . (b–c) Two distinct embedded graphs computed from $S(W_{X'}, X')$ by introducing vertex v in different ways (as described in the proof of Lemma 20). (d) The sketch $S(W_X, X)$ obtained by replacing the active boundary of the red face with the new active boundaries corresponding to the three newly created active faces in (c).

of $R_{X'}$, we apply a *deletion operation*, which consists of updating each active boundary B_f of $S(W_{X'}, X')$ containing y ; see Fig. 3b. Namely, let B_f be one of these active boundaries, we distinguish two cases based on whether B_f contains only y or it contains further vertices. Let π_y be the closed walk of B_f that contains all occurrences of y (there might be more than one). If B_f contains only y , we remove π_y (and hence the whole active boundary B_f) from $S(W_{X'}, X')$. If B_f contains further vertices, we shortcut every occurrence of y in π_y . Also for each edge e introduced to shortcut y such that e replaces edges e_1 and e_2 of π_y , we set $c(e) = c(e_1) + c(e_2)$. Observe that, if y has only one neighbor u in π_y , this procedure creates a self-loop at u , which we remove. If this procedure generates more than one pair of homotopic parallel edges with the same pair of end-vertices, then we keep only one such pair. Once all active boundaries have been updated, the resulting embedded graph is stored in R_X . After each sketch of $R_{X'}$ has been processed, we might have produced the same embedded graph for R_X from two distinct sketches of $R_{X'}$; in this case we keep only one copy.

Addition operation. Let X be an introduce bag whose child X' in T has a k -map (hole-free) valid record $R_{X'}$. Let v be the vertex introduced by X and $N_X(v) \subseteq N(v)$ be the set of vertices that are neighbors of v and belong to X . We generate R_X from $R_{X'}$ with the following *addition operation*. For each sketch $S(W_{X'}, X')$ of $R_{X'}$, the high-level idea is to exhaustively generate all possible embedded graphs that can be obtained by introducing v in $S(W_{X'}, X')$. We distinguish two cases.

Case 1. $N_X(v) = \emptyset$. For each active boundary B_f of $S(W_{X'}, X')$, we generate a new embedded graph by adding the closed walk $\langle v \rangle$ to B_f .

Case 2. $N_X(v) \neq \emptyset$. We look for a face f^* of $S(W_{X'}, X')$ that contains all the vertices of $N_X(v)$ on its active boundary B_f (which may consist of multiple closed walks). If such a face does not exist, we discard $S(W_{X'}, X')$. Else, for each such face, we generate a set of entries E_{f^*} as follows. Intuitively, we will insert v inside f^* and generate one entry of E_{f^*} for each possible way in which v can be connected to its neighbors. Namely, we can connect v to its neighbors by means of different intersection vertices and by realizing different permutations of the edges around v and around those neighbors that appear multiple times along some closed walk of B_f ; refer to Fig. 6 for an illustration. Concerning the intersection vertices, we can use those that already belong to B_f and are adjacent only to vertices in $N_X(v)$, as well as we can create new ones. We note that since v has at most $\omega - 1$ neighbors in $N_X(v)$, there are $\sum_{i=1}^{\omega-1} \binom{\omega-1}{i} = 2^{\omega-1}$ possible combinations of intersection vertices (see also the proof of

8:12 Recognizing Map Graphs of Bounded Treewidth

Lemma 14). This is done avoiding inessential intersection vertices and twin-pairs. For each choice of intersection vertices, since the degree of a vertex is $O(\omega)$, there are $\omega^{O(\omega)}$ distinct rotation systems to consider. Additionally, if B_f consists of multiple closed walks, we shall consider all possible permutations of the edges around v that do not cause edge crossings (i.e., any edge permutation in which there are no four edges e_1, e_2, e_3, e_4 in this order around v , such that e_1, e_3 connect v to the vertices of a closed walk π and e_2, e_4 connect v to the vertices of a closed walk π' with $\pi \neq \pi'$), and we consider each of them independently as a new embedded graph. Based on the fixed intersection vertices and rotation system, if the insertion of v does not split f^* into multiple faces, we can suitably update B_f , otherwise we can generate the new active boundaries that appear in place of B_f ; see in particular Fig. 6d. Also, for each newly introduced edge e in a closed walk, we set $c(e) = 1$.

Merge operation. Let X be a join bag whose children X_1 and X_2 in T have k -map (hole-free) valid records R_{X_1} and R_{X_2} , respectively. We generate R_X from R_{X_1} and R_{X_2} . Since X is a join bag, X , X_1 , and X_2 contain the same vertices, whereas G_{X_1} and G_{X_2} only share the vertices in X . Consider any pair of sketches $S(W_{X_1}, X)$ of R_{X_1} and $S(W_{X_2}, X)$ of R_{X_2} . Such sketches share the same set of real vertices, whereas they may have different sets of intersection vertices and different combinatorial embeddings. At high-level, we aim at combining $S(W_{X_1}, X)$ and $S(W_{X_2}, X)$ in all possible ways, provided that the original rotation and position systems of each sketch are preserved and that we never insert a subgraph of one sketch into a non-active face of the other. In practice, we apply the *merge operation*, consisting of the next steps.

- (S.1) We compute all possible unions of the two abstract graphs underlying the two sketches. Namely, let I_{X_1} and I_{X_2} be the sets of intersection vertices of $S(W_{X_1}, X)$ and $S(W_{X_2}, X)$, respectively. We identify each pair of real vertices the two sketches share, and we consider all possible abstract graphs whose set of intersection vertices I_X is such that: (a) $I_X \subseteq I_{X_1} \cup I_{X_2}$; (b) for each intersection vertex of I_{X_1} there is an intersection vertex in I_X with the same set of neighbors, and the same holds for I_{X_2} .
- (S.2) For each generated graph S^* , we compute all combinatorial embeddings, i.e., all possible rotation and position systems yielding a topological embedding on the sphere of S^* . If no such combinatorial embeddings exist, we discard S^* , else we go to the next step.
- (S.3) We generate all possible one-to-one mappings ϕ_1 between intersection vertices of S^* and of $S(W_{X_1}, X)$, and all possible one-to-one mappings ϕ_2 between intersection vertices of S^* and of $S(W_{X_2}, X)$.
- (S.4) We check, for each pair ϕ_1, ϕ_2 , that the restriction of the resulting embedded graph on the real vertices, intersection vertices (up to the mapping defined by ϕ_1 and ϕ_2) and edges of each of the two sketches preserves the corresponding rotation and position systems. If so, we go to the next step; otherwise, we discard the candidate solution.
- (S.5) Since the previous step guaranteed that the active boundaries of each sketch are preserved when looking at the corresponding restriction, we can verify that there is no subgraph of one sketch inside a non-active face of the other.
- (S.6) We suitably update the active boundaries of the resulting embedded graph and we add it to R_X . More precisely, the boundary of a face is active if it does not correspond to a non-active boundary in any of the two sketches and it contains either exactly one anchor vertex or at least two anchor vertices.
- (S.7) We remove inessential intersection vertices and iteratively one intersection vertex for each twin-pair, until there are no twin-pairs.
- (S.8) Once all pairs of sketches have been processed, we remove possible duplicates.

This concludes the description of the main algorithmic steps for proving Theorem 1. Next, we provide lemmas to establish the correctness and the time complexity of these steps.

► **Lemma 19.** *Let X be a forget bag whose child X' in T has a k -map (resp. hole-free) valid record $R_{X'}$. The algorithm either rejects the instance or computes a k -map (resp. hole-free) valid record R_X of X in $\omega^{O(\omega)}$ time.*

Proof. Let v be the vertex forgotten by X . We prove that the record R_X generated by applying the deletion operation is valid, given that $R_{X'}$ is valid. In particular, since we removed possible duplicates, F3 holds and it remains to argue about F1 and F2. To this aim, since X is a forget bag, note that $G_X = G_{X'}$. Hence any compact witness $W_{X'}$ of $G_{X'}$ is also a compact witness of G_X . Moreover, since $R_{X'}$ is valid, it follows by F1 that $R_{X'}$ contains a sketch $S(W_{X'}, X')$ for every compact witness $W_{X'}$. Now since $X' = X \cup \{v\}$, the sketch of $W_{X'}$ with respect to X , namely $S(W_{X'}, X)$, coincides with the one obtained by applying the deletion operation to $S(W_{X'}, X')$. Thus F1 holds for X . Similarly, since $R_{X'}$ is valid, it follows by F2 that every entry of $R_{X'}$ is the sketch $S(W_{X'}, X')$ of a compact witness $W_{X'}$ of $G_{X'}$. Again since $X' = X \cup \{v\}$, the entry of R_X obtained by applying the deletion operation to $S(W_{X'}, X')$ corresponds to the sketch $S(W_{X'}, X)$. Thus F2 holds for X and consequently R_X is valid, as claimed. Suppose now that $R_{X'}$ is k -map valid, i.e., $R_{X'}^* \neq \emptyset$. We show how to check whether a sketch of R_X belongs to R_X^* . Since the deletion operation does not modify the degree of any intersection vertex, the subrecord R_X^* contains all sketches of R_X generated from sketches in $R_{X'}^*$. Based on this observation, we can check whether $R_X^* = \emptyset$ or not. In the former case the algorithm rejects the instance, in the latter case R_X is k -map valid. Suppose that $R_{X'}$ is hole-free valid, i.e., $R_{X'}^\circ \neq \emptyset$. Again the subrecord R_X° contains all sketches of R_X that have been generated from sketches in $R_{X'}^\circ$, and that contain no active boundary whose edge counters sum up to 4. To decide whether an active boundary is complete, it suffices to check whether the parent of X is a forget bag such that the shortcuttings due to the removal of the forgotten vertex make that active boundary a self-loop. If any complete active boundary does not meet this condition, the corresponding sketch does not belong to R_X° . As before if $R_{X'}^\circ = \emptyset$ the algorithm rejects the instance, otherwise R_X is hole-free valid.

By Lemma 16, $R_{X'}$ contains $\omega^{O(\omega)}$ entries, each of size $O(\omega)$. Updating each of them takes $O(\omega)$ time. Also, R_X contains at most as many entries as $R_{X'}$. It follows that removing duplicates can be naively done in $(\omega^{O(\omega)})^2 \in \omega^{O(\omega)}$ time. For the sake of efficiency, if we interpret each rotation and position system together as a number with $\tilde{O}(\omega^2)$ bits, then removing duplicates can be done in $\tilde{O}(\omega^2) \cdot \omega^{O(\omega)} \in \omega^{O(\omega)}$ time by using radix sort (we omit the details as the asymptotic running time would be the same). We have seen that condition F4 is always verified. Checking condition F5 requires scanning each active boundary in R_X and decide whether it is complete or not, and if so to verify whether it will become a self-loop when visiting the parent of X . This can be done in $O(\omega)$ time for each of the $O(\omega)$ active boundaries of each of the $\omega^{O(\omega)}$ sketches, and thus in $\omega^{O(\omega)}$ time overall. Thus R_X and its subrecords can be computed in $\omega^{O(\omega)}$ time, as desired. ◀

► **Lemma 20.** *Let X be an introduce bag whose child X' in T has a k -map (resp. hole-free) valid record $R_{X'}$. The algorithm either rejects the instance or computes a k -map (resp. hole-free) valid record R_X of X in $\omega^{O(\omega)}$ time.*

Proof. Let v be the vertex introduced by X . We prove that the record R_X generated by applying the addition operation is valid, given that $R_{X'}$ is valid. Regarding F1, let $W_{X'}$ and W_X be a witness of $G_{X'}$ and G_X , respectively, such that $W_X[G_{X'}] = W_{X'}$. Since F1

holds for $R_{X'}$, we know that $S(W_{X'}, X') \in R_{X'}$. Observe that the only difference between W_X and $W_{X'}$ lies in the presence of vertex v and of a (possibly empty) set I_v of intersection vertices adjacent to v .

If $N_X(v) = \emptyset$, then v forms a trivial closed walk that might be added in any face of $W_{X'}$ that either consists of exactly one anchor vertex or contains at least two anchor vertices (among possibly other non-anchor vertices). We recall that an active face satisfying the mentioned properties corresponds to an active boundary of the witness' sketch. Also, adding the closed walk to a face that contains more than one vertex, but at most one anchor vertex, on its boundary would imply that the resulting witness cannot be augmented to a witness of G , since G is biconnected. Since Case 1 places v in all possible active boundaries of $S(W_{X'}, X')$, we can conclude that $S(W_X, X)$ belongs to R_X .

On the other hand, if $N_X(v) \neq \emptyset$, then all v 's neighbors belong to a common boundary of some face f of $W_{X'}$, as otherwise the rotation system of W_X would not be compatible with a topological embedding (in particular, some edges would cross each other). Hence all v 's neighbors are part of the same active boundary B_f of $S(W_{X'}, X')$. Since Case 2 exhaustively considers all ways in which v can be inserted into B_f , avoiding inessential intersection vertices and twin-pairs (which cannot belong to W_X since it is compact), we can again conclude that $S(W_X, X)$ belongs to R_X . Consequently F1 holds for R_X .

About F2, it suffices to prove that each entry generated by the addition operation is indeed a sketch of some compact witness of G_X with respect to X . Since F2 holds for $R_{X'}$, the addition operation starts from a sketch $S(W_{X'}, X')$ and it generates new entries in which there are neither inessential intersection vertices nor twin-pairs; therefore, such entries are indeed sketches of compact witnesses, as desired.

Concerning F3, if R_X contained two entries r_1, r_2 that are the same (up to a homeomorphism of the sphere), then r_1 and r_2 would have been originated by the same sketch r of $R_{X'}$, as otherwise either r_1 and r_2 would not be the same or F3 would not hold for $R_{X'}$. On the other hand, since the addition operation inserts v in different ways but without repetitions, it cannot generate two entries that are the same starting from a single entry of $R_{X'}$. Thus F3 holds for R_X .

If $R_{X'}$ is k -map valid, we know that R_X^* contains those sketches of $R_{X'}$ for which the addition operation did not introduce intersection vertices of degree larger than k . Based on this observation, we can check whether $R_X^* = \emptyset$ or not. In the former case the algorithm rejects the instance, in the latter case R_X is k -map valid. The case when $R_{X'}$ is hole-free valid can be proved analogously as in the proof of Lemma 19.

Finally, each single entry constructed by the addition operation can be computed in $O(\omega)$ time and R_X contains $\omega^{O(\omega)}$ entries by Lemma 16. Also, condition F4 can be easily verified in $O(\omega)$ time, for each of the $\omega^{O(\omega)}$ sketches of R_X . Checking condition F5 requires scanning each active boundary in R_X and decide whether it is complete or not. This can be done in $O(\omega)$ time, for each of the $O(\omega)$ active boundaries of each of the $\omega^{O(\omega)}$ sketches, and thus in $\omega^{O(\omega)}$ time overall. Thus R_X and its subrecords can be computed in $\omega^{O(\omega)}$ time. ◀

The proof of the next lemma exploits the merge operation.

► **Lemma 21.** *Let X be a join bag whose children X_1 and X_2 in T both have k -map (resp. hole-free) valid records R_{X_1} and R_{X_2} . The algorithm either rejects the instance or computes a k -map (resp. hole-free) valid record R_X of X in $\omega^{O(\omega)}$ time.*

Proof. We prove that the record R_X generated by applying the merge operation is valid, given that R_{X_1} and R_{X_2} are valid. Consider any compact witness W_X of G_X and its restrictions $W_X[G_{X_1}]$ and $W_X[G_{X_2}]$ to G_{X_1} and G_{X_2} , respectively. By definition of restriction, there must

exist a mapping of the intersection vertices of W_X to the intersection vertices of $W_X[G_{X_1}]$ such that when looking at the restriction of W_X to the real and intersection vertices of $W_X[G_{X_1}]$ (up to the above mentioned mapping), the rotation and position systems of $W_X[G_{X_1}]$ are preserved. The same property must hold for $W_X[G_{X_2}]$. These properties clearly carry over to the corresponding sketches $S(W_X, X)$, $S(W_X[G_{X_1}], X)$, and $S(W_X[G_{X_2}], X)$. Since R_{X_1} and R_{X_2} are valid, they contain $S(W_X[G_{X_1}], X)$ and $S(W_X[G_{X_2}], X)$, respectively. Hence, Steps S.1–S.4 guarantee that the aforementioned mapping is considered and that all the above properties hold on the candidate solutions given by the combination of $S(W_X[G_{X_1}], X)$ and $S(W_X[G_{X_2}], X)$. Moreover, any subgraph of W_X that belongs to $W_X[G_{X_1}]$ but not to $W_X[G_{X_2}]$, except for the shared vertices of X , must lie in an active face of $W_X[G_{X_2}]$ (and vice-versa); if this is not the case, then W_X would not be augmentable to a witness of G , since G is biconnected. This property translates into verifying that any subgraph of $S(W_X[G_{X_1}], X)$ lies in an active face of $S(W_X[G_{X_2}], X)$ (and vice-versa). This is achieved in Step S.5. Step S.6 suitably updates the active boundaries so that a boundary is active only if it represents a face of W_X that either consists of exactly one anchor vertex or contains at least two anchor vertices, as by definition of active boundary. Step S.7 removes inessential intersection vertices and twin-pairs, which is a safe operation because W_X is compact. Therefore we can conclude that $S(W_X, X)$ belongs to R_X , and thus F1 holds for R_X . Concerning F2, any entry S in R_X generated by the merge operation, starting from entries $S(W_{X_1}, X) \in R_{X_1}$ and $S(W_{X_2}, X) \in R_{X_2}$, defines a way to combine the combinatorial embeddings of $S(W_{X_1}, X)$ and $S(W_{X_2}, X)$ at common real vertices and at possibly common (based on some mappings ϕ_1 and ϕ_2) intersection vertices. Such information can be used to combine in the same way the corresponding witnesses W_{X_1} and W_{X_2} , which exist because F2 holds for R_{X_1} and R_{X_2} , respectively. On the other hand, such combination yields a compact witness W_X of G_X with respect to X , whose sketch is S , as desired. Thus F2 holds for R_X . In Step S.8 we remove possible duplicates, hence F3 holds by construction for R_X . Therefore R_X is valid. Since the merge operation does not increase the degree of intersection vertices, and since R_{X_1} and R_{X_2} are k -map valid, the subrecord R_X^* contains all sketches of R_X generated from sketches in $R_{X_1}^*$ and $R_{X_2}^*$. If $R_X^* = \emptyset$, the algorithm rejects the instance, otherwise R_X is k -map valid. If R_{X_1} and R_{X_2} are hole-free valid, R_X° contains all sketches of R_X that are generated from sketches in $R_{X_1}^\circ$ and $R_{X_2}^\circ$ and whose complete active boundaries are such that the edge counters sum up to 4. If $R_X^\circ = \emptyset$, the algorithm rejects the instance, otherwise R_X is hole-free valid.

Concerning the time complexity, we process each pair of sketches, one in R_{X_1} and one in R_{X_2} , and since both R_{X_1} and R_{X_2} are valid, we have $\omega^{O(\omega)}$ such pairs. Each of Steps S.1, S.2, and S.3 generates $\omega^{O(\omega)}$ new entries, and each entry is computed in $O(\omega)$ time. The remaining steps all run in $O(\omega)$ time for each processed entry. Condition F4 can be easily verified in $O(\omega)$ time, for each of the $\omega^{O(\omega)}$ sketches of R_X . Furthermore, verifying condition F5 requires scanning the active boundaries of each entry in R_X and deciding whether it is complete or not. This can also be done in $O(\omega)$ time for each of the $O(\omega)$ active boundaries of each of the $\omega^{O(\omega)}$ sketches, and thus in $\omega^{O(\omega)}$ time overall. Consequently, R_X and its subrecords can be computed in $\omega^{O(\omega)}$ time. ◀

Lemmas 19–21 imply the next theorem, which summarizes the correctness of the approach.

▶ **Theorem 22.** *Let G be a graph in input to the algorithm, along with a nice tree-decomposition (T, \mathcal{X}) of G and an integer $k > 0$. Graph G is a k -map graph, respectively a hole-free k -map graph, if and only if the algorithm reaches the root ρ of T and the record R_ρ is k -map valid, respectively hole-free valid.*

We are finally ready to prove Theorem 1. We recall that if $k \geq n - 1$, recognizing n -vertex (resp. hole-free) k -map graphs coincides with recognizing general n -vertex (resp. hole-free) map graphs.

Proof of Theorem 1. We first discuss the decision version of the problem for a fixed $k > 0$. Namely, the algorithm described below is used in a binary search to find the optimal value of k . Recall that t is the width of the tree decomposition (i.e., $\omega = t + 1$). Note that, if G is a positive instance, then k varies in the range $[1, t + 1]$, since the size of the largest clique of G is at most $t + 1$. Thus the algorithm is executed $O(\log t)$ times, which however does not affect the asymptotic running time.

If G is not biconnected, by Property 4, it is not hole-free, and it is k -map if and only if all its biconnected components are k -map. Hence we run our algorithm on each biconnected component independently. Theorem 22 implies the correctness of the algorithm (which assumes the input graph to be biconnected).

For the time complexity, suppose that G has $h \geq 1$ biconnected components and let n_i be the size of the i -th component C_i , for each $i \leq h$. Decomposing G into its biconnected components takes $O(n + m)$ time [30], where m is the number of edges of G and, since G has treewidth t , it holds $m \in O(n \cdot t^2)$. Given a tree-decomposition of G with $O(n)$ nodes and width t , we can easily derive a tree-decomposition (T_i, \mathcal{X}_i) for each C_i in overall $O(n)$ time, such that each T_i has $O(n_i)$ nodes and width at most t . Then we can apply the algorithm in [3] to obtain, in $O(n_i)$ -time, a nice tree-decomposition of C_i with $O(n_i)$ nodes without increasing the original width. Since each bag is processed in $t^{O(t)}$ time by Lemmas 19–21, the algorithm runs in $t^{O(t)} \cdot n_i$ time for each C_i . Since $\sum_{i=1}^h n_i \in O(n)$, decomposing the graph and applying the algorithm to all its biconnected components takes $t^{O(t)} \cdot n$ time.

To reconstruct a witness of a yes-instance, we store additional pointers for each record (a common practice in dynamic programming). Namely, for each sketch S of a record R_X of a bag X , we store a pointer to the sketch of the child bag X' that generated S , if X is an introduce or forget bag, and we store two pointers to the two sketches of the children bags X_1 and X_2 that generated S , if X is a join bag. With these pointers at hand, we can apply a top-down traversal of T , starting at any sketch of the non-empty subrecord of ρ , and reconstruct the corresponding witness W by incrementally combining the retrieved sketches, except at forget bags (the only points in which we lose information). Suppose first that G is a k -map graph but not hole-free. If G is not biconnected, a witness W^* of G is obtained by merging the witnesses of its biconnected components. Note that distinct witnesses corresponding to distinct biconnected components of G can only share real vertices. Thus, each intersection vertex of W^* has degree at most k and W^* is a certificate by Property 3. Suppose now that G is a hole-free k -map graph. Then G is biconnected and the resulting witness is a biconnected quadrangulation whose intersection vertices have degree at most k , a certificate by Theorem 7. ◀

6 Open Problems

Recognizing general map graphs efficiently remains a major algorithmic challenge. To restrict the complexity of the input, further parameters of interest might be the cluster vertex deletion number [24] and the clique-width [14] of the input graph, as well as the treewidth of the putative witness [28]. Another interesting line of research would be generalizing our framework to recognize (g, k) -map graphs, i.e., those graphs that admit a k -map on a surface of genus g (see, e.g., [19]).

References

- 1 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Intersection-link representations of graphs. *J. Graph Algorithms Appl.*, 21(4):731–755, 2017.
- 2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- 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. doi:10.1006/jagm.1996.0049.
- 4 Franz J. Brandenburg. Characterizing 5-map graphs by 2-fan-crossing graphs. *Discret. Appl. Math.*, 268:10–20, 2019.
- 5 Franz J. Brandenburg. Characterizing and recognizing 4-map graphs. *Algorithmica*, 81(5):1818–1843, 2019.
- 6 Zhi-Zhong Chen. Approximation algorithms for independent sets in map graphs. *J. Algorithms*, 41(1):20–40, 2001.
- 7 Zhi-Zhong Chen. New bounds on the edge number of a k -map graph. *J. Graph Theory*, 55(4):267–290, 2007. doi:10.1002/jgt.20237.
- 8 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Planar map graphs. In *STOC*, pages 514–523. ACM, 1998.
- 9 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Map graphs. *J. ACM*, 49(2):127–138, 2002.
- 10 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Recognizing hole-free 4-map graphs in cubic time. *Algorithmica*, 45(2):227–262, 2006.
- 11 Zhi-Zhong Chen, Xin He, and Ming-Yang Kao. Nonplanar topological inference and political-map graphs. In *SODA*, pages 195–204. ACM/SIAM, 1999.
- 12 Graham Cormode. Data sketching. *ACM Queue*, 15(2):60, 2017.
- 13 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- 14 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993.
- 15 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 16 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r) -center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.
- 17 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Orthogonal planarity testing of bounded treewidth graphs. *J. Comput. Syst. Sci.*, 125:129–148, 2022.
- 18 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.
- 19 Vida Dujmovic, David Eppstein, and David R. Wood. Structure of graphs with locally restricted crossings. *SIAM J. Discret. Math.*, 31(2):805–824, 2017. doi:10.1137/16M1062879.
- 20 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. URL: <https://dl.acm.org/doi/10.1145/3385731>, doi:10.1145/3385731.
- 21 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f -deletion: Approximation, kernelization and optimal FPT algorithms. In *FOCS*, pages 470–479. IEEE, 2012.
- 22 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Decomposition of map graphs with applications. In *ICALP*, volume 132 of *LIPICs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 23 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *SODA*, pages 1563–1575. SIAM, 2012.

8:18 Recognizing Map Graphs of Bounded Treewidth

- 24 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 47(1):196–217, 2010.
- 25 Bart M. P. Jansen, Daniel Lokshantov, and Saket Saurabh. A near-optimal planarization algorithm. In *SODA*, pages 1802–1811. SIAM, 2014.
- 26 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.
- 27 Tomasz Kociumaka and Marcin Pilipczuk. Deleting vertices to graphs of bounded genus. *Algorithmica*, 81(9):3655–3691, 2019.
- 28 Matthias Mnich, Ignaz Rutter, and Jens M. Schmidt. Linear-time recognition of map graphs with outerplanar witness. *Discret. Optim.*, 28:63–77, 2018.
- 29 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
- 30 Robert Endre Tarjan and Uzi Vishkin. Finding biconnected components and computing tree functions in logarithmic parallel time (extended summary). In *FOCS*, pages 12–20. IEEE, 1984. doi:10.1109/SFCS.1984.715896.
- 31 Mikkel Thorup. Map graphs in polynomial time. In *FOCS*, pages 396–405. IEEE, 1998.

A Novel Prediction Setup for Online Speed-Scaling

Antonios Antoniadis ✉🏠¹

University of Twente, Enschede, The Netherlands

Peyman Jabbarzade ✉

University of Maryland, College Park, MD, USA

Golnoosh Shahkarami ✉🏠²

Max Planck Institut für Informatik, Saarbrücken, Germany

Universität des Saarlandes, Saarbrücken, Germany

Abstract

Given the rapid rise in energy demand by data centers and computing systems in general, it is fundamental to incorporate energy considerations when designing (scheduling) algorithms. Machine learning can be a useful approach in practice by predicting the future load of the system based on, for example, historical data. However, the effectiveness of such an approach highly depends on the quality of the predictions and can be quite far from optimal when predictions are sub-par. On the other hand, while providing a worst-case guarantee, classical online algorithms can be pessimistic for large classes of inputs arising in practice.

This paper, in the spirit of the new area of machine learning augmented algorithms, attempts to obtain the best of both worlds for the classical, deadline based, online speed-scaling problem: Based on the introduction of a novel prediction setup, we develop algorithms that (i) obtain provably low energy-consumption in the presence of adequate predictions, and (ii) are robust against inadequate predictions, and (iii) are smooth, i.e., their performance gradually degrades as the prediction error increases.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases learning augmented algorithms, speed-scaling, energy-efficiency, scheduling theory, online algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.9

Related Version *Full Version:* <https://arxiv.org/abs/2112.03082>

1 Introduction

Energy is a major concern in society in general and computing environments in particular. Indeed, data centers alone are estimated to consume 200 terawatt-hours (TWh) per year, which is likely to increase by a factor of 15 by year 2030 [21]. Hardware manufacturers approach this problem by incorporating energy-saving capabilities into their hardware, with the most popular one being *dynamic speed scaling*, i.e., one can adjust the speed of the processor or device. A higher speed implies a higher energy consumption but also more processing capacity. In contrast, a lower speed incurs energy savings while being able to perform less processing per unit of time. Naturally, to take advantage of this energy-saving capability, scheduling algorithms need to decide on what speed to use at each timepoint and consider the energy consumption of the produced schedule alongside more “traditional” quality-of-service considerations.

This paper studies online, deadline-based *speed-scaling* scheduling, augmented with machine-learned predictions. More specifically, a set of jobs \mathcal{J} , each job $j \in \mathcal{J}$ with an associated release time r_j , deadline d_j and processing requirement w_j , arrives online and has to be scheduled on a single speed-scalable processor. A scheduling algorithm needs to decide for each timepoint t on: (i) the processor speed $s(t)$ and (ii) which job $j \in \mathcal{J}$ to execute at t



© Antonios Antoniadis, Peyman Jabbarzade, and Golnoosh Shahkarami;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 9; pp. 9:1–9:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

($j(t)$). Both decisions have to be made by the algorithm at any timepoint t while only having knowledge of the jobs with a release time equal to or less than t . A schedule is said to be feasible if the whole processing requirement of every job j is executed within the respective release time and deadline interval, i.e., if $\int_{t:j(t)=j} s(t)dt \geq w_j$. The energy consumption of a schedule, which we seek to minimize over all feasible schedules, is given by $\int_0^{+\infty} s(t)^\alpha dt$, where $\alpha > 1$ is a constant, which in practice is between 1.1 and 3 depending on the employed technology [15, 31]. The offline setting of the problem in which the complete job set \mathcal{J} including their release times, deadlines, and workloads are known in advance was solved in the seminal paper by Yao, Demers, and Shenker [32] who gave an optimal offline algorithm called YDS. The arguably more interesting online setting in which the characteristics of a job j only become known at its release time r_j has been extensively studied [32, 10, 12, 11, 3], and the currently best known online algorithm is qOA, by Bansal et al. [11] achieving a competitive ratio of $4^\alpha / (2e^{1/2}\alpha^{1/4})$.

However, the purely online setting may be too restrictive in many practical scenarios for which one can predict – with reasonable accuracy – the characteristics of future jobs, for example, by employing a learning approach on historical data. *Learning augmented algorithms* is a very novel research area (arguably first introduced in 2018 by Lykouris and Vassilvitskii [23]) trying to capture such scenarios in which predictions of uncertain quality are available for future parts of the input. The goal in learning augmented algorithms is to design algorithms that are at the same time (i) *consistent*, i.e., obtain an improved competitive ratio in the presence of adequate predictions, (ii) *robust*, i.e., there is a worst case guarantee independently of the prediction accuracy (ideally within a constant factor of the competitive ratio of the best known online algorithm that does not employ any predictions) and (iii) *smooth*, i.e., the performance guarantee degrades gracefully with the quality of the predictions.

Previous Predictions Setups and Our Setup

Online Speed-Scaling with machine learned predictions has been investigated before by Bamas et al. [8] who consider a prediction setup in a sense orthogonal to ours; the release times and deadlines of jobs are known in advance, and there is a prediction on the processing requirement. Although any input instance (with integer release times and deadlines) can be modeled in such a way (by considering all possible pairs of release times and deadlines and a processing requirement of zero for the pairs that do not correspond to a job), this can be computationally quite expensive. Bamas et al. present a consistent, robust, and smooth algorithm for the particular case in which the interval length of each job is the same and generalize their consistency and robustness results to the general case (in which each job can have an arbitrary interval length). For this more general setting, the proof of smoothness is omitted because “... the prediction model and the measure of error quickly get complex and notation heavy”.

In the current paper, we consider the novel prediction setup in which predictions on the release times and deadlines are provided to the algorithm. To keep the model simple, we assume that the actual processing requirement of each job $j \in \mathcal{J}$, as well as the number of jobs n are known. It may be useful for the reader to think about our setup as having as many unit-size jobs as total processing volume in the instance, and a prediction on the release time and deadline of each such job. We note, however, that our actual setup requires significantly fewer predictions than this simplified one.

In this context, the main contribution of the current paper is to introduce a natural alternative prediction setup and error measure as well as an algorithm (SWP) within that setup, which possesses the desired properties of consistency, smoothness, and robustness in

the general setting. It should be pointed out that since the two papers consider different prediction settings and in turn also error measures, the algorithms as well as their guarantees are incomparable. However one can consider the two prediction setups as complementary of each other.

Our Contribution

We show how the predictions can be used to develop an algorithm called SCHEDULING WITH PREDICTIONS (SWP), that improves upon qOA when the predictions are reasonably accurate. More formally, in Section 3 we show the following theorem:

► **Theorem 1.** *Given a parameter λ , algorithm SWP achieves a competitive ratio of $\left(\frac{1}{1-\mu}\right)^{\alpha-1} \left(\frac{2\eta+1}{1-2\lambda}\right)^{\alpha-1}$ if $\eta \in (0, \lambda)$, and $2^{\alpha-1}\alpha^\alpha \left(\frac{1}{\mu}\right)^{\alpha-1}$ otherwise.*

Here, η is the *error* of the prediction (defined formally later) that captures the distance between the predicted and the actual input instances, and $0 \leq \lambda < 1/2, 0 \leq \mu \leq 1$ are two hyperparameters that can be thought of as the confidence in the prediction. Theorem 1 implies that SWP is at the same time *consistent*, *smooth* and *robust* where the exact consistency, smoothness and robustness depend on the choice of the hyperparameters λ and μ .

Additionally, in Section 4 we obtain improved results for the restricted case in which all jobs have a common deadline d , and we are given predictions regarding the release times of the jobs. The corresponding algorithm is called COMMON-DEADLINE-SCHEDULING WITH PREDICTIONS (CDSWP) and obtains the following improved competitive ratio:

► **Theorem 2.** *Given a parameter λ , algorithm CDSWP achieves a competitive ratio of $\left(\frac{1+\eta}{1-\lambda}\right)^{\alpha-1}$ if $\eta \in (0, \lambda)$, and $2^\alpha \left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1}$ otherwise.*

Although restricted, this case seems to capture the difficulty of the online setting for the problem, as supported by the fact that the strongest lower bound of $e^{\alpha-1}/\alpha$ on the competitive ratio for online algorithms for the problem is proven on such an instance [11].

Finally, in Section 5 we present an empirical evaluation of our results on a real-world data-set, which suggests the practicality of our algorithm. The actual results are preceded by Section 2 which contains preliminary results and observations. All omitted proofs can be found in the supplementary material.

1.1 Related Work

1.1.1 Online Energy-Efficient Scheduling

As already mentioned, speed-scaling was first studied from an algorithmic point of view by [32]. They studied the deadline-based version of the problem also considered here, and in addition to providing the optimal offline algorithm called *YDS*, two online algorithms called *Optimal Available (OA)* and *Average Rate (AVR)*. OA recalculates an optimal offline schedule for the remaining instance at each release time, whereas AVR “spreads” the processing volume equally between its release time and deadline in order to determine the speed for each timepoint t . The actual schedule then is simply an *Earliest Deadline First (EDF)* schedule with these speeds. They show that AVR obtains a competitive ratio of $2^{\alpha-1}\alpha^\alpha$ which is essentially tight as shown by [10]. Algorithm OA, on the other hand, was analyzed by [11] who proved a tight competitive ratio of α^α .

The currently best known algorithm for the problem, at least for modern processors which satisfy $\alpha = 3$ is the aforementioned qOA algorithm, which for any parameter $q \geq 1$ sets the speed of the processor to be q times the speed that the optimal offline algorithm would run the jobs in the current state. Algorithm qOA attains a competitive ratio of $4^\alpha / (2e^{1/2}\alpha^{1/4})$, for $q = 2 - 1/\alpha \approx 1.667$.

The multiprocessor version of online, deadline-based, speed-scaling has also been studied, see [3, 5] as well as other objectives, for example, flow time [4, 13]. We refer the interested reader to surveys [2, 20].

1.1.2 Further Results on Learning Augmented Algorithms

[23] was arguably the seminal paper in the area, considered the online caching problem. Subsequently, [27] considered the ski-rental problem as well as non-clairvoyant scheduling. Similar to the current work, the robustness and consistency guarantees were given as a function of a hyperparameter that is part of the input to the algorithm. Both the caching and the ski-rental problem have since been extensively studied in the literature (see for example [28, 6, 30] and [29, 18]).

Several other online problems have been investigated through the lens of learning-augmented algorithms and results of similar flavor were obtained. Examples include scheduling and queuing problems [26, 24], online selection and matching problems [7, 16], or the more general framework of online primal-dual algorithms [9]. We direct the interested reader to a recent survey [25].

2 Preliminaries

We consider *online, deadline-based speed-scaling* as described in the introduction. Given a scheduling algorithm A on the set jobs \mathcal{J} , the energy consumption of A on \mathcal{J} is denoted by $\mathcal{E}_A(\mathcal{J})$. When clear from the context, we may write \mathcal{E}_A instead of $\mathcal{E}_A(\mathcal{J})$ to simplify the notation.

As usual for online problems, the performance guarantees are given by employing *competitive analysis*. Following the speed-scaling literature (see for example [11]) we use the *strict competitive ratio*. Formally, the (strict) competitive ratio of algorithm A for the online, deadline-based speed-scaling problem, on input instance I is given by

$$\max_I \frac{\mathcal{E}_A(I)}{\mathcal{E}_{YDS}(I)},$$

where $\mathcal{E}_A(I)$ is the cost that algorithm A incurs on instance I , and the maximum is taken over all possible input instances I . The competitive ratio in many cases will depend on the prediction error.

Prediction Setup

The algorithm initially gets information about the number of jobs n , the corresponding processing volumes $w_j, \forall j \in \mathcal{J}$, as well as for every job $j \in \mathcal{J}$ a prediction p_j for the release time r_j and another prediction q_j for the deadline d_j . Again, the actual values of r_j and d_j only become known at timepoint r_j . Let $R = \{r_1, \dots, r_n\}$, $D = \{d_1, \dots, d_n\}$, $P = \{p_1, \dots, p_n\}$ and $Q = \{q_1, \dots, q_n\}$. Note that in the special case where all jobs have a common deadline d we naturally only obtain predictions for the release times.

The quality of the prediction is measured in terms of a *prediction error* η , which intuitively η measures the distance between the predicted values and the actual ones. We start by defining the individual prediction error η_i for each job $i \in \mathcal{J}$.

► **Definition 3.** Let the prediction error for job i be $\eta_i = \max \left\{ \frac{|p_i - r_i|}{q_i - p_i}, \frac{|q_i - d_i|}{q_i - p_i} \right\}$.

Note that we implicitly assume that $p_i \leq q_i$ for all $i \in \mathcal{J}$ since otherwise, it is immediately obvious that the quality of the predictions is low, and one could just run a classical online algorithm for the problem. Furthermore, if the instance has a common deadline then η_i simplifies to $\eta_i = \frac{|p_i - r_i|}{d - p_i}$.

The (total) prediction error η of an input instance is then given by $\eta = \max_i \eta_i$. We call this *max-norm-error*.

► **Definition 4.** We say that the total error η is a max-norm-error if η is given by the infinity norm of the vector of the respective errors for each job. More formally,

$$\eta = \|\boldsymbol{\eta}\|_\infty = \max(\eta_1, \eta_2, \dots, \eta_n).$$

Performance Guarantees

In the following we formalize the performance guarantees used to evaluate our algorithms.

► **Definition 5.** We say that an algorithm within the above prediction setup is:

- Consistent, if its competitive ratio is strictly better than that of the best online algorithm without predictions for the problem, whenever $\eta = 0$.
- Robust, if its competitive ratio is within a constant factor from that of the best online algorithm without predictions for the problem. Note that Robustness is independent of the prediction quality.
- Smooth, if its competitive ratio is a smooth function of η .

Shrinking of Intervals

The most straightforward way to consider the predictions would arguably be to blindly trust the predictions, i.e., schedule jobs assuming that the predicted instance is the actual instance.

Consider the instance J_{PQ} (resp. J_{RD}) in which every job has the corresponding predicted (resp. actual) release time and deadline. The naive algorithm would compute the optimal offline schedule $YDS(J_{PQ})$ and try to schedule tasks according to it. If the predictions are perfectly accurate, then this clearly is an optimal schedule, and the best one can do. However, if the predictions are even slightly inaccurate, then the resulting schedule may be infeasible. Moreover, our goal is to have a robust algorithm, which cannot be obtained by following the predictions blindly. For these reasons, one has to trust the predictions more cautiously and not blindly.

One of our crucial ideas is to slightly shrink the interval between each job's release time and deadline before scheduling it. The intuition is that if the predictions are only slightly off, then a YDS schedule for the newly obtained instance will be feasible at a slight increase in energy consumption over the YDS schedule of the predicted instance. The following lemmas formalize this intuition. We note that a similar result is also presented in [8]; however, given that the actual setups are different new proofs are required (the proofs of these lemmas can be found in the supplementary material).

9:6 A Novel Prediction Setup for Online Speed-Scaling

► **Lemma 6.** Consider a common deadline instance \mathcal{J} , and another common deadline instance $\hat{\mathcal{J}}$ constructed from \mathcal{J} such that every job $\hat{j}_i \in \hat{\mathcal{J}}$ has workload $\hat{w}_i = w_i$, $\hat{d} = d$, and $\hat{r}_i = r_i + (1 - c_i) \cdot (d - r_i)$ for some shrinking parameter $0 < c_i \leq 1$. Set $c = \max_i c_i$. Then,

$$\mathcal{E}_{YDS}(\hat{\mathcal{J}}) \leq (1/c)^{\alpha-1} \mathcal{E}_{YDS}(\mathcal{J}).$$

► **Lemma 7.** Consider a (general) instance \mathcal{J} , and another instance $\hat{\mathcal{J}}$ in which every job $j_i \in \mathcal{J}$ corresponds to a job $\hat{j}_i \in \hat{\mathcal{J}}$ with workload $\hat{w}_i = w_i$, $\hat{r}_i = r_i + \frac{1-c}{2} \cdot (d_i - r_i)$ and $\hat{d}_i = d_i - \frac{1-c}{2} \cdot (d_i - r_i)$ for some shrinking parameter $0 < c \leq 1$. Then,

$$\mathcal{E}_{YDS}(\hat{\mathcal{J}}) \leq (1/c)^{\alpha-1} \mathcal{E}_{YDS}(\mathcal{J}).$$

It will be useful to bound the energy consumption of (the possibly infeasible for the original input instance) schedule $YDS(J_{PQ})$. We compute the energy consumption of schedule $YDS(J_{PQ})$ in the following lemma.

► **Lemma 8.** For any $\eta \geq 0$ there holds

$$\mathcal{E}_{YDS(J_{PQ})} \leq (2\eta + 1)^{\alpha-1} \mathcal{E}_{YDS(J_{RD})}.$$

Proof. Consider two sets $P^* = \{p_1^*, \dots, p_n^*\}$ and $Q^* = \{q_1^*, \dots, q_n^*\}$, with $p_i^* = p_i - \eta_i(q_i - p_i)$ and $q_i^* = q_i + \eta_i(q_i - p_i)$.

By the definition of η_i , p_i^* and q_i^* , we have $(r_i, d_i) \subseteq (p_i^*, q_i^*)$, and therefore

$$\mathcal{E}_{YDS(J_{P^*Q^*})} \leq \mathcal{E}_{YDS(J_{RD})}.$$

By having $c = \frac{1}{(2\eta+1)}$, and $J = J_{P^*Q^*}$ ($r_i = p_i^*, d_i = q_i^*$) in Lemma 7, we obtain $J' = J_{PQ}$ and therefore,

$$\mathcal{E}_{YDS(J_{PQ})} \leq (2\eta + 1)^{\alpha-1} \mathcal{E}_{YDS(J_{P^*Q^*})}. \quad \blacktriangleleft$$

Using Lemma 6, we can obtain a similar result for common deadline instances.

► **Corollary 9.** In common deadline instances for any parameter $\eta \geq 0$, there holds

$$\mathcal{E}_{YDS(J_P)} \leq (\eta + 1)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

The idea of shrinking intervals as described above will be useful for the general case as well as the restricted common deadline case.

How much each algorithm will shrink the predicted job intervals will depend on the *confidence*. This will be denoted by a *confidence parameter* $0 < \lambda \leq 1/2$ that will be given as input to the respective algorithm. In the following, we define the *shrunk prediction set* of release times and deadlines parametrized by this λ , and use the above lemmas to argue about how this “shrinking” actually affects the energy consumption of the corresponding YDS -schedule.

► **Definition 10.** Let $P' = \{p'_1, \dots, p'_n\}$ and $Q' = \{q'_1, \dots, q'_n\}$ be the *shrunk prediction set* of release times and deadlines respectively in which $p'_i = \lfloor p_i + \lambda(q_i - p_i) \rfloor$ and $q'_i = \lceil q_i - \lambda(q_i - p_i) \rceil$ for all $i \in [n]$.

We first observe that any schedule that considers the sets P' and Q' as the actual release times and deadlines of the jobs will be feasible, as long as the error η is not larger than λ .

► **Observation 11.** *Under the assumption that $\eta \in (0, \lambda)$, it follows that $r_i \leq p'_i$ and $q'_i \leq d_i$ hold for every job i .*

Therefore, the schedule $YDS(J_{P'Q'})$ is feasible. Although shrinking the intervals and then running YDS is not a robust algorithm, it will be useful to bound its energy consumption when $\eta \leq \lambda$ holds.

► **Lemma 12.** *For any $\eta \in (0, \lambda)$ there holds*

$$\mathcal{E}_{YDS(J_{P'Q'})} \leq \left(\frac{2\eta + 1}{1 - 2\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_{RD})}.$$

Proof.

$$\mathcal{E}_{YDS(J_{P'Q'})} \leq \frac{1}{(1 - 2\lambda)^{(\alpha-1)}} \cdot \mathcal{E}_{YDS(J_{PQ})} \leq \left(\frac{2\eta + 1}{1 - 2\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_{RD})}. \quad (1)$$

By Lemma 7 we have the first inequality in (1), and the second inequality holds because of Lemma 8. ◀

Similarly for common deadline instances, since we shrink from one side, we obtain a better competitive ratio.

► **Corollary 13.** *For any $\eta \in (0, \lambda)$ in common deadline instances there holds*

$$\mathcal{E}_{YDS(J_{P'})} \leq \left(\frac{1 + \eta}{1 - \lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

Proof.

$$\mathcal{E}_{YDS(J_{P'})} \leq \frac{1}{(1 - \lambda)^{(\alpha-1)}} \cdot \mathcal{E}_{YDS(J_P)} \leq \left(\frac{1 + \eta}{1 - \lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

By Lemma 6 we have the first inequality, and the second inequality holds because of Corollary 9. ◀

3 General Case

In this section we present algorithm `SCHEDULEWITHPREDICTIONS`(λ, μ) (`SWP`(λ, μ) for short) for the general learning-augmented speed-scaling setting. Parameter $0 \leq \lambda < 1/2$ describes for which range of prediction errors we would like to obtain an improved competitive ratio. The smaller the λ , the smaller that range but the better the corresponding competitive ratio for $\eta < \lambda$. On the other hand, parameter $0 \leq \mu \leq 1$ allows us to set the desired trade-off between consistency and robustness. As we will see, perfect predictions and $\lambda = \mu = 0$ would give a competitive ratio of 1.

Inspired by [8] algorithm `SWP` begins by partitioning each time slot $I_t = [t, t + 1), t \in \mathbb{Z}$ into two parts: $I_t^\ell = [t, t + (1 - \mu))$ and $I_t^r = [t + (1 - \mu), t + 1)$. We call I_t^ℓ the *left part*, and I_t^r the *right part* of time slot I_t . The idea is to reserve the left parts of time-slots for following the prediction, and the right parts of the time-slots are, roughly speaking, intended for safeguarding against inaccurate predictions. A key component of our algorithm consists of elegantly and dynamically distributing the processing volume of each job upon its arrival among the two parts. This distribution is crucial in order to obtain a trade-off between consistency and robustness, based on the parameters λ and μ . The algorithm consists of two steps, the *preprocessing* and the *online* step which we now describe in more detail.

Preprocessing: Partition left parts into intervals and assign jobs to them

Upon receiving the predictions (P, Q) , SWP computes a YDS-schedule S' for instance (P', Q') – which is obtained by “shrinking” (P, Q) as described above. Although S' may not be feasible for the actual instance (R, D) , it will be used to partition the left parts into intervals and subsequently assign each such interval of the partition to a specific job.

To this end, let $I_t(j) := [t + a_t(j), t + b_t(j)] \subseteq I_t$ be the maximal subinterval of I_t during which j is executed under S' . Note that $I_t(j)$ could be empty for some combinations of j and t . Furthermore, since by definition there are no release times or deadlines within I_t , and YDS schedules according to EDF, there can be at most one execution interval of j within I_t . Let, for every job j and left part I_t^l , $I_t^l(j) := [t + a_t^l(j), t + b_t^l(j)]$, where $a_t^l(j) = a_t(j)/(1 - \mu)$ and $b_t^l(j) = b_t(j)/(1 - \mu)$, be the subinterval of I_t^l assigned to job j .

To obtain some intuition, scheduling the whole processing volume of each job j at a uniform speed throughout intervals $I_t^l(j)$ would result in a “compressed” version of $YDS(P'Q')$ where each time-slot is sped-up by a factor of $1/(1 - \mu)$ to fit in the left part only, thus having an energy consumption increased by a factor of $(1/(1 - \mu))^\alpha$ over that of $YDS(P'Q')$. Although (as we will see) such a compressed schedule would be consistent, it may not be robust (or even feasible) in the presence of subpar predictions. For this reason, we will eventually only schedule part of the volume of each job in the associated left parts whenever feasible, and the remaining volume will be processed on right parts.

Online Step: Job arrivals and processing

SWP needs to decide exactly when each job is to be processed within each time-slot and at what speed. This is done by (i) distributing the processing volume of each job j to right parts of different time-slots I_t and associated left parts $I_t^l(j)$ upon its arrival, and (ii) feasibly scheduling the whole volume assigned to the current time-slot (both to its left and right part), within the time-slot itself. In the following we discuss how this is accomplished.

(i) Job Arrivals: Upon arrival of job j at r_j , let $\delta_j = w_j/(d_j - r_j)$ be its density and $\ell(j) := \sum_{t \in [r_j, d_j]} |I_t^l(j)|$ be the total processing time reserved for job j on the left parts during the preprocessing step that can actually be feasibly used for job j . Furthermore let $V_t(j)$ be the total volume currently (from jobs $1, 2, \dots, j - 1$) assigned to I_t^r , for all t (thus $V_t(1) = 0$).

The algorithm assigns some amount of volume y_j^t (to be determined later) of job j to interval I_t^r (thus $V_t(j + 1) := V_t(j) + y_j^t$), for all $t \in [r_j, d_j]$, with $0 \leq y_j^t \leq \delta_j$. Finally the remaining volume $X_j := w_j - \sum_t y_j^t$ is assigned to the left parts $I_t^l(j)$ with $t \in [r_j, d_j]$, proportionally to their length, i.e., an interval $I_t^l(j)$ with $t \in [r_j, d_j]$ receives an $|I_t^l(j)|/\ell(j)$ -fraction of X_j which implies that the average speed within $I_t^l(j)$ must be $X_j/\ell(j)$. To gain some intuition on the values of y_j^t , it is useful to think of the algorithm as waterfilling the volume of j to both the left and the right parts such that no right part receives more than δ_j amount of volume. More formally, the y_j^t , with $0 \leq y_j^t \leq \delta_j$ and $t \in [r_j, d_j]$ are defined such that they satisfy the following inequalities:

$$\frac{V_t(j)}{\mu} \geq X_j/\ell(j) \quad \forall t \in [r_j, d_j] \text{ with } y_j^t = 0 \tag{2}$$

$$\frac{V_t(j) + y_j^t}{\mu} = X_j/\ell(j) \quad \forall t \in [r_j, d_j] \text{ with } 0 < y_j^t < \delta_j \tag{3}$$

$$\frac{V_t(j) + y_j^t}{\mu} \leq X_j/\ell(j) \quad \forall t \in [r_j, d_j] \text{ with } y_j^t = \delta_j. \tag{4}$$

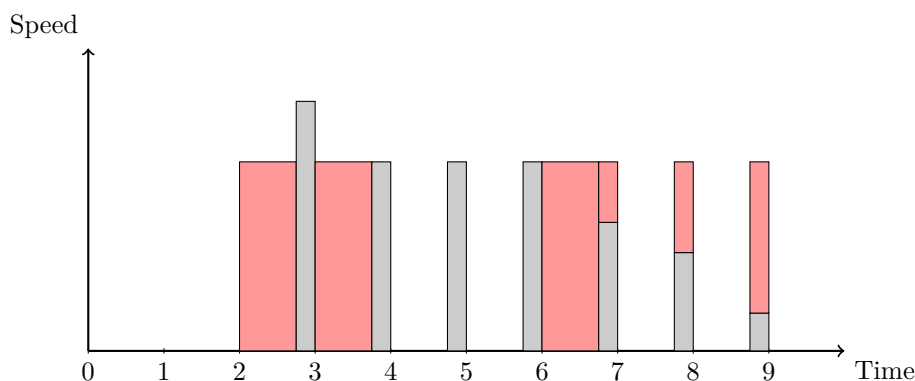


Figure 1 The speed profile corresponds to an instance with $\mu = 0.25$. Job i arrives at $r_i = 2$, with $d_i = 9$, and $w_i = 7$. Hence, $\delta_i = \frac{w_i}{d_i - r_i} = 1$. For this instance, $YDS(P'Q')$ runs job i only in 3 blocks, so we have $\ell(i) = 3 \cdot 0.75 = 2.25$. For the first four blocks we have $y_i^t = 0$ and inequality (2) holds. In the fifth and sixth blocks, $0 < y_i^t < \delta_i$ and equality (3) holds. And in the last block, $y_i^t = \delta_i = 1$ and inequality (4) holds.

Note that the left hand side in each of the above inequalities corresponds exactly to $V_t(j+1)/\mu$ and therefore to the average speed required to process the volume assigned to t before the arrival of job $j+1$ within I_t^r . We prove the existence of such y_j^t and describe how they can be computed in Appendix A.

(ii) Processing: For each $I_t, t = r_j, \dots, r_{j+1} - 1$ the algorithm processes job $j' \leq j$ within every $I_t^\ell(j')$ at a speed of $X_{j'}/\ell(j')$, and the assigned volume to I_t^r is processed within I_t^r at a speed of $V_t(j+1)/\mu$, with the order of the jobs within each I_t^r being determined by EDF.

The online step gets repeated upon the arrival of each job. We next show that the resulting schedule is feasible.

► Lemma 14. *In the schedule output by $SWP(\lambda, \mu)$ a volume of w_j is fully processed for each job j within $[r_j, d_j]$.*

Proof. It is relatively easy to see that a total volume of w_j is assigned to left and right parts of I_t 's with $t \in [r_j, d_j]$: Indeed, by the algorithm definition volume of w_j only gets assigned to $I_t^\ell \subset I_t$ or $I_t^r \subset I_t$ with $t \in [r_j, d_j]$. In addition, a volume of $\sum_t y_j^t$ gets assigned to the right parts and $w_j - \sum_t y_j^t$ to the left parts for a total volume assignment of w_j . It therefore remains to show that all the assigned volume is feasibly processed in the processing step.

Consider some I_t with $r_j \leq t < r_{j+1}$ and the corresponding I_t^ℓ and I_t^r . Note that by the above argument, for any such t no job with an index greater than j will assign any volume, and that only job $j' \leq j$ may be assigned to $I_t^\ell(j')$. Therefore a speed of $X_{j'}/\ell(j')$ throughout every such $I_t^\ell(j)$ is sufficient to schedule all volume assigned to it. Finally, since there are no release times or deadlines within each individual interval, the total volume of $V_t(j+1)$ can be feasibly scheduled within I_t^r at a speed of $V_t(j+1)/\mu$. ◀

We next show consistency and robustness of the algorithm.

► Lemma 15 (Consistency & Smoothness). *For any $\eta \in (0, \lambda)$ there holds*

$$\mathcal{E}_{SWP} \leq \left(\frac{1}{1-\mu} \right)^{\alpha-1} \left(\frac{2\eta+1}{1-2\lambda} \right)^{\alpha-1} \mathcal{E}_{YDS(RD)}.$$

9:10 A Novel Prediction Setup for Online Speed-Scaling

Proof. We can express \mathcal{E}_{SWP} as:

$$\begin{aligned} \mathcal{E}_{\text{SWP}} &= \sum_{j=1}^n \left(\frac{X_j^\alpha}{\ell(j)^{\alpha-1}} + \sum_{t \in [r_j, d_j)} \left(\frac{V_t(j+1)^\alpha}{\mu^{\alpha-1}} - \frac{V_t(j)^\alpha}{\mu^{\alpha-1}} \right) \right) \\ &= \sum_{j=1}^n \left(\frac{X_j^\alpha}{\ell(j)^{\alpha-1}} + \sum_{t \in [r_j, d_j)} \left(\frac{(V_t(j) + y_j^t)^\alpha}{\mu^{\alpha-1}} - \frac{V_t(j)^\alpha}{\mu^{\alpha-1}} \right) \right) \\ &\leq \sum_{j=1}^n \frac{w_j^\alpha}{\ell(j)^{\alpha-1}} = \left(\frac{1}{1-\mu} \right)^{\alpha-1} \mathcal{E}_{\text{YDS}(J_{P'Q'})}. \end{aligned}$$

The inequality holds by convexity of the power function and by the fact that $V_t(j+1)/\mu \leq X_j/\ell(j)$ for each t such that $y_j^t > 0$ (Equations 3 and 4). The last equality follows since for $\eta \in (0, \lambda)$, for every job j there holds $[r_j, d_j) \supseteq [p'_j, q'_j)$ (Observation 11), and by construction $\ell(j)$ is $1/(1-\mu)$ times the total processing time reserved for job j under $\text{YDS}(P'Q')$.

The lemma directly follows, since by Lemma 12,

$$\mathcal{E}_{\text{YDS}(J_{P'Q'})} \leq \left(\frac{2\eta + 1}{1 - 2\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{\text{YDS}(J_{RD})}. \quad \blacktriangleleft$$

► **Lemma 16 (Robustness).** *For any instance, we have*

$$\mathcal{E}_{\text{SWP}} \leq 2^{\alpha-1} \alpha^\alpha \left(\frac{1}{\mu} \right)^{\alpha-1} \mathcal{E}_{\text{YDS}(J_{RD})}.$$

Proof. Note that by the algorithm definition there holds that $V_t(j) \geq V_t(i)$, for $j > i$ and any t , since upon each release time new volume gets assigned but volume never gets removed. We therefore have

$$\begin{aligned} \mathcal{E}_{\text{SWP}} &\leq \sum_{j=1}^n \left(\frac{X_j^\alpha}{\ell(j)^{\alpha-1}} \right) + \sum_t \left(\frac{V_t(n+1)^\alpha}{\mu^{\alpha-1}} \right) \\ &\leq \sum_t \left(\frac{\left(\sum_{j: t \in [r_j, d_j)} \delta_j \right)^\alpha}{\mu^{\alpha-1}} \right) = \left(\frac{1}{\mu} \right)^{\alpha-1} \mathcal{E}_{\text{AVR}}. \end{aligned}$$

The second inequality follows by the convexity of the power function and the fact that $V_t(n+1)/\mu \geq V_t(j+1)/\mu \geq X_j/\ell(j)$ for each t such that $y_j^t < \delta_j$ (Equations 3 and 2). The lemma follows by the competitive ratio of AVR [10]. \blacktriangleleft

Lemmas 15 and 16 together directly imply Theorem 1. Note that Theorem 1 not only implies consistency and robustness, but also smoothness: the competitive ratio gracefully degrades as the error increases.

4 All Jobs Have a Common Deadline

In this section, we present a simpler algorithm that achieves improved consistency and robustness over SWP for the special case in which all jobs have the same deadline, i.e., $d_j = d$ for all $j \in \mathcal{J}$. Since the deadline is the same for all jobs, we only consider predictions on the n release times $R = \{r_1, \dots, r_n\}$ and denote these by a set $P = \{p_1, \dots, p_n\}$.

We begin by analyzing a framework for combining different algorithms before presenting an algorithm in Subsection 4.1 that is based on combining two different algorithms; the classic online algorithm qOA that has a worst-case guarantee independent of the prediction error, and a second one, that considers the predictions and has a good performance in the case of small prediction error.

The general idea of combining online algorithms has been repeatedly employed in the past in the areas of online algorithms and online learning, see, for example, the celebrated results of Fiat et al. [17], Blum and Burch [14], Herbster and Warmuth [19], Littlestone and Warmuth [22]. Such a technique has also been used in the learning augmented setting, see Antoniadis et al. [6] for an explicit framework for combining algorithms, and Lykouris and Vassilvtiskii [23] as well as Rohatgi [28] for implicit uses of such algorithm combinations. However, as we will see, the specific problem considered in this paper allows for way more flexibility in such algorithm combinations since it is possible to simulate the parallel execution of different algorithms by increasing the speed. This allows us to obtain a much more tailored result with at most one switch between the different algorithms and more straightforward analysis. We start with the following structural lemma.

► **Lemma 17.** *Consider a partition of the job set of instance J into m job sets J_1, J_2, \dots, J_m , and furthermore consider m schedules C_1, C_2, \dots, C_m with speed functions $s_1(t), s_2(t), \dots, s_m(t)$ respectively, such that C_i is a feasible schedule for J_i for all $i = 1, \dots, m$. Then there exists a schedule C with speed function $s_C(t) = \sum_i s_i(t)$ that is feasible for the complete job set J and has an energy consumption of $\mathcal{E}_C \leq m^{\alpha-1} \sum_i \mathcal{E}_i$, where for each i , $\mathcal{E}_i = \int_t s_i(t)^\alpha dt$ is the energy consumption of the respective schedule.*

4.1 Algorithm CommonDeadlineScheduleWithPredictions (CDSwP)

At a high level $CDSwP(\lambda)$ (almost) follows the optimal schedule for the predicted instance as long as the prediction error is not higher than λ and switches to a classical online algorithm (i.e., one without predictions) in case the prediction error becomes higher than λ .

More formally, the algorithm can reside in one of two modes: *follow the prediction* (FtP) mode, and *recovery* mode. Initially, before the release time r_1 of the first job the algorithm is in the FtP-mode and has an associated speed-profile given by $s(FtP(0), t) = 0$ for all $t \in [0, d]$. Upon each release time r_i , $i = 1, \dots, n$, and while in the FtP-mode, $CDSwP(\lambda)$ does the following:

- If $\eta_i \leq \lambda$, $CDSwP$ remains in the FtP-mode and updates the speed profile from $s(FtP(i-1), t)$ to $s(FtP(i), t)$ for $[r_i, d]$ with the help of a job instance J^i . Instance J^i consists of:
 - One job i' with release time $r_{i'} = r_i$, workload $w_{i'}$ equal to the total amount of unfinished workload at r_i workload that was released at any timepoint $t \leq r_i$, and deadline d .
 - For each job j not yet released at r_j , include job j with a release time of p'_j , a deadline of d and a volume of w_j in J^i .

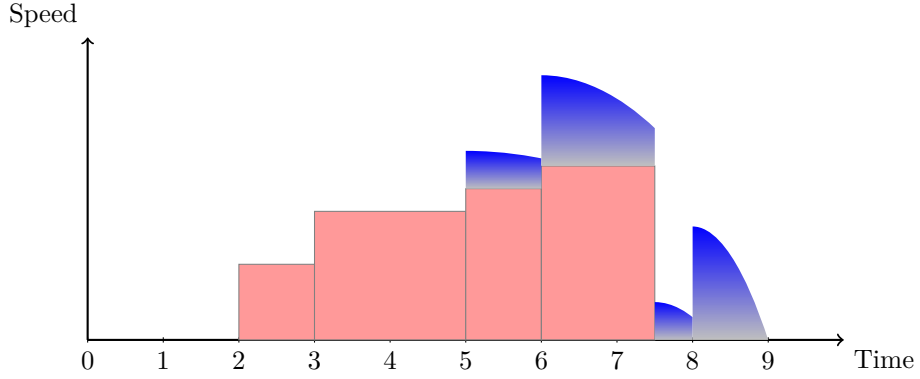
The new speed-profile $s(FtP(i), t)$ is given for any $t \in [r_i, d]$ by

$$s(FtP(i), t) := \begin{cases} s(YDS(J^i), t), & \text{if } YDS(J^i) \text{ runs job } i' \text{ at } t, \\ 0, & \text{otherwise.} \end{cases}$$

Algorithm $CDSwP$ now runs at $s(FtP(i), t)$ for any $t \in [r_i, r_{i+1})$, and remains in the FtP-mode.

- Otherwise, if $\eta_i > \lambda$ then $CDSwP$ switches to the recovery-mode, and sets $k := i$.

9:12 A Novel Prediction Setup for Online Speed-Scaling



■ **Figure 2** A common deadline instance with $\eta > \lambda$. The first time point with $\eta_i > \lambda$ is time 5 in which we start to run qOA for the rest of the jobs (blue part) while we continue running YDS for the jobs released before 5 (red part). At time point 7.5, the workload of the first set of jobs is finished.

When in recovery-mode, the algorithm runs at speed $s(t) = s(\text{FtP}(k-1), t) + s(\text{qOA}(k), t)$ at each timepoint t until d , where $s(\text{FtP}(k-1), t)$ is the last speed-profile generated in the FtP-mode, and $s(\text{qOA}(k), t)$, is the speed that the online algorithm qOA would have at timepoint t when presented (in an online fashion) with (the actual) jobs k, \dots, n .

Note that defining the speed at any timepoint t is sufficient in order to fully describe the algorithm. Indeed, since all jobs have a common deadline of d , it is irrelevant which job (among the active jobs) is being processed at any timepoint t . Nevertheless, to simplify the presentation we will implicitly assume in the following that at timepoint t the currently active and unfinished job with the earliest release time is the one being processed – and ties are broken arbitrarily. We first prove that the algorithm produces feasible schedules:

► **Observation 18.** *Algorithm CDSWP fully processes the whole processing volume of each job w_j , within $[r_j, d]$.*

Proof. Note that by the algorithm definition, no job starts being processed before its arrival in any mode. So it suffices to show that the complete processing volume of each job is completed before its deadline. Assume first that the algorithm remains in the FtP-mode until d . By the definition of the job instances J^i , any still unfinished processing volume $w_{n'}$ will be assigned to job n' at timepoint $r_{n'}$ and YDS will schedule it within $[r_{n'}, d]$ according to YDS at a speed of $w_{n'}/(d - r_{n'})$. So the resulting schedule is feasible in that case. If the algorithm switches to the recovery mode at some r_k , then by the above argument the speed profile $s(\text{FtP}(k-1), t)$ is sufficient to finish jobs $1, \dots, k-1$, and furthermore speed profile $s(\text{qOA}(k), t)$ is feasible for jobs k, \dots, n , by the feasibility of algorithm qOA . So the overall speed profile $s(\text{FtP}(k-1), t) + s(\text{qOA}(j), t)$ is sufficient for processing the whole volume. ◀

We begin by showing the following theorem which will imply consistency and smoothness.

► **Lemma 19** (Consistency & Smoothness). *Under the assumption that $\eta \in (0, \lambda)$, there holds*

$$\mathcal{E}_{\text{CDSWP}} \leq \left(\frac{1 + \eta}{1 - \lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{\text{YDS}(J_R)}.$$

Before proving Lemma 19 we show the following intermediate result.

► **Lemma 20.** *Assuming that $\eta \in (0, \lambda)$, there holds*

$$\mathcal{E}_{CDSwP} \leq \mathcal{E}_{YDS(J_{P'})}$$

Proof. Consider job instance $J^{i'}$ which consists of:

- A job j^{i-1} with release time r_i , deadline d and volume $w_{j^{i-1}} := w'_i - w_i$ equal to the total volume of jobs $1, \dots, i-1$ that is still unfinished at r_i ,
- Job i with release time at p'_i (and still deadline d and processing volume w_i),
- For each job j not yet released at r_j , include job j with a release time of p'_j , a deadline of d and a volume of w_i in $J^{i'}$.

Note that, instance $J^{i'}$ differs from J^i only in that job i is considered separately, and not together with all previously released jobs that are still not finished. By Observation 11 a YDS schedule for the former is a feasible schedule for the later, and therefore by optimality of YDS,

$$\mathcal{E}_{CDSwP(J^i)}^{[r_i, \infty)} \leq \mathcal{E}_{CDSwP(J^{i'})}^{[r_i, \infty)}, \quad (5)$$

where $\mathcal{E}_{A(J)}^{[a, b]}$, refers to the energy consumption that the schedule produced by algorithm A on instance J has within interval $[a, b]$.

Using this notation, we can express the total energy-consumption of the $CDSwP$ as

$$\begin{aligned} \mathcal{E}_{CDSwP} &= \sum_{i=1}^n \mathcal{E}_{CDSwP(J^i)}^{[r_i, r_{i+1})} \\ &= \sum_{i=1}^{n-2} \mathcal{E}_{CDSwP(J^i)}^{[r_i, r_{i+1})} + \mathcal{E}_{CDSwP(J^{n-1})}^{[r_{n-1}, r_n)} + \mathcal{E}_{CDSwP(J^n)}^{[r_n, d)} \\ &\leq \sum_{i=1}^{n-2} \mathcal{E}_{CDSwP(J^i)}^{[r_i, r_{i+1})} + \mathcal{E}_{CDSwP(J^{n-1})}^{[r_{n-1}, r_n)} + \mathcal{E}_{CDSwP(J^{n'})}^{[r_n, d)} \\ &= \sum_{i=1}^{n-2} \mathcal{E}_{CDSwP(J^i)}^{[r_i, r_{i+1})} + \mathcal{E}_{CDSwP(J^{n-1})}^{[r_{n-1}, d)} \\ &\vdots \\ &\leq \mathcal{E}_{CDSwP(J^1)}^{[r_1, d)} \leq \mathcal{E}_{YDS(J_{P'})}, \end{aligned}$$

where the inequalities follow by applying Equation (5). ◀

Proof of Lemma 19. By combining Lemmas 20 and 13 we have,

$$\mathcal{E}_{CDSwP} \leq \mathcal{E}_{YDS(J_{P'})} \leq \left(\frac{1+\eta}{1-\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)},$$

and the lemma directly follows. ◀

We note that the above proof also works in exactly the same way when only a subset A of the job set is processed.

► **Corollary 21.** *Consider a set of jobs $A \subseteq \mathcal{J}$ and assume that $\eta_i \in (0, \lambda)$ holds for every job $i \in A$. Then*

$$\mathcal{E}_{CDSwP(A)} \leq \mathcal{E}_{YDS(J_{P'}(A))}$$

We next analyze the case of inadequate predictions.

► **Lemma 22.** (*Robustness*) *With a parameter $\eta \notin (0, \lambda)$, we have*

$$\mathcal{E}_{CDSWP} \leq 2^\alpha \left(\frac{1+\lambda}{1-\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{qOA}.$$

Proof. As in the definition of CDSWP, let k be the smallest index, such that $\eta_k > \lambda$. Hence, the algorithm switches to the recovery mode at r_k . We partition the job set into two subsets $A = \{1, \dots, k-1\}$ and $B = \{k, \dots, n\}$. By Lemma 17, and by the fact that by Corollary 21 the energy consumption for set B is at most the energy consumption of qOA for the whole job instance, it suffices to upper bound the energy consumption required for set A by the total energy that $qOA(k)$ uses.

We transform the schedule obtained by CDSWP for job set A through three intermediate steps to the schedule produced by $YDS^J(R)$. Since $\mathcal{E}_{YDS^J(R)} \leq \mathcal{E}_{qOA^J(R)}$ this will imply the theorem.

Step 1. Let J^A be the job instance that contains all jobs in A , along with jobs $j = k, k+1, \dots, n$ with respective release time p'_j , deadline d and processing volume w_j .

Let \mathcal{E}_{CDSWP}^A , and $\mathcal{E}_{YDS(J^A)}^A$ be the energy consumptions incurred while scheduling the subset of jobs A for CDSWP(J) and $YDS(J^A)$ respectively. By Corollary 21,

$$\mathcal{E}_{CDSWP}^A \leq \mathcal{E}_{YDS(J_{P'})}^A.$$

Let J_P^A be the job instance, consisting of the predicted release times (p_i) of jobs in set A and the “shrunk” predicted release times (p'_i) for the remaining jobs. Note that J_P^A differs from J^A only in the release-times of jobs in set A . Since $\eta_i \leq \lambda$ for any $i \in A$, there holds for any such i that $d - p'_i = 1/(1-\lambda)(d - p_i)$. By Lemma 6, there therefore holds

$$\mathcal{E}_{YDS(J_{P'})}^A \leq \left(\frac{1}{1-\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_P^A)}^A.$$

Consider set $P^* = \{p_1^*, \dots, p_{k-1}^*, p'_k, \dots, p'_n\}$ with $p_i^* = p_i - \eta_i(q_i - p_i)$ for all $j \in [k-1]$. There holds

$$\mathcal{E}_{YDS(J_P^A)} \leq (1+\lambda)^{\alpha-1} \mathcal{E}_{YDS(J_{P^*})} \leq (1+\lambda)^{\alpha-1} \mathcal{E}_{YDS(J^A)}. \quad (6)$$

By having $c = \frac{1}{(1+\lambda)}$, and $J = J_{P^*}$ ($r_i = p_i^*$) in Lemma 6, we obtain $J' = J_P$. Since we have $\eta_j < \lambda$ for all $j < k$, the first inequality in (6) holds. For every job $i \in A$ there holds $(r_i, d) \subseteq (p_i^*, d)$. More specifically, a feasible schedule for J^A is feasible for J_{P^*} as well. The second inequality in (6) then directly follows by the optimality of YDS .

Putting things together we therefore have

$$\mathcal{E}_{CDSWP}^A \leq \left(\frac{1+\lambda}{1-\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J^A)}^A, \quad (7)$$

Step 2. In this step, we want to compare $\mathcal{E}_{YDS(J^A)}^A$ with the energy of YDS algorithm for a new job instance in which we consider the real release times for some jobs in set B that their shrinking predictions are after their real release times.

A job instance J^I is defined, consisting of the real release times of jobs in set A , the real release times of job j in set B for which $r_j \leq p'_j$, and the shrunk prediction (p'_j) for the rest. Since moving the release times of the future jobs to the left could increase the speed (and hence increases energy) in the first part,

$$\mathcal{E}_{YDS(J^A)}^A \leq \mathcal{E}_{YDS(J^I)}^A.$$

Step 3. In the last step, we want to compare $\mathcal{E}_{YDS(J^l)}^A$ with the optimum offline algorithm (YDS) for the complete job instance J and their real release time J_R . We want to show

$$\mathcal{E}_{YDS(J^l)}^A \leq \mathcal{E}_{YDS(J^l)}^J \leq \mathcal{E}_{YDS(J_R)}^J.$$

The first inequality holds because $A \subseteq J$. Consider the difference between two job instances J_R and J^l . Since for each job i , its available time in J_R is a subset of its available time in J^l , $YDS(J_R)$ is a feasible algorithm for job instance J^l . Therefore, the second inequality holds.

So far we proved that

$$\mathcal{E}_{CDSwP}^A \leq \left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}^J.$$

Since we run qOA for the job set B ,

$$\mathcal{E}_{CDSwP}^B = \mathcal{E}_{qOA(J_R)}^B \leq \mathcal{E}_{qOA(J_R)}^J.$$

And by Lemma 17,

$$\mathcal{E}_{CDSwP} \leq 2^{\alpha-1} \cdot \left(\left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}^J + \mathcal{E}_{qOA(J_R)}^J\right).$$

Since $\mathcal{E}_{YDS(J_R)}^J \leq \mathcal{E}_{qOA(J_R)}^J$,

$$\mathcal{E}_{CDSwP} \leq 2^{\alpha-1} \cdot (\mathcal{E}_{qOA}) \left(\left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} + 1\right) \leq 2^\alpha \left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot (\mathcal{E}_{qOA}). \quad \blacktriangleleft$$

Lemmas 19 and 22 together imply Theorem 2.

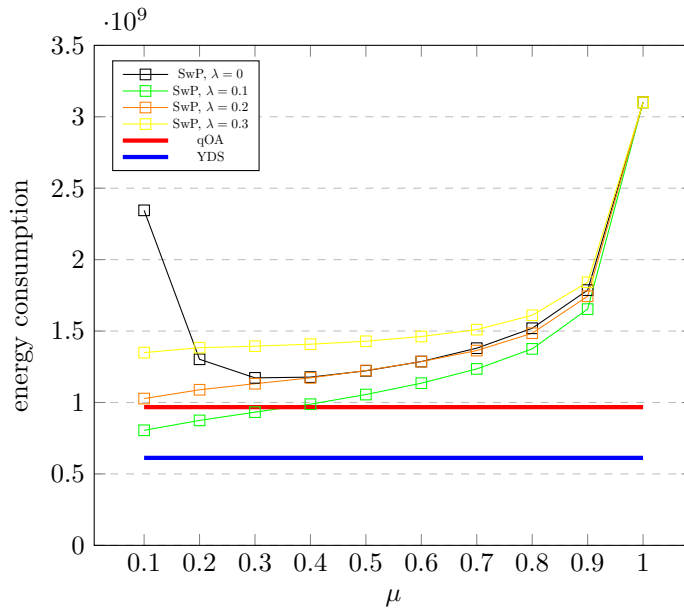
5 Discussion on Confidence Parameters λ and μ

In order to give some intuition on how the confidence parameters μ and λ affect the obtained performance guarantees of SWP, we perform some numerical experiments for different settings. Moreover, we compare our algorithm with the currently best-known online algorithm qOA and the optimum offline algorithm YDS using real-world data. All experiments were run on a typical laptop computer.

We only consider $\alpha = 3$ for the experiments, as this is the typical value of α for real-world processors, see for example [15, 31]. Furthermore for qOA , we only consider $q = 2 - \frac{1}{\alpha} \approx 1.667$ since this is the value that minimizes the competitive ratio [11].

The input data for our experiments is the same as in [1]. There, jobs are generated from http requests received on EPAs web-server. For practical reasons, we limit our input instances to the first 1000 jobs of their sample. In order to generate predictions for the input, we use a normal distribution with a mean of 0, and a standard deviation of 0.01, 0.05, or 0.1. For each job, two samples from this distribution are taken and each of them is scaled by the real interval length of the job. The result is then added to each job's actual release time and deadline to obtain predictions for them.

In order to illustrate the effect of parameters λ and μ , we run SWP for different combinations of these values. In particular we consider $\lambda = 0, 0.1, 0.2, 0.3$ and $\mu = 0.1, 0.2, \dots, 1$. Our results with standard deviation 0.05 can be found in Figure 3. Our results with standard deviations 0.01, and 0.1 can be found in the supplementary material.



■ **Figure 3** Prediction set 2, stddev=0.05.

To gain some intuition on the results, recall that μ denotes the portion of each block for which AVR is run. In particular, for $\mu = 1$ the SwP algorithm becomes identical to the AVR algorithm and disregards the predictions, whereas the smaller μ 's value the more the predictions are trusted. This explains why the competitive ratio increases with μ . Similarly, recall that λ defines how much the predicted interval will be shrunk and that the improved competitive ratio is only proven for $\eta \leq \lambda$ but on the other hand the bigger λ gets the smaller that improvement in the competitive ratio will be. Although the best choices for λ and μ depend on the quality and/or structure of the predictions, our experiments highlight that for appropriate such choices, one can significantly improve upon the energy-consumption of qOA. To summarize, in practice the most sensible settings of λ and μ will depend on the quality as well as structure of the predictions and it may be worthwhile experimenting with different such settings.

6 Conclusion

In this paper, we have presented a consistent, smooth, and robust algorithm for the general classical, deadline-based, online speed-scaling problem using ML predictions for release times and deadlines.

We can remove the assumption of knowing the number of jobs n , by slightly adapting the error definition, so that the prediction is considered to be inadequate if the predicted number of jobs is wrong.

It remains an interesting open question on whether a similar robust, consistent and smooth algorithm exists for the more general setup in which the workloads of the jobs are not known in advance but predicted along with their release times and deadlines. Although we were able to extend SWP under the assumption that it satisfies a natural monotonicity property, it is unclear if that property holds in general.

References

- 1 Ahmed Abousamra, David P. Bunde, and Kirk Pruhs. An experimental comparison of speed scaling algorithms with deadline feasibility constraints. *CoRR*, abs/1307.0531, 2013. [arXiv:1307.0531](#).
- 2 Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.
- 3 Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration. *J. Comput. Syst. Sci.*, 81(7):1194–1209, 2015.
- 4 Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3(4):49, 2007.
- 5 Eric Angel, Evripidis Bampis, Fadi Kacem, and Dimitrios Letsios. Speed scaling on parallel processors with migration. *J. Comb. Optim.*, 37(4):1266–1282, 2019.
- 6 Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, pages 345–355. PMLR, 2020.
- 7 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.
- 8 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.
- 9 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.
- 10 Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.
- 11 Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory Comput.*, 8(1):209–229, 2012.
- 12 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), March 2007.
- 13 Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.
- 14 Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.
- 15 Dale L. Critchlow, Robert H. Dennard, and Stanley Schuster. Design and characteristics of n-channel insulated-gate field-effect transistors. *IBM J. Res. Dev.*, 44(1):70–83, 2000.
- 16 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In *EC*, pages 409–429. ACM, 2021.
- 17 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994.
- 18 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, 2019.
- 19 Mark Herbster and Manfred K Warmuth. Tracking the best expert. *Machine learning*, 32(2):151–178, 1998.
- 20 Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
- 21 Nicola Jones. How to stop data centers from gobbling up the world’s electricity, 2018. [Online; accessed 02-August-2021].
- 22 N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. doi:10.1006/inco.1994.1009.
- 23 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.
- 24 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, volume 151 of *LIPICs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

- 25 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.
- 26 Benjamin Moseley, Sergei Vassilvitskii, Silvio Lattanzi, and Thomas Lavastida. Online scheduling via learned weights. In *SODA*, 2020.
- 27 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- 28 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, 2020.
- 29 Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *AAMAS*, pages 2035–2037. International Foundation for Autonomous Agents and Multiagent Systems, 2020.
- 30 Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, volume 176 of *LIPICs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 31 Adam Wierman, Lachlan L. H. Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *Perform. Evaluation*, 69(12):601–622, 2012.
- 32 F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.

A Calculating the y_i^t 's

First we show the following lemma.

► **Lemma 23.** *For any given $0 \leq X \leq \ell(j) \max_{t \in [r_j, d_j]} (V_t(j) + \delta_j) / \mu$ there exist values y_j^t , with $0 \leq y_j^t \leq \delta_j$ so that equations (3), (4) and (2) are satisfied for all $t \in [r_j, d_j]$, with X in place of X_j . Furthermore for any t, t' with $y_j^t \leq y_j^{t'}$ there holds $V_{t'}(j) \leq V_t(j)$, and $\sum y_j^t$ is a continuous and non-decreasing function in X .*

Proof. If $X/\ell(j) < \min_{t \in [r_j, d_j]} V_t(j)/\mu$, then it is easy to verify that $y_j^t = \delta_j$ for all $t \in [r_j, d_j]$ satisfies all equations. So we assume for the remainder of this proof that $\min_{t \in [r_j, d_j]} V_t(j)/\mu \leq X/\ell(j) \leq \max_{t \in [r_j, d_j]} (V_t(j) + \delta_j)/\mu$.

For any $t \in [r_j, d_j]$, let

$$y_j^t := \begin{cases} 0, & \text{if } V_t(j)/\mu \geq X/\ell(j), \\ \delta_j, & \text{if } (V_t(j) + \delta_j)/\mu \leq X/\ell(j), \\ \mu X/\ell(j) - V_t(j), & \text{otherwise.} \end{cases} \quad (8)$$

It is easy to verify that for the above definition of y_j^t , equations (3), (4) and (2) are satisfied with X in place of X_j , and that for any t, t' with $y_j^t \leq y_j^{t'}$ there holds $V_{t'}(j) \leq V_t(j)$. Finally, $\sum y_j^t$ is a continuous function as a sum of a finite number of continuous functions, and non-decreasing in X (as each y_j^t is by definition a non-increasing function of X). ◀

► **Lemma 24.** *For any set of values $V_t(j)$, there exist values y_j^t , with $0 \leq y_j^t \leq \delta_j$ so that equations (3), (4) and (2) are satisfied for all $t \in [r_j, d_j]$.*

Proof. Note that it suffices to show that there exists $X_j = w_j - \sum_t y_j^t$ where the y_j^t are as defined in the proof of Lemma 23, since then by Lemma 23 the equations (3), (4), and (2) would hold for $X_j = w_j - \sum_t y_j^t$.

First, let $X = w_j$ and compute the values of y_j^t via (8). If $\sum_t y_j^t = 0$, then we have found the desired X and are done. Assume therefore, that $0 < \sum_t y_j^t \leq w_j$. By Lemma 23, $\sum_t y_j^t$ is

a non-decreasing and continuous function of X within $[0, w_j]$ that obtains value 0 for $X = 0$, and a value $\leq w_j$ for $X = w_j$. Equivalently the function $w_j - \sum_t y_j^t$ is non-increasing and continuous in X within $[0, w_j]$ and obtains value w_j for $X = 0$ and a value ≥ 0 for $X = w_j$. Therefore, by the intermediate value theorem there must exist an $X_j \in [0, w_j]$, such that $w_j - \sum_t y_j^t$ obtains a value of X_j , which concludes the proof of the lemma. ◀

Algorithm

Lemmas 23 and 24 directly imply an algorithm for identifying such values of y_j^t . In particular, since for any t, t' with $y_j^t \leq y_j^{t'}$ there holds $V_{t'}(j) \leq V_t(j)$, we can order all relevant t 's by $V_t(j)$ and find (through enumeration) t', t'' such that for any $V_t(j) \geq V_{t'}(j)$ we have $y_j^t = 0$, for any $V_t(j) \leq V_{t''}(j)$, $y_j^t = \delta_j$ and for all other t there holds $0 < y_j^t < \delta_j$. Let N be the number of t 's such that $y_j^t = \delta_j$, and $Z = w_j - N\delta_j$ be the remaining processing volume that needs to be assigned through the y_j^t 's for t 's with $V_{t''}(j) < V_t(j) < V_{t'}(j)$. In other words we need to find $0 < y_j^t < \delta_j$ so that $Z - \sum_t y_j^t = X_j$, and for each individual such y_j^t , we have $y_j^t = \mu X_j / \ell(j) - V_t(j)$. This implies a system of $k + 1$ equations (for some k) with $k + 1$ unknowns, that by Lemma 24 has a solution assuming that t', t'' were chosen correctly.

B Missing Plots of Section 5

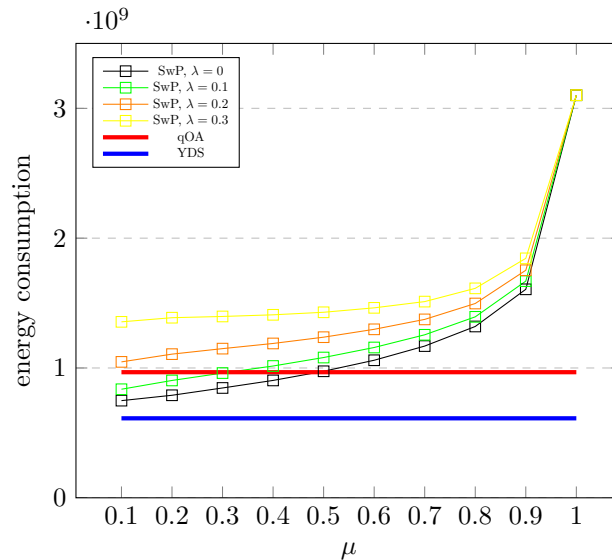
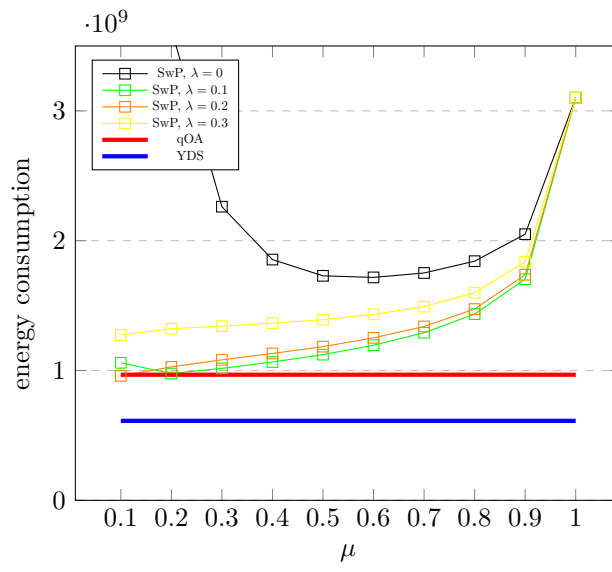


Figure 4 Prediction set 1, stddev=0.01.



■ Figure 5 Prediction set 3, stddev=0.1.

On the Approximability of the Traveling Salesman Problem with Line Neighborhoods

Antonios Antoniadis ✉

University of Twente, Enschede, The Netherlands

Sándor Kisfaludi-Bak ✉

Aalto University, Finland

Bundit Laekhanukit ✉

Shanghai University of Finance and Economics, China

Daniel Vaz ✉

Operations Research Group, Technische Universität München, Germany

Abstract

We study the variant of the Euclidean Traveling Salesman problem where instead of a set of points, we are given a set of lines as input, and the goal is to find the shortest tour that visits each line. The best known upper and lower bounds for the problem in \mathbb{R}^d , with $d \geq 3$, are NP-hardness and an $O(\log^3 n)$ -approximation algorithm which is based on a reduction to the group Steiner tree problem.

We show that TSP with lines in \mathbb{R}^d is APX-hard for any $d \geq 3$. More generally, this implies that TSP with k -dimensional flats does not admit a PTAS for any $1 \leq k \leq d - 2$ unless $P = NP$, which gives a complete classification regarding the existence of polynomial time approximation schemes for these problems, as there are known PTASes for $k = 0$ (i.e., points) and $k = d - 1$ (hyperplanes). We are able to give a stronger inapproximability factor for $d = O(\log n)$ by showing that TSP with lines does not admit a $(2 - \varepsilon)$ -approximation in d dimensions under the Unique Games Conjecture. On the positive side, we leverage recent results on restricted variants of the group Steiner tree problem in order to give an $O(\log^2 n)$ -approximation algorithm for the problem, albeit with a running time of $n^{O(\log \log n)}$.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Traveling Salesman with neighborhoods, Group Steiner Tree, Geometric approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.10

Related Version The article has an earlier version available on ArXiv.

Full Version: <https://arxiv.org/abs/2008.12075>

Funding *Antonios Antoniadis:* Work done in part while the author was at Saarland University and Max Planck Institute for Informatics and supported by DFG grant AN 1262/1-1.

Daniel Vaz: This work has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF). Work done in part while the author was at Saarland University and Max Planck Institute for Informatics.

1 Introduction

In the Euclidean Traveling Salesman problem, one is given n points in d -dimensional Euclidean space (denoted by \mathbb{R}^d), and the goal is to find the shortest tour visiting all the points. The problem is NP-hard for $d \geq 2$ [41], but it has a celebrated polynomial time approximation scheme (PTAS), i.e., a polynomial-time algorithm that produces a tour of length at most $1 + \varepsilon$ times the optimum for any fixed $\varepsilon > 0$, due to Arora [3] and (independently) by Mitchell [38].



© Antonios Antoniadis, Sándor Kisfaludi-Bak, Bundit Laekhanukit, and Daniel Vaz; licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 10; pp. 10:1–10:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the past decades, a considerable amount of work has concentrated on finding approximations for variants and generalizations of the Euclidean Traveling Salesman Problem, e.g., by changing the underlying space [4, 33, 16, 6], or the objects being visited [15, 7, 11, 20, 39, 40, 28]. In the latter case which is known as the *Traveling Salesman Problem with Neighborhoods (TSPN)*, the input consists of n neighborhoods, and the goal is to find the shortest tour that visits each neighborhood. More formally, we are given the sets $S_1, \dots, S_n \subset \mathbb{R}^d$, and we wish to compute the shortest closed curve τ such that for each $i \in \{1, \dots, n\}$ we have $S_i \cap \tau \neq \emptyset$. (Observe that the optimum curve τ consists of at most n segments.)

In contrast to regular TSP, TSPN is already APX-hard in the Euclidean plane [15], i.e., it has no PTAS unless $P = NP$. Worse still, even the basic case in which each neighborhood is an arbitrary finite set of points in the Euclidean plane (the so called *Group TSP*) admits no polynomial-time $O(1)$ -approximation (unless $P = NP$) [43]. Even in the case in which each neighborhood consists of exactly two points [18] the problem remains APX-hard.

This inherent hardness of TSPN gives rise to studying variants of the problem in which the neighborhoods are restricted in some ways. In a seminal paper, Arkin and Hassin [2] looked into the problem for various cases of *bounded neighborhoods*, including translates of convex regions and parallel unit segments, and gave constant-factor approximation algorithms for them. The best known approximation algorithm for a more general case of bounded neighborhoods in the plane is due to Mata and Mitchell [35] and attains an $O(\log n)$ approximation factor. However, there exist special cases of such bounded neighborhoods in the plane that do allow for $O(1)$ -approximation algorithms. These include neighborhoods which are disjoint, fat, or have comparable sizes [15, 7, 11, 20, 39, 40].

The complementary case of TSPN where neighborhoods are *unbounded* regions (the focus of this paper) is, in general, less well understood. Consider neighborhoods that are affine subspaces (*flats*) of dimension $k < d$ in \mathbb{R}^d . On the positive side, and despite the APX-hardness of the general TSPN problem already in \mathbb{R}^2 , the version with flats (in this case lines) as neighborhoods can be solved exactly in $O(n^4 \log n)$ -time via a reduction to the watchman route problem [29, 17]. Furthermore, Dumitrescu [19] provides a 1.28-approximation algorithm that runs in linear time. In \mathbb{R}^3 , the problem of line and plane neighborhoods was first raised by Dumitrescu and Mitchell [20]. For the line case, they already point out that the problem is NP-hard as a direct consequence of the NP-hardness of Euclidean TSP in the plane [41]. Although this leaves the possibility for a PTAS open, the best known approximation algorithm to date for TSPN with lines in \mathbb{R}^3 was given by Dumitrescu and Tóth [21] and achieves an $O(\log^3 n)$ -approximation. For the case of $(d-1)$ -dimensional flats in \mathbb{R}^d (which also includes planes in \mathbb{R}^3), they give a linear-time (for any constant dimension d and any constant $\varepsilon > 0$) $(1 + \varepsilon)2^{d-1}/\sqrt{d}$ -approximation. This result was subsequently improved by Antoniadis et al. [1] to an EPTAS that also runs in linear time for fixed d and ε . Whether this variant is NP-hard or not remains an interesting open problem. As for the case of line neighborhoods in \mathbb{R}^d for $d \geq 3$, a PTAS for k -dimensional flats for $1 \leq k \leq d-2$ also remained out of reach.

We show that unless $P = NP$, there is no PTAS for lines in \mathbb{R}^3 . As a direct consequence, we can rule out the existence of a PTAS in all remaining open cases of TSPN with flats: there is no PTAS for k -dimensional flat neighborhoods for any $1 \leq k \leq d-2$, unless $P = NP$.

Let us call the Euclidean TSP problem in \mathbb{R}^d with k -dimensional flat neighborhoods (k, d) -TSPN. Although ruling out a PTAS for $(1, 3)$ -TSPN is an important step towards settling the approximability of the problem, the inapproximability factor obtained is very close to 1. It would be desirable to obtain a stronger inapproximability factor, especially given how far we are from any constant-approximation algorithm for the problem. A natural

way to obtain such a stronger inapproximability result is to consider the problem in higher dimensional spaces. For example, regarding the classic Euclidean TSP, it is known that the problem becomes APX-hard for $d = \log n$ [44]. This result directly implies that TSPN with line neighborhoods in $\mathbb{R}^{1+\log n}$ is APX-hard, but this is barely satisfactory, since it again only gives a small inapproximability factor. However, by using a different reduction from the vertex cover problem, we are able to show that the problem has no polynomial $(2 - \varepsilon)$ -approximation in $\mathbb{R}^{O(\log n)}$ for any fixed $\varepsilon > 0$ under the Unique Games Conjecture [30].

On the algorithmic side, very little is known about (k, d) -TSPN. For $d = 3$, the best known polynomial time approximation for $(1, 3)$ -TSPN is the aforementioned $O(\log^3 n)$ -approximation algorithm due to Dumitrescu and Tóth [21]. Their approach is to discretize the problem by selecting a polynomial number of “relevant” points on each line. It is shown that restricting the solution to visiting lines at these points only increases the tour length by a constant factor. The resulting instance can now be seen as an instance of group-TSP, where the relevant points of each line form a group. By feeding this into the $O(\log^3 n)$ -approximation algorithm for general group Steiner tree [25, 24] (it is easy to go from the tree solution to a tour by doubling each edge), they obtain the same asymptotic approximation factor for TSPN with line neighborhoods. This is somewhat unsatisfactory, since it ignores that the group Steiner tree instances constructed by the reduction are (i) Euclidean and (ii) all the points of a group are collinear. In other words, although the constructed group Steiner tree instances are highly restricted, there is no known technique to exploit this restriction.

However, the reduction from TSPN with line neighborhoods to the group Steiner tree problem implies that, if we allow quasi-polynomial running time, then TSPN with line neighborhoods admits an approximation ratio of $O(\log^2 n / \log \log n)$ in $O(n^{\log^2 n})$ time [13]. This approximation ratio of group Steiner tree is tight for the class of quasi-polynomial time algorithms due to the recent work of Grandoni et al. [26], which holds under the *Projection Game Conjecture* and $\text{NP} \not\subseteq \bigcap_{0 < \epsilon < 1} \text{ZPTIME}(2^{n^\epsilon})$. Their hardness result is built on the seminal work of Halperin and Krauthgamer [27], who prove that group Steiner tree admits no $\log^{2-\epsilon} n$ -approximation for any fixed $\epsilon > 0$, unless $\text{NP} \subseteq \text{ZTIME}(n^{\text{poly}(\log n)})$.

For the class of polynomial-time approximation algorithms, the group Steiner tree problem admits an approximation ratio of $O(\log^2 n)$ on some special cases, e.g., trees [25] and bounded treewidth graphs [10, 9]. It is still open whether the group Steiner tree problem in general graphs admits a polynomial-time $O(\log^2 n)$ -approximation algorithm; the best running time to obtain an $O(\log^2 n)$ -approximation is $n^{O(\log n)}$ [13].

The connection between TSPN and group Steiner tree also holds in the reverse direction: Given an instance of group Steiner tree, one may embed the input metric into a Euclidean space with distortion $O(\log n)$ [8] and cast it as TSPN with “set neighborhoods”.

While we cannot improve the approximation factor in polynomial time, we can do so in quasi-polynomial time: we give an $O(\log^2 n)$ -approximation in $n^{O(\log \log n)}$ time. We obtain this result by using Arora’s PTAS for TSP [3], together with the framework of Chalermsook et al. [10, 9], to transform TSPN into a variant of group Steiner tree when the input graph is a tree, and then employing an $O(\log^2 n)$ -approximation algorithm.

Our Contribution. Our first contribution is to show that unlike the problem with hyperplane neighborhoods, the problem with line neighborhoods is APX-hard.

► **Theorem 1.** *The TSPN problem for lines in \mathbb{R}^3 is APX-hard. More specifically, it has no polynomial time $(1 + \frac{1}{230000})$ -approximation unless $\text{P} = \text{NP}$.*

The reduction is from the vertex cover problem on tripartite graphs. The idea is to represent the graph edges with lines, where two lines intersect if and only if they correspond to incident edges. The main challenge is to keep the pairwise distance between non-intersecting

10:4 On the Approximability of TSP with Line Neighborhoods

lines large enough. We solve this by carefully placing the intersection points on non-adjacent edges of a cube. For technical reasons, we do not work directly with this placement, but rather on a “flattened” version of this point set. Additionally, we want to restrict the optimal tour so that it visits each line near one of its intersection points with other lines. This is achieved by forcing the optimal tour to follow a certain closed curve using special point gadgets (each consists of polynomially many lines), and to visit the lines representing the edges only at (or close to) intersection points. Visiting an intersection point corresponds to including the corresponding vertex in the vertex cover of the graph. As a direct consequence of Theorem 1, we obtain the following.

► **Corollary 2.** *The Euclidean TSP problem with k -dimensional flat neighborhoods in \mathbb{R}^d is APX-hard for all $1 \leq k \leq d - 2$.*

To prove Corollary 2, suppose we are given a set \mathcal{L} of lines in \mathbb{R}^3 . We can first change each line $\ell \in \mathcal{L}$ into the flat $\ell \times \mathbb{R}^{k-1}$, resulting in k -dimensional flats in \mathbb{R}^{k+2} . Since $k \leq d - 2$, we have that \mathbb{R}^{k+2} is a subspace of \mathbb{R}^d , so this is a valid construction for (k, d) -TSPN. Moreover, any tour in \mathbb{R}^3 visiting the lines is also a valid tour of the k -flats, and a valid tour of the k -flats can be projected into a valid tour of \mathcal{L} in \mathbb{R}^3 of less or equal length.

Our second contribution is to show a larger inapproximability factor in higher dimensions under the Unique Games Conjecture:

► **Theorem 3.** *For any $\varepsilon > 0$, there exists a constant c such that there is no $(2 - \varepsilon)$ -approximation algorithm for TSPN with line neighborhoods in $\mathbb{R}^{c \cdot \log n}$, unless the Unique Games Conjecture is false. Moreover, for any $\varepsilon > 0$, there is a constant c such that it is NP-hard to give a $(\sqrt{2} - \varepsilon)$ -approximation for TSPN with line neighborhoods in $\mathbb{R}^{c \cdot \log n}$.*

This reduction is from the general vertex cover problem. Again we represent the edges of the graph with lines and the vertices correspond to intersection points. This time however the intersection points are almost equidistant: they are obtained via the Johnson-Lindenstrauss lemma applied on an n -simplex. This allows the tour to visit the intersection points in any order. To obtain a direct correspondence with vertex cover, we need to ensure that lines are visited near intersection points. To this end, we blow up the underlying graph by replacing each edge by a complete bipartite graph. Thus, we get the following corollary of Theorem 3.

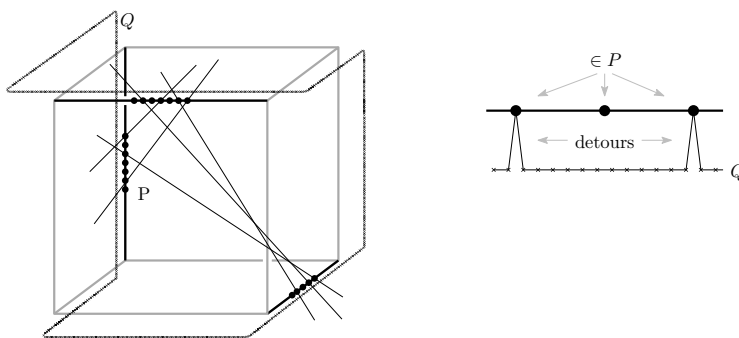
► **Corollary 4.** *For any $\varepsilon > 0$ there is a number $c = c(\varepsilon)$ such that the Euclidean TSP problem with k -dimensional flat neighborhoods in \mathbb{R}^d has no polynomial $(2 - \varepsilon)$ -approximation for any $k \in \{1, \dots, d - c \log n\}$, unless the Unique Games Conjecture is false.*

On the positive side, our third contribution is to develop an $O(\log^2 n)$ -approximation algorithm with slightly superpolynomial running time.

► **Theorem 5.** *There is a deterministic $O(\log^2 n)$ -approximation algorithm for TSPN with line neighborhoods in \mathbb{R}^d that runs in time $n^{O(\log \log n)}$ for any fixed dimension d .*

The algorithm is based on adapting the dynamic program by Arora [3], and reformulating TSPN into the problem of finding a solution in the dynamic programming space that visits all the line neighborhoods. We then build upon the techniques of Chalermsook et al. [10, 9], and show that this task can be reduced to a variant of the group Steiner tree problem that admits an $O(\log^2 n)$ -approximation in slightly superpolynomial running time. The $O(\log \log n)$ -factor in the exponent of the running time is a consequence of the running time of Arora’s algorithm, and it is possible that we can improve it to polynomial time if an appropriate EPTAS for TSP with running time $O(f(\varepsilon, d)n \log n)$ is discovered.

Due to space constraints, missing proofs are deferred to the full version of the paper.



■ **Figure 1** Left: Overview of a basic construction with a cube. Right: The optimal tour must visit all points of Q , and it makes detours to some points p_v .

2 Inapproximability in 3 dimensions

The goal of this section is to prove Theorem 1. The overall setup of our construction is lightly inspired by a reduction in Elbassioni et al. [22]. The reduction is from vertex cover on 3-partite graphs (i.e., on graphs G where the vertices can be partitioned into three independent sets V_1, V_2 and V_3). It is NP-hard to decide whether a given instance has a vertex cover of size $n/2$ or if all vertex covers have size at least $\frac{34}{33} \frac{n}{2}$ [14]. In our construction, each vertex v of G is assigned to a point p_v on an edge of a unit cube; the classes V_1, V_2, V_3 are mapped to pairwise non-adjacent and non-parallel (i.e., skew) cube edges. For each edge $uv \in E(G)$, we add the line $p_u p_v$; see Fig. 1.

Consider now a closed curve γ of length 10 which is disjoint from the cube, but follows some edges of the cube at a distance c/n for some constant c . Let Q be a set of points along γ such that any two consecutive points have distance $c/(10n)$.

We define a special *point gadget* – which consists of a large collection of lines – at each point $q \in Q$. This ensures that any TSP tour that has length at most 20 will touch an infinitesimally small ball around each vertex of Q . Consequently, any not too long TSP tour will have to “trace” γ . The points in $P = \cup p_v$ which are placed near the cube edges are arranged so that one can visit each point p_v with a short detour from γ of length c/n . Given a vertex cover of size k in G one can create a TSP tour of length at most $10 + kc/n$, namely by following γ and making the short detour at p_v if and only if v is in the vertex cover. Conversely, by a careful arrangement of the lines and point gadgets, we can ensure that a tour of length $10 + kc/n$ implies the existence of a vertex cover of size at most $1.011k$.

For technical reasons, we need to transform the constructed cube to a very flat parallelepiped; it is convenient to define the point set Q and the point gadgets only after this flattening transformation takes place. We are now ready to define our construction.

2.1 The construction

Let $G = (V, E)$ be a tripartite graph on n vertices with partition classes V_1, V_2, V_3 . We add dummy vertices (without any incident edges) to G so that each class has n vertices; the vertices of V_a ($a = 1, 2, 3$) are denoted by v_1^a, \dots, v_n^a . Notice that the addition of dummy vertices does not change the set of vertex covers of G . Let \mathcal{C} denote the unit cube $[0, 1]^3$, and let e^1, e^2, e^3 be the unit segments $(0, 0, 1)^T(1, 0, 1)^T$, $(1, 0, 0)^T(1, 1, 0)^T$ and $(0, 1, 0)^T(0, 1, 1)^T$ respectively. We assign each vertex v_i^a to a point on the middle third of e^i . The assignment is denoted by p , and defined as:

$$p(v_i^a) = \begin{cases} (\frac{n+i}{3n}, 0, 1)^T & \text{if } a = 1 \\ (1, \frac{n+i}{3n}, 0)^T & \text{if } a = 2 \\ (0, 1, \frac{n+i}{3n})^T & \text{if } a = 3. \end{cases}$$

We denote by $P = p(V(G))$ the set of points created this way. For each edge $uv \in E(G)$, let $\ell(uv)$ be the line through $p(u)$ and $p(v)$, and let \mathcal{L} be the set of lines created this way: $\mathcal{L} = \{\ell(uv) \mid uv \in E(G)\}$. The following technical lemma plays a key role in the construction.

► **Lemma 6.** *If $\ell, \ell' \in \mathcal{L}$ correspond to non-incident edges, then they are disjoint and the distance between them is at least $\frac{1}{20n}$.*

Flattening. We must ensure that the points of Q near a cube edge are similarly distant from the lines in \mathcal{L} incident to the cube edge (we will do this in Claim 9). We achieve this by transforming the above construction so that the angle of each line ℓ with the plane $x + y + z = 0$ is at most some small constant. Practically, we transform the point set P and the line set \mathcal{L} with the linear transformation $x \mapsto Ax$, where $A = I - 0.3J$ and J is the all-ones matrix.

Essentially, the transformation pushes everything closer to the plane $H : x + y + z = 0$: for a given point p and its perpendicular projection q on H , the point Ap is the point on the segment pq for which $\text{dist}(q, Ap) = \frac{1}{10} \text{dist}(q, p)$. Note that if pq is any segment of length λ , then its length after the transformation is at least $\lambda/10$ and at most λ . When the transformation is applied to an edge e^a of the cube \mathcal{C} , then the resulting segment has length $\sigma = \sqrt{0.7^2 + 0.3^2 + 0.3^2} \simeq 0.8185$. Consequently, $Ap(v_i^a)$ and $Ap(v_{i+1}^a)$ has distance $\sigma/(3n)$.

Let \bar{P} and $\bar{\mathcal{L}}$ be the resulting point set and line set. Using Lemma 6 and the above arguments we get the following corollary.

► **Corollary 7.** *The minimum distance between points of \bar{P} is $\frac{\sigma}{3n}$, and the minimum distance between lines of $\bar{\mathcal{L}}$ corresponding to non-incident edges of G is at least $\frac{1}{200n}$.*

Defining the point gadgets, and wrapping up the construction. For a point set X , let \bar{X} denote its image under the flattening transformation A . Let F_1^a and F_2^a be the planes of the faces of $[0, 1]^3$ incident to e^a . The following claim shows that \bar{F}_1^a and \bar{F}_2^a are two planes through \bar{e}^a whose angle is small.

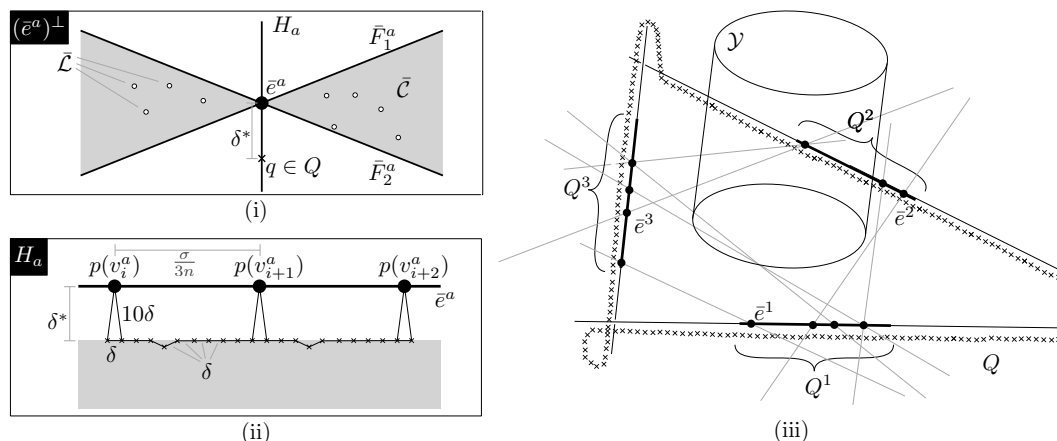
▷ **Claim 8.** For $a = 1, 2, 3$ we have $\angle(\bar{F}_1^a, \bar{F}_2^a) < \frac{1}{4}$.

Let H^a be the angle bisector plane of \bar{F}_1^a and \bar{F}_2^a which does not intersect the image of \mathcal{C} , see Figure 2(i). Within H^a , we place a set of points Q^a , which we define next.

Let $\delta = \frac{1}{4000n}$, and let δ^* be the height of the isocles triangle T_δ with base δ and two sides of length 10δ , that is $\delta^* = \sqrt{99.75}\delta$. Consider a half-plane in H_a whose boundary is parallel to \bar{e}^a and is at distance δ^* from it. Within this half-plane, let Q_a be a set of at most $4000n$ points with the following properties: (i) for each $p(v_i^a)$ there are two points $q, q' \in Q^a$ such that $p(v_i^a)$, q and q' form an isocles triangle of side lengths $10\delta, 10\delta, \delta$ and (ii) there is a unique shortest TSP path of Q_a , whose edges are of length exactly δ ; see Figure 2(ii).

Let Q be a point set with the following properties:

- $Q^1 \cup Q^2 \cup Q^3 \subseteq Q$
- For any pair of distinct points $q, q' \in Q$, $\text{dist}(q, q') \geq \delta$.
- Each segment of the minimum TSP tour $T(Q)$ of Q has length δ , and $\text{cost}(T(Q)) = 10$.
- The minimum distance of points of Q from $\bar{\mathcal{L}}$ is attained only in $Q^1 \cup Q^2 \cup Q^3$
- Q is disjoint from the cylinder \mathcal{Y} of axis $(0, 0, 0)^T(1, 1, 1)^T$ and radius $\sigma/2$.



■ **Figure 2** (i) Cross-section given by a plane perpendicular to \bar{e}^a . (The segment \bar{e}^a appears as a point, and the plane H_a as a line in this picture.) All lines of $\bar{\mathcal{L}}$ intersect such a plane in the gray area. (ii) Defining Q^a within the plane H_a , so that all points have distance at least δ^* from \bar{e}^a . (iii) Defining Q so that it has all the required properties. The cylinder \mathcal{Y} is perpendicular to the plane $x + y + z = 0$, a plane to which all points of the construction are close to. The “triangle” defined by the skew lines \bar{e}^a wraps around the cylinder \mathcal{Y} .

Such a set Q is easy to find, for example by following the lines \bar{e}^a and connecting them far from the origin. See Fig. 2(iii) for an illustration.

We need the following claim on the distance of Q from the lines in $\bar{\mathcal{L}}$. Intuitively, it shows that the points in Q are far from the lines in $\bar{\mathcal{L}}$, and thus a certain detour is necessary to visit a line in $\bar{\mathcal{L}}$. Note that the bound would not be strong enough without the flattening.

▷ **Claim 9.** For any $q \in Q$ and $\ell \in \bar{\mathcal{L}}$ we have $\text{dist}(q, \ell) > 9.9\delta$.

▶ **Lemma 10 (Point gadget).** *Given a positive integer n and a point $q \in \mathbb{R}^3$, there is a set \mathcal{L} of $O(n^6)$ lines through q such that any TSPN tour of \mathcal{L} which is disjoint from the ball $B(q, \frac{1}{n^3})$ has length at least 20.*

Our construction is the union of the line set $\bar{\mathcal{L}}$ together with a point gadget placed at each point $q \in Q$; let \mathcal{L}^* denote the resulting line set.

2.2 Putting things together

▶ **Lemma 11.** *If G has a vertex cover of size k , then there is a tour in \mathcal{L}^* of length $10 + 19\delta k$. If \mathcal{L}^* has a tour of length $10 + 19\delta k$, then G has a vertex cover of size $1.011k$.*

The proof of the first part of the lemma is straightforward. To prove the second claim, we use the fact that the tour must touch the small balls $B(q, \frac{1}{n^3})$ for each point $q \in Q$ by Lemma 10. We can then consider a portion of the tour between two consecutive ball visits, i.e., a polygonal curve g that starts near some point $q \in Q$ and ends near some other point $q' \in Q$, and visits some of the lines in $\bar{\mathcal{L}}$ along the way. In the full version we show that g cannot touch lines from all three classes, in other words there is a segment \bar{e}^i such that all lines visited by g have an endpoint on \bar{e}^i . The proof relies on the property that Q avoids the cylinder \mathcal{Y} with axis $(0, 0, 0)^T, (1, 1, 1)^T$ and radius $\sigma/2$. Intuitively, if g would touch lines from all three classes, then it would have to go around the cylinder partially, which

10:8 On the Approximability of TSP with Line Neighborhoods

would be too costly. We can then define a vertex cover based on the tour portions g : for each line ℓ visited by g , the line ℓ has a point on \bar{e}^i that corresponds to some vertex v of the graph. These vertices v form a set W which is clearly a vertex cover; the goal is then to prove that $|W| \leq 1.011k$. The proof hinges on the fact that if a tour portion g contributes s unique vertices to W , then it must jump between non-incident lines of $\bar{\mathcal{L}}$ at least $(s - 1)$ times, which incurs a cost of at least $20(s - 1)\delta$ by Corollary 7. In case of $s = 1$, the tour still needs to visit *some* line in $\bar{\mathcal{L}}$, which incurs a cost of at least 19.8δ by Claim 9. Putting these observations together (and that the minimum cost tour of the balls $B(q, 1/n^3)$ has length very close to 10) yields the desired bound on $|W|$.

Proof of Theorem 1. Suppose that there is a polynomial time algorithm that approximates TSPN with lines in \mathbb{R}^3 within a factor of $1 + \frac{1}{230000}$. Let G be a given 3-partite graph. If G has a vertex cover of size $n/2$, then the above construction would have a tour of length $10 + 19\delta \frac{n}{2} = 10.002375$. On the other hand, if all vertex covers of G have size at least $\frac{34}{33} \frac{n}{2}$, then all tours of the construction have length at least $10 + 19\delta \frac{34}{33} \frac{n}{2 \cdot 1.011} > 10.00242$. As $10.00242/10.002375 > 1 + \frac{1}{230000}$, we could use the hypothetical approximation algorithm to distinguish between these two cases in polynomial time, which would imply $P = NP$. ◀

3 No $(2 - \epsilon)$ -approximation Algorithm

In this section we prove Theorem 3. In particular, we will show that when the objects are lines, TSPN is at least as hard to approximate as the Vertex Cover problem which is known to be hard to approximate to within a factor of $2 - \epsilon$, for any constant $\epsilon > 0$, under the Unique Games Conjecture (and inapproximable within a factor of 1.42 unless $P = NP$ [31]).

► **Theorem 12** ([32]). *Unless the Unique Games Conjecture is false, for any constant $\epsilon > 0$, there is no polynomial-time algorithm that, given a graph $G = (V, E)$ and an integer k , distinguishes between the cases (i) G has a vertex cover of size at most k or (ii) G has no vertex cover of size less than $(2 - \epsilon)k$.*

The main idea behind the reduction is to represent a graph G in Euclidean space such that:

- Each vertex $v \in V(G)$ corresponds to a point $p_v \in \mathbb{R}^d$,
- Each edge $e = uv \in E(G)$ corresponds to a line going through the points p_u and p_v ,
- An optimal tour visits each line sufficiently close to the points p_v , and therefore the vertex set corresponding to the points in the vicinity of the tour is a vertex cover.

However, in order to enforce that an optimal tour passes through (or not too far from) the points p_v , we will have to further build upon this idea. In particular, for each vertex v , instead of constructing only one point p_v , we will construct a set P_v of polynomially many points corresponding to v . If there is an edge $e = uv \in E(G)$, then we connect each point corresponding to u with each point corresponding to w . More precisely, for each edge uv and for every pair of points (p_u, p_w) with $p_u \in P_u$ and $p_w \in P_w$, we add a line going through p_u and p_w . Notice that the number of edges increases quadratically in the number of vertex copies. Therefore, tours that visit lines away from the vertices are disproportionately affected, which forces an optimal tour to visit lines at (or close to) the points in P_v .

Another key aspect of our construction is that we position the points of $\mathcal{P} := \bigcup_{v \in V(G)} P_v$ in \mathbb{R}^d so that the distance between any pair of distinct points is (roughly) the same. This helps us to have a more direct correspondence between the cost of the optimal tour and the size of an optimal vertex cover. The reduction is described formally in the next subsection.

3.1 Reduction: Vertex Cover to TSP with Line Neighborhoods

Take an instance of the Vertex Cover problem on a graph $G = (V, E)$ with n vertices and m edges. We first take a *lexicographic product* of the graph G with an independent set of size $\alpha = n^2$. Informally speaking, we construct a graph G' by making α copies of each vertex $v \in V(G)$, and denote the corresponding vertex set by Q_v . Then, for each edge $vw \in E(G)$, we add edges between every pair of vertices $v^i \in Q_v$ and $w^j \in Q_w$, thus forming a complete bipartite graph on Q_v and Q_w . More formally, the graph G' is defined as:

$$V(G') = \{v^i : v \in V(G) \wedge i \in [\alpha]\} \quad \text{and} \quad E(G') = \{v^i w^j : vw \in E(G) \wedge v \neq w \wedge i, j \in [\alpha]\}.$$

Next, we use the graph G' to construct an instance $I(G')$ of the TSPN with line neighborhoods problem in $d = O(\delta^{-2} \ln n')$ dimensions for any small enough $\delta > 0$ and with $n' = |V(G')| = \alpha \cdot n$. We map each vertex v of G' to a point p_v in \mathbb{R}^d such that for any two points p_v and p_u with $v \neq u, v, u \in V(G')$ the distance $\text{dist}(p_v, p_u)$ between them satisfies the following property: $1 \leq \text{dist}(p_v, p_u) \leq 1 + \delta$.

The fact that this is possible and can be done in polynomial time follows by Theorem 3.1 by Engebretsen, Indyk and O'Donnell [23], which we restate for completeness in the Appendix as Theorem 19. In particular, we can employ the theorem in order to deterministically map a unit side length simplex from $\mathbb{R}^{n'-1}$ to \mathbb{R}^d such that the desired property holds for all pairs of points.

We denote the resulting point set by P . Next, we create a collection of lines \mathcal{L} in an instance of TSPN, by adding to \mathcal{L} a line ℓ_{vw} passing through points p_v and p_w if $vw \in E(G')$.

We devote the rest of this section to prove completeness and soundness of our reduction.

Completeness. Suppose the graph G has a vertex cover of size $\leq k$. Then we claim that there is a tour T of cost at most $\alpha k(1 + \delta)$ that touches each line at least once. To see this, let $S = \{v_1, \dots, v_k\}$ denote the vertex cover of G . By construction, $S' = \{v_j^i : i \in [\alpha] \wedge j \in [k]\}$ is a vertex cover of G' . By the construction of \mathcal{L} and by the fact that S' is a vertex cover of G' , it follows that any tour that visits points $p_{v_1^1}, p_{v_1^2}, \dots, p_{v_k^\alpha}$ (in any order) is a feasible tour, i.e., it touches all lines in \mathcal{L} . So, in total such a tour visits a total of at most αk points, and the distance between any pair of these points is by construction at most $1 + \delta$. Thus, there is a solution to TSPN with cost at most $\alpha k(1 + \delta)$.

Soundness. We show that if there is a tour of cost x (where $x \leq \alpha n(1 + \delta)$), then there is a vertex cover in G of size at most $\frac{x}{\alpha(1-2\Delta)\lambda}$, where Δ is a small positive number and $\lambda \in [0, 1]$ is very close to 1.

The intuition behind Δ is that it describes the maximum distance that the tour is allowed to have to a given point, assuming that the vertex corresponding to that point contributes to the vertex cover. For each point $p_{v^i} \in P$ (note that $v^i \in V(G')$), let $B(v^i)$ be a d -dimensional ball of radius Δ centered at p_{v^i} . Note that Δ is small enough so the only lines from \mathcal{L} intersecting a ball $B(v^i)$ are the ones that go through p_{v^i} . Given a tour T , we say that a ball $B(v^i)$ is *non-empty* if $T \cap B(v^i) \neq \emptyset$; otherwise, we say that $B(v^i)$ is *empty*. We say that a line ℓ_{uw} is *covered by a ball* if at least one of the balls $B(u)$ and $B(w)$ is non-empty. Otherwise ℓ_{uw} is *not covered by a ball*. We first show that any point $p \in \ell_{uw}$ that is outside the two balls corresponding to u and w will not be “too close” to any other line:

► **Lemma 13.** *For any point $p \in \ell_{uw}$ such that $p \notin B(u)$ and $p \notin B(w)$ and for any $\ell \in \mathcal{L} \setminus \{\ell_{uw}\}$ we have $\text{dist}(p, \ell) \geq \Delta/2$.*

We are now ready to prove that any optimal tour T must cover almost all lines by balls:

10:10 On the Approximability of TSP with Line Neighborhoods

► **Lemma 14.** *Let T be a tour of cost at most x with $x \leq \alpha n(1 + \delta)$ for the instance $I(G')$. Then the number of lines of $I(G')$ that are not covered by balls is at most $\frac{2x}{\Delta}$.*

Proof. Note it is without loss of generality to assume that T consists of line segments with endpoints on lines of $I(G')$. By Lemma 13 any line $\ell_{u^i w^j} \in \mathcal{L}_{uw}$ that is visited at a point p with $p \notin B(u^i)$ and $p \notin B(w^j)$, must have two adjacent segments on T of length at least $\Delta/2$ each. Since the total tour cost of T is at most x there can be at most $\frac{x}{\Delta/2} = \frac{2x}{\Delta}$ lines that are visited by T outside a ball. ◀

Let $\lambda = 1 - \varepsilon^2$ and set $\alpha = n^2$. We can construct a vertex cover of G based on a tour T the following way: if a set Q_v has at least $\lambda\alpha$ non-empty balls, then we add v to the vertex cover.

► **Lemma 15.** *The set $S = \{v : |\cup_{i \in [a]} \{v^i : B(v^i) \text{ is non-empty}\}| \geq \lambda\alpha\}$ is a vertex cover of G of size $|S| < \frac{x}{\alpha(1-2\Delta)\lambda}$.*

Proof. We first argue that S is a vertex cover of G . Assume for the sake of contradiction that some edge $uv \in E(G)$ is not covered by S . Then it must be the case that there are at least $\alpha(1 - \lambda)$ empty balls among the balls corresponding to both u and v . But any line defined by two such empty balls corresponding to u and v is not covered by a ball. In total there are more than $(1 - \lambda)^2 \alpha^2 = \Omega(n^4)$ many such lines. This is a contradiction, since by Lemma 14 there can be at most $\frac{2n\alpha k(1+\delta)}{\Delta} = O(n^3)$ such lines in total over the whole instance.

Let S be the vertex cover of G we have obtained. Since the distance of any two balls is at least $1 - 2\Delta$, and we have visited at least $\alpha\lambda$ balls among Q_v for each $v \in S$, the total cost of the tour is at least

$$x > |S|\alpha\lambda(1 - 2\Delta),$$

therefore we have that $|S| < \frac{x}{\alpha\lambda(1-2\Delta)}$. ◀

Proof of Theorem 3. Suppose that there is an algorithm that can distinguish in polynomial time, for any $0 < \varepsilon'$ and any $x \in \mathbb{R}_+$, whether there is a tour of length at most x or all tours have length at least $(2 - \varepsilon')x$. Take some instance of vertex cover, where the goal is to decide if there is a vertex cover of size at most k or all vertex covers of the graph have size at least $(2 - \varepsilon)k$, where $\varepsilon \in (0, 0.1]$. By the above polynomial construction, it would be sufficient to distinguish the cases where $\mathcal{I}(G')$ has a tour of size at most $k\alpha(1 - 2\Delta)\lambda$ (implying a vertex cover of size at most k), or all tours have length at least $(2 - \varepsilon)k\alpha(1 + \delta)$ (implying that all vertex covers have size at least $(2 - \varepsilon)k$). If we set $\delta = \Delta = \varepsilon^2$, then we get that the ratio of these tours is:

$$\frac{(2 - \varepsilon)k\alpha(1 + \delta)}{k\alpha(1 - 2\Delta)\lambda} = \frac{(2 - \varepsilon)(1 + \varepsilon^2)}{(1 - 2\varepsilon^2)(1 - \varepsilon^2)} < 2,$$

so the hypothetical algorithm on $I(G')$ distinguishes these cases, which is a contradiction. ◀

We note that our reduction implies that TSPN with Line Neighborhoods is Vertex Cover hard, and therefore also inapproximable within a factor of $\sqrt{2} - \varepsilon$ unless $P = NP$ [31].

4 A Quasipolynomial-Time Approximation Algorithm

In this section, we will show a quasi-polynomial time algorithm to approximate TSPN for lines to a factor of $O(\log^2 n)$. In fact, our approach is more general: we show how to $O(\log N \log n)$ -approximate TSPN for discrete neighborhoods of total size N , in running time

$N^{O(\log \log N)}$ for any fixed d . In this problem, we are given n neighborhoods $P_i \subset \mathbb{R}^d$, which are discrete sets of points. Let $P = \bigcup_{i \in [n]} P_i$, $N = |P|$. Using the approach of Dumitrescu and Tóth [21], we can convert any instance of TSPN with line neighborhoods into an instance of discrete TSPN on a set of $N = O(n^4)$ points and n neighborhoods. This transformation has a running time of $O(N)$, and incurs the loss of a constant factor in the approximation. From now on, we focus on TSPN for discrete neighborhoods.

Our main result is an $O(\log N \log n)$ -approximation algorithm that runs in time $N^{O(\log \log N)}$ for constant d . Our algorithm combines the dynamic program by Arora [3] with the framework of Chalermsook et al. [10, 9]. As Dumitrescu and Tóth show [21], TSPN is related to the group Steiner tree problem, and can be reduced to this problem to obtain an $O(\log^3 n)$ -approximation. We show that, using the structure of the Euclidean space, which is exploited in the algorithm presented by Arora for TSP, we can use the techniques of Chalermsook et al. to approximate discrete instances of TSPN and group Steiner tree in \mathbb{R}^d .

We note that, even on tree metrics, the group Steiner tree problem is $\Omega(\log^2 n / \log \log n)$ -hard to approximate under the projection games conjecture [27, 26]. As every tree metric can be embedded into some Euclidean space with distortion $O(\sqrt{\log \log n})$ [34, 36, 37], the group Steiner tree problem in Euclidean space is also hard to approximate to within $\Omega(\log^2 n / (\log \log n)^{3/2})$ under the same assumption.

► **Theorem 16.** *There is a randomized $O(\log N \log n)$ -approximation algorithm for TSPN with discrete neighborhoods in \mathbb{R}^d that runs in time $N^{O(\log \log N)}$ for constant d .*

The theorem above, together with the result of Dumitrescu and Tóth [21] imply Theorem 5, along with the derandomization techniques of Arora [3] and Charikar et al. [12].

We start by recalling the main steps of the PTAS for TSP by Arora, as our result builds upon the dynamic program used there. While describing the algorithm, we state some modifications that are necessary for our purpose. Then, we show how to use the framework of Chalermsook et al. [10, 9] to find a feasible solution using the dynamic program. We note that, among different results on Euclidean TSP, the work of Arora yields the best running time for our algorithm as the use of other techniques, e.g., spanners [42, 5], to improve the running time is not compatible with our approach. Nevertheless, an existence of a “pure” dynamic program for Euclidean TSP that has constant number of branches in each recursive call will immediately lead to a polynomial-time algorithm without further modification.

4.1 Chalermsook et al.’s Framework

Firstly, we discuss the technique of Chalermsook et al. [10, 9] that reduces an instance of the group Steiner tree problem on bounded treewidth graphs into a tree-instance. In fact, the algorithm of Chalermsook et al. works on arbitrary graphs, but the size of the tree produced depends on the running time of a dynamic program or any recursive algorithm that solves the corresponding point-to-point problem, i.e., the classical Steiner tree problem. Specifically, the resulting tree will have a polynomial-size only if the recursive algorithm terminates in polynomial-time. Indeed, each node in the tree corresponds to a state in the search space of the dynamic program. In our case, while a polynomial-time algorithm is not available for Euclidean TSP (due to its NP-hardness), there exists a pure dynamic-programming based PTAS by Arora [3], which suffices for reducing the Euclidean TSP instance into a tree-instance of the group Steiner tree problem, albeit losing a constant factor in the approximation ratio.

Now we wish to extract a near-optimal solution of the original problem from the tree-instance, which appears as a subtree. This is relatively straightforward for the point-to-point network design problems like the classical Steiner tree problem and Euclidean TSP. The

10:12 On the Approximability of TSP with Line Neighborhoods

neighborhood problem (i.e., TSPN), on the other hand, has multiple points where we can enter a neighborhood, and each choice affects the solution globally. Chalermsook et al. solves this issue by exploiting linear programs to find an (fractional) optimal solution that satisfies the global constraints. Henceforth, we are left with rounding a fractional solution obtained from the LP, and this problem reduces to solving a group Steiner tree problem on the tree-instance, which admits an $O(\log N \log n)$ -approximation algorithm.

Lastly, we remark that, while Chalermsook's algorithm runs in polynomial-time, our algorithm for TSPN runs in slightly super polynomial-time even in constant dimension. This is because each recursive call in Arora's algorithm may branch up to $O(\log n)^{2^{O(d \log d)}}$ sub-processes, thus causing the transformation to produce a tree of size $O(n^{2^{O(d \log d)} \log \log n})$.

The remaining part of this section is devoted to give more details on our algorithm. The full discussion and proofs are deferred to an appendix due to page limit.

4.2 Arora's Algorithm

In this section, we briefly summarize the algorithm of Arora [3] (a more detailed description can be found in Appendix B.1). This algorithm approximates TSP to a factor of $1 + 1/c$; for our purpose, it is sufficient to consider $c = 1$. Arora's algorithm has three main steps:

1. Perturbation, which ensures that all coordinates are integral and bounded by $O(n)$;
2. Construction of a shifted quadtree;
3. Dynamic program, which finds the approximate solution for TSP.

The dynamic program is based on the (m, r) -multipath problem (see Definition 21), which given a cell of the quadtree and a set of pairs of *portals* on the boundary of the cell, has as its objective to find a minimum-cost set of paths, each connecting a pair of portals, and such that all of the points in the cell are visited. We refer to the multiset of portals and their pairing as the *state* of an (m, r) -multipath problem.

Two main changes are required to use the algorithm by Arora to approximate TSPN. First, we must guess a point v_0 in an optimum solution, as well as a value $R = O(\text{OPT})$ (see Appendix B.1.1). Second, we must allow the solutions to the (m, r) -multipath problem to not visit every point in the cell. We achieve this by adding a *visit bit* to the state of the (m, r) -multipath problem on leaves, which indicates if the (unique) point in the cell must be visited (it is True), or it is sufficient to connect the portals (see Appendix B.1.3).

4.3 Approximating TSPN using the framework by Chalermsook et al.

After perturbation and construction of the shifted quadtree, we use the dynamic program above to define a *dynamic programming graph*. The intuition is that a solution to the problem can be represented as a tree in this graph, where the vertices in the tree correspond to all of the (m, r) -multipath problems that assemble into the solution.

We now describe the nodes and edges of this graph, denoted by H .

- **Nodes:** There are two types of nodes, which we refer to as *subproblem nodes* and *combination nodes*. The graph contains one subproblem node for every entry of the modified dynamic programming table in Section 4.2, i.e. one node for each instance of the (m, r) -multipath problem for every cell, pairing of portals and visit bit (for leaves). Combination nodes correspond to the possibilities of recursion for a given subproblem: for a given (m, r) -multipath problem (for a non-leaf cell), there is a combination node for every possible way for the p paths to cross the boundary between children cells.
- **Root:** The root of H corresponds to (m, r) -multipath on the root cell with no portals.

- **Edges:** There are (directed) edges connecting the node for each (m, r) -multipath problem to the corresponding combination nodes, and then the combination nodes to the corresponding nodes for the subproblems in the children cells.
- **Costs:** Edges incident to leaf nodes have cost equal to the corresponding entry in the dynamic programming table; all other edges have cost 0.

Using this definition, we can represent any (m, r) -light salesman path, that is a path which crosses each facet of each cell at most r times and always at a portal, as a tree T in H . For each cell, the solution restricted to that cell consists of a union of disjoint paths, which induce a set of portals and their pairing, and hence an instance of the (m, r) -multipath problem. We include the corresponding subproblem node in T . For each non-leaf cell, there is a combination node which represents the way in which the paths cross boundaries between children cells. We add that combination node to T , as well as all of the edges containing it.

The trees obtained by this process have a specific structure, which was implicitly formulated in the work of Chalermsook et al. [10, 9], and which we formalize below.

► **Definition 17 (Solution tree).** *Let H be a DAG with root r , and its nodes be partitioned into combination nodes H_c , and subproblem nodes H_p . We say an out-arborescence $T \subseteq H$ rooted at r is a solution tree if:*

1. *Every combination node $t^c \in T \cap H_c$ has full out-degree (i.e., all children are also in T),*
2. *Every non-leaf subproblem node $t \in T \cap H_p$ (including the root r) has out-degree 1 in T .*

As we mentioned above, we can construct a solution tree for any (m, r) -light salesman path. The converse is also true: for each solution tree, there is a corresponding (m, r) -light salesman path. The final requirement for a solution to be feasible is that each neighborhood must be covered, meaning that the tour must intersect every neighborhood.

Consider a tour corresponding to a solution tree in T . The set of points visited by this tour is exactly the set of points contained in the leaf cells for subproblem nodes whose visit bit is set to **True**. In other words, the points of the leaf subproblems with visit bit **True** are visited by the tour, and so the corresponding neighborhoods are covered. Therefore, we can solve TSPN by formulating it as finding a solution tree that covers every neighborhood.

Let S_i be the set of all subproblem nodes whose visit bit is **True**, and whose cell contains a point in P_i . We can formulate our goal as finding a minimum-cost solution tree that contains at least one node of each set S_i . This problem resembles GST, and is defined in the work of Chalermsook et al. [10, 9]. We redefine the problem using our own notation.

► **Definition 18 (Solution Tree Group Steiner Tree (STGST)).** *Let H be a DAG with edge-costs $cost : E(H) \rightarrow \mathbb{R}$ and root r , as well as groups $S_i \subseteq V(H)$, for $i \in [h]$, and a partition of the nodes into H^c and H^p . The objective of this problem is to find a minimum-cost solution tree T that contains at least one vertex of every group S_i .*

Their work shows that we can obtain an $O(\log^2 n)$ -approximation to STGST (see [10, Sec. 4]). By reducing TSPN to STGST, we can apply the same approximation result, thus proving Theorem 16. We show the details of these steps in Appendix B.2.

5 Conclusion

We have shown that TSPN with line neighborhoods is APX-hard, so a PTAS for this problem is unlikely. This implies the same hardness for k -dimensional flats in \mathbb{R}^d for $1 \leq k \leq d - 2$, which together with the known PTAS results for $k = 0$ and $k = d - 1$ gives a complete classification of these problems. We have also proved a stronger inapproximability factor

for $d = O(\log n)$: there is no $(\sqrt{2} - \epsilon)$ -approximation assuming $P \neq NP$ and no $(2 - \epsilon)$ -approximation assuming the UGC. On the positive side, we gave an $O(\log^2 n)$ -approximation algorithm in slightly superpolynomial time.

There is still a large gap between the lower bounds and the algorithms for TSPN with line neighborhoods. Perhaps the most important question related to TSPN is to find a constant-approximation for line neighborhoods in \mathbb{R}^3 , or to prove that it does not exist. Furthermore, for general point sets in higher dimensions there is an inapproximability of $\Omega(\log^2 n / (\log \log n)^{3/2})$ under the Projection Games Conjecture. Whether that holds for flats or lines is an open problem.

References

- 1 Antonios Antoniadis, Krzysztof Fleszar, Ruben Hoeksma, and Kevin Schewior. A PTAS for Euclidean TSP with hyperplane neighborhoods. In *SODA '19*, pages 1089–1105. SIAM, 2019. doi:10.1137/1.9781611975482.67.
- 2 Esther M. Arkin and Refael Hassin. Approximation algorithms for the geometric covering salesman problem. *Discret. Appl. Math.*, 55(3):197–218, 1994.
- 3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.
- 4 Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *SODA '98*, pages 33–41. ACM/SIAM, 1998. URL: <http://dl.acm.org/citation.cfm?id=314613.314632>.
- 5 Yair Bartal and Lee-Ad Gottlieb. A linear time approximation scheme for Euclidean TSP. In *FOCS'13*, pages 698–706. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.80.
- 6 Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The traveling salesman problem: Low-dimensionality implies a polynomial time approximation scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016. doi:10.1137/130913328.
- 7 Hans L. Bodlaender, Corinne Feremans, Alexander Grigoriev, Eelko Penninx, René Sitters, and Thomas Wolle. On the minimum corridor connection problem and other generalized geometric problems. *Comput. Geom.*, 42(9):939–951, 2009. doi:10.1016/j.comgeo.2009.05.001.
- 8 Jean Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.
- 9 Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. In *APPROX-RANDOM*, volume 116 of *LIPICs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 10 Parinya Chalermsook, Syamantak Das, Bundit Laekhanukit, and Daniel Vaz. Beyond metric embedding: Approximating group Steiner trees on bounded treewidth graphs. In *SODA '17*, pages 737–751. SIAM, 2017. doi:10.1137/1.9781611974782.47.
- 11 T.-H. Hubert Chan and Shaofeng H.-C. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. *ACM Trans. Algorithms*, 14(1):9:1–9:18, 2018. doi:10.1145/3158232.
- 12 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k -median. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 114–123. ACM, 1998. doi:10.1145/276698.276719.
- 13 Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *FOCS'05*, pages 245–253. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.9.

- 14 Andrea E. F. Clementi, Pierluigi Crescenzi, and Gianluca Rossi. On the complexity of approximating colored-graph problems. In *COCOON'99*, volume 1627 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 1999. doi:10.1007/3-540-48686-0_28.
- 15 Mark de Berg, Joachim Gudmundsson, Matthew J. Katz, Christos Levcopoulos, Mark H. Overmars, and A. Frank van der Stappen. TSP with neighborhoods of varying size. *J. Algorithms*, 57(1):22–36, 2005. doi:10.1016/j.jalgor.2005.01.010.
- 16 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in h-minor-free graphs and algorithmic applications. In *STOC'11*, pages 441–450, 2011. doi:10.1145/1993636.1993696.
- 17 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *STOC'03*, pages 473–482. ACM, 2003. doi:10.1145/780542.780612.
- 18 Moshe Dror and James B. Orlin. Combinatorial optimization with explicit delineation of the ground set by a collection of subsets. *SIAM J. Discret. Math.*, 21(4):1019–1034, 2008. doi:10.1137/050636589.
- 19 Adrian Dumitrescu. The traveling salesman problem for lines and rays in the plane. *Discrete Math., Alg. and Appl.*, 4(4):44:1–44:12, 2012.
- 20 Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003. doi:10.1016/S0196-6774(03)00047-6.
- 21 Adrian Dumitrescu and Csaba D. Tóth. The traveling salesman problem for lines, balls, and planes. *ACM Trans. Algorithms*, 12(3):43:1–43:29, 2016. doi:10.1145/2850418.
- 22 Khaled M. Elbassioni, Aleksei V. Fishkin, and René Sitters. Approximation algorithms for the Euclidean traveling salesman problem with discrete and continuous neighborhoods. *Int. J. Comput. Geometry Appl.*, 19(2):173–193, 2009. doi:10.1142/S0218195909002897.
- 23 Lars Engebretsen, Piotr Indyk, and Ryan O'Donnell. Derandomized dimensionality reduction with applications. In *SODA'02*, pages 705–712. SIAM, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545476>.
- 24 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. Approximating metrics by tree metrics. *SIGACT News*, 35(2):60–70, 2004. doi:10.1145/992287.992300.
- 25 Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000. doi:10.1006/jagm.2000.1096.
- 26 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k / \log \log k)$ -approximation algorithm for directed Steiner tree: a tight quasi-polynomial-time algorithm. In *STOC'19*, pages 253–264. ACM, 2019. doi:10.1145/3313276.3316349.
- 27 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *STOC'03*, pages 585–594. ACM, 2003. doi:10.1145/780542.780628.
- 28 Su Jia and Joseph S. B. Mitchell. Geometric tours to visit and view polygons subject to time lower bounds, 2019. URL: https://www.andrew.cmu.edu/user/sjia1/wrp_with_tlb.pdf.
- 29 Håkan Jonsson. The traveling salesman problem for lines in the plane. *Inf. Process. Lett.*, 82(3):137–142, 2002. doi:10.1016/S0020-0190(01)00259-9.
- 30 Subhash Khot. On the power of unique 2-prover 1-round games. In *STOC'02*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.
- 31 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In *FOCS'18*, pages 592–601. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00062.
- 32 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.
- 33 Robert Krauthgamer and James R. Lee. Algorithms on negatively curved spaces. In *FOCS'06*, pages 119–132. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.9.
- 34 Nathan Linial, Avner Magen, and Michael E Saks. Low distortion Euclidean embeddings of trees. *Israel Journal of Mathematics*, 106(1):339–348, 1998.

10:16 On the Approximability of TSP with Line Neighborhoods

- 35 Cristian S. Mata and Joseph S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *SoCG'97*, pages 264–273. ACM, 1997. doi:10.1145/262839.262983.
- 36 Jiří Matoušek. On embedding trees into uniformly convex Banach spaces. *Israel Journal of Mathematics*, 114(1):221–237, 1999.
- 37 Jiri Matoušek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.
- 38 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999. doi:10.1137/S0097539796309764.
- 39 Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *SODA'07*, pages 11–18. SIAM, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283385>.
- 40 Joseph S. B. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *SoCG'10*, pages 183–191. ACM, 2010. doi:10.1145/1810959.1810992.
- 41 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comput. Sci.*, 4(3):237–244, 1977.
- 42 Satish Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *STOC'98*, pages 540–550, 1998. doi:10.1145/276698.276868.
- 43 Shmuel Safra and Oded Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. *Comput. Complex.*, 14(4):281–307, 2006. doi:10.1007/s00037-005-0200-3.
- 44 Luca Trevisan. When Hamming meets Euclid: The approximability of geometric TSP and Steiner tree. *SIAM J. Comput.*, 30(2):475–485, 2000. doi:10.1137/S0097539799352735.

A Theorem 3.1 from [23]

► **Theorem 19** (3.1 from [23]). *Let v_1, v_2, \dots, v_m be a sequence of vectors in \mathbb{R}^d and let $\varepsilon, F \in (0, 1]$. Then we can compute, in deterministic time $O(dm (\log n + \frac{1}{\varepsilon})^{O(1)})$, a linear mapping $A : \mathbb{R}^d \rightarrow \mathbb{R}^k$ where $k = O(\log(1/F)/\varepsilon^2)$ such that:*

$$k|v_i|^2 \leq |Av_i|^2 \leq k(1 + \varepsilon)|v_i|^2$$

for at least a fraction $1 - F$ of i 's.

B Details of Section 4

Among all the PTASes for Euclidean TSP, we choose to base our algorithm on the work of Arora, as it results in the lowest running time for our algorithm. Unfortunately, the results of Rao and Smith [42], and Bartal and Gottlieb [5] cannot be adapted for our purposes, since their algorithms use spanners to reduce the total weight of the graph to be a constant factor away from the optimum. It is unclear if this technique can be used for discrete TSPN, as the spanner contains the entire set P of points, and a minimum-cost tree spanning P may be much larger than the optimum solution. In other words, the total weight of the graph can be more than a constant factor away from the weight of an optimal solution.

B.1 Arora's Algorithm

Arora's algorithm consists of three main steps:

1. Perturbation, which changes the instance so that all coordinates are integral and bounded by $O(n)$;
2. Construction of a shifted quadtree;
3. Dynamic program, which finds the approximate solution for TSP.

We describe all of these steps, including any minor alterations needed for them to work in our setting.

B.1.1 Perturbation

Arora shows how to perturb the solution such that:

1. All nodes have integer coordinates;
2. Every (non-zero) distance between two points is at least 8 units;
3. The maximum distance between two points is $O(n)$.

Given a bounding box on the instance of size L_0 , Arora achieves this perturbation by snapping points to an appropriately fine grid. To use this step for our problem, we need to specify a value of L_0 such that $\text{OPT} \leq L_0 \leq O(\text{OPT})$. To this effect, we guess the value of OPT rounded up to a power of 2, as well as a vertex v_0 that is included in an optimum solution. We implement this guessing step by iterating over all of the possible values, and computing a feasible solution for each possibility. The best feasible solution we obtain will be at least as good as the solution for the correct guess (in expectation).

The guessing step is done as follows. We start by guessing a vertex v_0 that is contained in an optimum solution. Then, we compute the minimum radius R_0 such that at least one point from each neighborhood is contained in the ball B of radius R_0 centered at v_0 . Such a ball can be computed simply by iterating over all neighborhoods and finding the neighborhood's nearest point to v_0 . If the optimum solution contains v_0 , then its cost is at least R_0 , as it must visit the farthest neighborhood, at distance R_0 . On the other hand, $\text{OPT} \leq 2R_0n$, since the ball B contains at least one point from each neighborhood, and the distance between any two points in B is at most $2R_0$. Hence, there is a tour of cost at most $2R_0n$. Knowing that $R_0 \leq \text{OPT} \leq 2R_0n$ (assuming v_0 is in an optimum solution), we can simply run the algorithm for every v_0 and for any $R \in [R_0, 4R_0n]$ that is a power of 2.

Given a vertex v_0 and a guess R for the value of the optimum solution, we set $L_0 = R/2$ (so that if $R/2 \leq \text{OPT} \leq R$, $L_0 \leq \text{OPT}$). Finally, we remove all of the vertices $u \in P$ that are at a distance more than R from v_0 , that is, $\text{dist}(v_0, u) > R$. A solution containing both v_0 and u would cost more than $R \geq \text{OPT}$, implying that for correct choices of R and v_0 , such vertices can be safely removed. We now have a bounding box of side length $4L_0$ containing all the points in the instance, and hence the perturbation step in Arora's algorithm ensures the stated properties.

B.1.2 Construction of a shifted quadtree

Let $L = O(n)$ be the size of the bounding box. The algorithm computes a random shift $a' = (a'_1, a'_2, \dots, a'_d)$, with $a'_i \in \{0, \dots, L-1\}$, $i \in [d]$. Then, it constructs a quadtree where the dissection points are shifted according to a' . The resulting quadtree has height $O(\log n)$, and $O(n \log n)$ cells. For our purpose, no changes are needed to this process.

B.1.3 Dynamic Program

Arora's algorithm uses dynamic programming to find a *salesman path*, which may visit additional points along the boundary of the cells of the quadtree. The following definition formalizes this concept.

► **Definition 20.** *Let m, r be positive integers. An m -regular set of portals for a shifted dissection is a set of points on the facets of the cells in it. Each cell has a portal at each of its vertices and m other portals on each facet, placed in a $d - 1$ -dimensional square grid whose vertices are identical to the vertices of the facet.*

A salesman path is a path in \mathbb{R}^d that visits all the input points, and some subset of portals. It may visit a portal more than once.

The salesman path is (m, r) -light with respect to the shifted dissection if it crosses each facet of each cell in the dissection at most r times and always at a portal.

The goal of the dynamic program is to find a minimum cost (m, r) -light salesman path, for the instance. For our purpose, a 2-approximation of TSP is sufficient, and hence we set $m = O(\sqrt{d} \log n)^{d-1}$ and $r = O(\sqrt{d})^{d-1}$. By restricting the solution to cross the cell boundaries only through portals, we can see that any solution to the problem, restricted to a single cell, consists of a set of paths that together cover all of the points inside the cell. Since we want to find an (m, r) -light solution, this further implies that at most r portals per facet of the cell are used. This motivates the definition of the (m, r) -multipath problem, which is the problem solved by the dynamic program for each cell:

► **Definition 21** ((m, r) -multipath problem [3]). *An instance of this problem is specified by the following inputs:*

1. *A nonempty cell in the quadtree.*
2. *A multiset of r portals on each of the $2d$ facets of this cell such that the sum of the sizes of these multisets is an even number $2p \leq 2dr$.*
3. *A pairing $(a_1, a_2), (a_3, a_4), \dots, (a_{2p-1}, a_{2p})$ between the $2p$ portals specified in Item 2.*

The goal in the (m, r) -multipath problem is to find a minimum cost collection of p paths in the cell that is (m, r) -light. The i -th path connects a_{2i-1} to a_{2i} , and the p paths together visit all the points in the cell.

The dynamic programming table consists of all of these instances of (m, r) -multipath problem, for each cell and pairing of portals (considered here to include the multiset of portals in Item 2. We refer to the multiset of portals and their pairing as the *state* of an (m, r) -multipath problem.

The values of the table can be computed recursively. The entries corresponding to leaves of the quadtree can be easily determined: given the portal set of size $2p$ and the pairing, we simply need to find the shortest paths between the paired portals, and add the (single) point in the cell to one of these paths. For all other entries, the algorithm enumerates all possible ways that the p paths can cross the boundary between children cells. For each of these arrangements, the cost of the solution can be obtained by summing the costs of the respective instances for the children cells. Once all of the entries have been computed, the minimum cost (m, r) -light salesman path can be found by looking at the (m, r) -multipath problem for the root cell of the quadtree with no portals used.

The dynamic programming table contains a total of $O(n(\log n)^{O(d)^{(d-1)/2}})$ entries, and the value at each cell can be computed in time $(\log n)^{O(d)^{(d-1)/2}}$. Therefore, the running time of this algorithm is $O(n(\log n)^{O(d)^{(d-1)/2}})$.

Our algorithm uses a very similar dynamic program, with only a small change needed at the leaves. In the TSP problem, all of the points must be visited, which implies that any feasible solution to the (m, r) -multipath problem must visit all the points contained in that cell. However, the same is not true of the TSPN problem: as long as one point from each neighborhood is visited in the whole path, the solution is feasible, which means that not all neighborhoods are visited in every cell that intersects them. To that effect, we add an extra input to the (m, r) -multipath problem for leaf cells, which we call *visit bit*. If the visit bit is set to `True`, then the (single) point in the cell must be visited; if it is set to `False`, then the solution only needs to connect the portals as specified in the input (meaning that the optimum solution will be a union of shortest paths between paired portals).

We remark that finding a solution in this new dynamic program can be thought of as two different tasks: choosing which points in the leaf cells should be visited (by choosing the corresponding subproblems with visit bit `True` or `False`) and choosing the tour visiting these points (using the portals as in the original dynamic program).

B.2 Approximating TSPN using the framework by Chalermsook et al.

To prove Theorem 16, we need to show how to formulate TSPN as an instance of STGST, and then show how to obtain an $O(\log^2 n)$ -approximation to this problem. In this section, we provide details to both of these steps.

B.2.1 Formulating discrete TSPN as an instance of STGST

We will formally describe the construction of a DAG H based on the dynamic program for TSP. We assume that the perturbation and random shift steps implemented by Arora have been performed, with the alterations described in Section 4.2.

We now consider the dynamic program as presented by Arora, and construct our DAG H as follows. The vertex set is partitioned into *subproblem nodes* H_p and *combination nodes* H_c .

- For every (m, r) -multipath subproblem considered by Arora, we create a *subproblem node*. Formally, for every cell C in the quadtree, and every state A or (A, b) (where b represents the visit bit if C is a leaf cell), we create a node $t[C, A]$ (resp. $t[C, A, b]$) in H_p .
- For every non-leaf cell C with children C_1, \dots, C_k and states X for C and X_i for C_i , we add a *combination node* $t^c[C, X, \{X_i\}_{i \in [k]}]$ if the states are consistent, that is, if the combination of the portal pairings for each of the cells C_i forms the portal pairing represented by X in C .
- For each combination node $t' = t^c[C, X, \{X_i\}_{i \in [k]}]$, we add edges from $t[C, X]$ to t' and from t' to $t[C_i, X_i]$ for each $i \in [k]$.
- The edges entering leaf nodes $t[C, A, b]$ have cost equal to the minimum cost of a solution to the (m, r) -multipath problem in C with portal pairings specified by A , and which visits the point in C if $b = \text{True}$.
- All other edges have cost 0.

The root of H is the node $t[C, X]$, where C is the root cell of the quadtree (the bounding box of the instance), and X represents an empty set of portals. The height of the resulting DAG H is $O(\log n)$, since its structure is similar to the dynamic program of Appendix B.1.

► **Lemma 22.** *Let $v_0 \in P$ be a point and R_0 be a radius guessed in Appendix B.1.1.*

For every (m, r) -light tour F in the resulting quadtree there is a solution tree X in H such that $\text{cost}(F) = \text{cost}(X)$ and they visit the same set of points in P .

Similarly, for any solution tree $X \subset H$, there is an (m, r) -light tour F of the same cost, which visits the same points in P .

B.2.2 Obtaining an $O(\log^2 n)$ -approximation

We will use the following theorem, which is implicit in the work of Chalermsook et al. [10, Section 4]. We remark that, even though the work of Chalermsook et al. is formulated in terms of the tree decomposition of a graph, it also applies when obtaining the DAG from a quad-tree, as the necessary properties still hold.

► **Theorem 23** ([10, 9]). *Let H be a DAG with edge-costs $\text{cost} : E(H) \rightarrow \mathbb{R}$ and root r , as well as groups $S_i \subseteq V(H)$, for $i \in [h]$, and a partition of the nodes into H_c and H_p . There is an algorithm that outputs a solution tree $X \subseteq H$ sampled from a distribution \mathcal{D} such that:*

1. $\mathbb{E}_{X \sim \mathcal{D}}[\text{cost}(X)] \leq \text{cost}(\text{OPT})$, where $\text{cost}(\text{OPT})$ denotes the cost of the optimal solution
2. For any group S_i , the probability that the group is covered (for some constant $\alpha > 1$) is $\Pr_{X \sim \mathcal{D}}[|S_i \cap X| > 0] \geq \frac{1}{\alpha \text{height}(H)}$

The algorithm runs in time $\Delta(H)^{O(\text{height}(H))}$, where $\Delta(H)$ is the max-degree of H .

We will now show how to use Theorem 23 and Lemma 22 to obtain an $O(\log N \log n)$ -approximation for the TSPN problem on discrete neighborhoods, and hence prove Theorem 16.

We start by guessing a vertex v_0 to be the starting point of our solution. For every vertex $v_0 \in P$, we compute the minimum radius R_0 such that every neighborhood contains a point at distance at most R_0 from v_0 . Next, we guess R , an approximation for OPT, in the range $[R_0, 4nR_0]$. For the powers $R = 2^i$, $i \in \mathbb{Z}$, $R_0 \leq R \leq 4nR_0$, we can now preprocess the instance according to the perturbation step of Arora's algorithm. (Appendix B.1.1). Next, we enumerate the shift $a = (a_1, \dots, a_d) \in \{0, \dots, L-1\}^d$, and construct the shifted tree as in Arora's algorithm (Appendix B.1.2). Finally, we construct the DAG H based on the dynamic programming table, as specified in Appendix B.2.1. We recall that the height of the tree, as well as of DAG H is $O(\log N)$.

We now use Theorem 23 repeatedly to obtain solution trees X_1, \dots, X_ℓ , where $\ell = c \log n \log N$, and c is a large constant. Then, we use Lemma 22 to convert each solution tree X_i into a tour F_i , and finally take the union of all these tours to obtain a solution F . While F is not necessarily a tour, it is simple enough to remove crossings. For every neighborhood P_i that is not visited by F , we add a detour visiting the closest point in P_i . We denote by F^* the minimum-cost solution among all solutions F for all the enumerated values of v_0 , R_0 , and a .

By construction, F^* is a feasible solution, as it is a tour that visits every group. To prove that it is $O(\log N \log n)$ -approximate, consider the solution F' that we obtained for the correct values of v_0 , R_0 , and a , that is, for a vertex v_0 in an optimum solution, R_0 such that $R_0/2 \leq \text{OPT} \leq R_0$, and a shift a for which an (m, r) -light tour exists. By Theorem 23, each of the solution trees X'_i obtained has expected cost at most OPT, and by Lemma 22, the corresponding tour F'_i also has expected cost at most OPT. Therefore, the union of all tours F'_i costs at most $O(\log N \log n \text{OPT})$ in expectation. The probability that a neighborhood is not visited, and hence that we must add a detour, is (for sufficiently large c)

$$\begin{aligned} \Pr \left[\bigcap_j |S_i \cap X_j| = 0 \right] &\leq \left(1 - \frac{1}{\alpha \text{height}(H)} \right)^\ell \\ &\leq e^{-O(\log n)} \\ &\leq \frac{1}{n^3} \end{aligned}$$

We conclude that the expected cost of F' is at most $O(\log N \log n \text{OPT})$, and since, by Lemma 22, $\text{cost}(F^*) \leq \text{cost}(F')$, F^* is $O(\log N \log n)$ -approximate in expectation. By Theorem 23, the running time of our algorithm is


$$N^{O(d)} (\log N)^{O(d)^{(d-1)/2} O(\log N)} = N^{O(d)^{(d-1)/2} \log \log N}.$$

This completes the proof of Theorem 16.

Dynamic Approximate Multiplicatively-Weighted Nearest Neighbors

Boris Aronov  

Department of Computer Science and Engineering, Tandon School of Engineering,
New York University, Brooklyn, NY, USA

Matthew J. Katz  

Department of Computer Science, Ben Gurion University of the Negev, Beer Sheva, Israel

Abstract

We describe a dynamic data structure for approximate nearest neighbor (ANN) queries with respect to multiplicatively weighted distances with additive offsets. Queries take polylogarithmic time, while the cost of updates is amortized polylogarithmic. The data structure requires near-linear space and construction time.

The approach works not only for the Euclidean norm, but for other norms in \mathbb{R}^d , for any fixed d .

We employ our ANN data structure to construct a faster dynamic structure for approximate SINR queries, ensuring polylogarithmic query and polylogarithmic amortized update for the case of non-uniform power transmitters, thus closing a gap in previous state of the art.

To obtain the latter result, we needed a data structure for dynamic approximate halfplane range counting in the plane. Since we could not find such a data structure in the literature, we also show how to dynamize one of the known static data structures.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Nearest neighbors, Approximate nearest neighbors, Weighted nearest neighbors, Nearest neighbor queries, SINR queries, Dynamic data structures

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.11

Funding Supported by Grants 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.

Boris Aronov: Supported in part by NSF grant CCF 15-40656.

Matthew J. Katz: Supported in part by Grant 1884/16 from the Israel Science Foundation.

Acknowledgements We wish to thank Pankaj K. Agarwal for his help and encouragement, and David Mount for a clarification regarding the approximating polytope in [8].

1 Introduction

Nearest-neighbor (NN) search is a fundamental problem that has received much attention in a variety of research fields, such as databases, machine learning, and statistics. It is a central ingredient in clustering and pattern matching algorithms, as well as in numerous retrieval and recommendation systems. In this study, we are interested in the multi-shot version of the problem: Given a set P in some space S and a distance function $d(\cdot, \cdot)$ on S , preprocess P for nearest-neighbor queries, where such a query is specified by an element $q \in S$ and the goal is to return an element of P that is nearest to q under d among all elements in P . In this study, we shall restrict our attention to the case where the input set consists of points in \mathbb{R}^d and the distance is measured by a weighted version of a metric derived from a norm.

Let $P = \{p_1, \dots, p_n\}$ be a set of n points in \mathbb{R}^d , such that each point $p_i \in P$ is associated with a positive real weight w_i and a non-negative real weight a_i , and let $\|\cdot\|$ denote any norm on \mathbb{R}^d . The distance from $q \in \mathbb{R}^d$ to $p_i \in P$ is now defined as $d_{W,A}(q, p_i) := w_i d(q, p_i) + a_i$, where $d(q, p_i) := \|q - p_i\|$ and W and A are the sets of multiplicative



© Boris Aronov and Matthew J. Katz;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 11; pp. 11:1–11:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and additive weights, respectively. We consider nearest-neighbor search in this setting, that is, following a preprocessing stage, service a sequence of queries of the form: given a query point q , return a point $p_i \in P$ realizing $\min_{p_i \in P} d_{W,A}(q, p_i)$. Notice that when $\|\cdot\|$ is the Euclidean norm, we obtain the *additively weighted* Euclidean nearest-neighbor search problem, if $w_1 = \dots = w_n = 1$, the *multiplicatively weighted* Euclidean nearest-neighbor search problem, if $a_1 = \dots = a_n = 0$, and the combined *additively and multiplicatively weighted* Euclidean nearest-neighbor search problem, otherwise.

More specifically, we consider *dynamic* approximate nearest-neighbor (ANN) search in this setting, where, in addition, points may be inserted into and deleted from P over time. An approximate nearest neighbor of q in P is a point $p_j \in P$, such that $d_{W,A}(q, p_j) \leq (1 + \varepsilon)d_{W,A}(q, p_i)$, where p_i is a nearest neighbor of q in P and $\varepsilon > 0$ is a prespecified parameter. The reason for considering approximate, rather than exact, nearest-neighbor search is that already in the plane the induced (multiplicatively-weighted) Voronoi diagram may have complexity $\Theta(n^2)$ [11], so constructing it explicitly for sufficiently large values of n is impractical. On the other hand, we want to be able to handle queries efficiently, typically in time polylogarithmic in n , thus we resort to approximate queries.

Although there are known solutions for approximate nearest-neighbor search (see below), we are not aware of any published technique suitable for efficient updates of P , where each such update involves the adjustment of the data structure following an insertion or a deletion of a point to/from P . We present a data structure for dynamic approximate nearest-neighbor search in this general setting. The data structure is of size $O(tn \text{ polylog } n)$, which is also the bound on its construction time; it supports approximate nearest-neighbor queries in $O(t \text{ polylog } n)$ time and it can be updated in $O(t \text{ polylog } n)$ amortized time, where $t = \sqrt{1/\varepsilon}$ for $d = 2$, $t = 1/\varepsilon$ for $d = 3$, and in general $t = 1/\varepsilon^{(d-1)/2}$ for $d \geq 2$.

The case where $\|\cdot\|$ is the Euclidean norm and $a_1 = \dots = a_n = 0$, namely, dynamic approximate multiplicatively weighted Euclidean nearest-neighbor search, is of special interest, since (i) it applies to many practical scenarios, and (ii) it is more difficult than some of the other common cases, such as in the additively weighted scenario. In Section 5, we show that our data structure for this case enables us to significantly improve the best known solution to the most general version of the approximate SINR (signal-to-interference-plus-noise ratio) query problem. In this version, we are given a set of simultaneous transmitters, where each transmitter is represented by a point in the plane and has its own power level. Moreover, transmitters may appear or disappear over time. The goal is to construct a dynamic data structure, so that given a receiver (i.e., a point) q , one can (approximately) determine which transmitter (if any) is received at q , according to the SINR model (see Section 5). After a preprocessing stage of near-linear time, in which a data structure of near-linear size is constructed, we can answer a query in polylogarithmic time and insert or delete a transmitter in polylogarithmic amortized time. In contrast, in the best previous solution, the query time was roughly \sqrt{n} [4]. Our algorithm is randomized, with performance guarantees holding with high probability; see Theorem 5.

To obtain the latter result, we also need a data structure for handling approximate halfplane range counting queries in a dynamic setting in two dimensions. Since we could not find such a data structure in the literature, we also show how to dynamize one of the known static data structures. We include the description for completeness.

Previous work

There is extensive literature, both in computational geometry and in other fields, on nearest-neighbor (NN) and approximate nearest-neighbor (ANN) data structures; see, for example, [9,10,15,17,18,20,26], and the book [30] for a database perspective. (As we assume throughout

that the dimension d is a small constant, the work addressing nearest-neighbor queries in high dimension [3] is out of scope for this summary.) This includes structures under the Euclidean metric and other metrics, supporting both exact and approximate queries, possibly in a dynamic setting. Moreover, many of these structures can also accommodate (with minor adjustments) additive weights (i.e., a non-negative weight a_i is added to the distance between q and p_i). However, multiplicative weights are much more challenging, and we are only aware of two teams of researchers who studied this case in low dimension, as well as more general ones, albeit not in a dynamic setting.

Next we discuss the work of these two teams. Recall first that, if we require a near-linear-size structure, then we need to resort to approximation. Har-Peled and Kumar [22] were the first to present a near-linear-size data structure with logarithmic query time for approximate multiplicatively weighted Euclidean nearest-neighbor search. Actually, their result is for a much more general problem, where the input is a set \mathcal{F} of d -variate functions, satisfying several rather weak conditions, and the goal is to construct a data structure, so that given a query point $q \in \mathbb{R}^d$, one can return a function in \mathcal{F} whose value at q is at most $1 + \varepsilon$ times the minimum value attained at q by any function in \mathcal{F} . Now, if we define $f_i(q) = w_i \|q - p_i\|$, we get approximate multiplicatively weighted nearest-neighbor search. The bounds that they obtain in this case are $O(\frac{n}{\varepsilon^{2(d+1)}} \log^{d+2} n + \frac{n}{\varepsilon^{d(d+1)}})$ for the structure size, $O(\frac{n}{\varepsilon^{2(d+1)}} \log^{2d+3} n + \frac{n}{\varepsilon^{d(d+1)}})$ for the construction time, and $O(\log \frac{n}{\varepsilon})$ for the query time.

Subsequently, Abdelkader et al. [1] presented a data structure, based on *convexification*, for approximate multiplicatively weighted Euclidean nearest-neighbor search (see also [28]). Their result is actually for *scaling distance functions*, which also include the Minkowski distance (provided the unit ball is fat and smooth) and the Mahalanobis distance. It improves the bound on the structure size to $O(\frac{n \log \frac{1}{\varepsilon}}{\varepsilon^{d/2}})$, while retaining the $O(\log \frac{n}{\varepsilon})$ bound on the query time, thus almost matching (up to a $\log \frac{1}{\varepsilon}$ factor) the best known bounds for approximate Euclidean nearest-neighbor search [9]. However, it is not clear how fast this structure can be constructed.

Our results

- (i) In Section 2 we describe a near-linear-size data structure that supports queries for approximate Euclidean nearest neighbors with multiplicative weights in the plane. The query and the amortized update times are both polylogarithmic.
- (ii) In Section 3 we explain how to handle other norms and also a combination of additive and multiplicative weights.
- (iii) We point out, in Section 4, that a further generalization extends the results of Section 3 to higher dimensions.
- (iv) We show how the data structures described above allow approximate dynamic SINR queries with logarithmic query times and amortized logarithmic time updates; see Section 5. The data structure can accommodate non-uniform transmitter powers; to the best of our knowledge it was not known how to achieve this performance for the case of non-uniform powers.
- (v) To facilitate the latter result, we also show in Section 6 how to dynamize with minimal overhead a data structure for approximate halfplane range counting queries.

The purpose of this paper is to demonstrate the existence of several related data structures, of near-linear size and construction time, polylogarithmic query time, and polylogarithmic amortized update time. We did not make any effort to optimize the performance of these structures and often used the “classical” well known tools as the building blocks. More

precisely, it was not our objective to optimize polylogarithmic factors; this may be a worthwhile project on its own. Hence, with one or two exceptions, we leave the precise polylogarithmic factors unspecified.

2 Multiplicatively weighted Euclidean nearest neighbors

For ease of presentation, we start with the basic case, that is, we consider the Euclidean norm in the plane with multiplicative weights. Generalizations to other norms, higher dimensions, and more complicated weights are described in some detail subsequently.

Consider a set $P = \{p_1, \dots, p_n\}$ of n points in the plane. The points have an associated (multiplicative) weight function $W: P \rightarrow \mathbb{R}^+$; the (positive) weight of p_i will be denoted by $w_i := W(p_i)$. The (*multiplicatively weighted*) distance from $q \in \mathbb{R}^2$ to $p \in P$ is defined as $d_W(q, p) := W(p) \cdot d(q, p)$, where $d(p, q) := \|p - q\|$ stands for the Euclidean distance and $\|\cdot\|$ – for the Euclidean norm, in this section. A *multiplicatively weighted Euclidean nearest neighbor* $N(q) = N(q; P, W)$ of $q \in \mathbb{R}^2$ is a point $p \in P$ realizing $\min_{p \in P} d_W(q, p)$. For brevity, we will refer to $N(q)$ simply as the *nearest neighbor* of q (in P). (When we need to compute $N(q)$, if more than one point realizes the minimum distance, we are allowed to pick one of them arbitrarily.) We are interested in building a data structure that supports fast nearest-neighbor queries: preprocess a given pair (P, W) , so that $N(q) = N(q; P, W)$ can be computed quickly for any query point $q \in \mathbb{R}^2$; “quickly” here and hereafter means “in time polylogarithmic in $n = |P|$.”

A natural approach to preprocessing for fast nearest-neighbor queries is to consider the induced Voronoi diagram; unfortunately, in presence of multiplicative weights, the Voronoi diagram may have complexity $\Theta(n^2)$ [11] and is thus not a viable option for a compact data structure. We therefore focus on looking for an *approximate (multiplicatively weighted Euclidean) nearest neighbor* $\tilde{N}(q)$ for q , which is a point $p \in P$ such that $d_W(q, p) \leq (1 + \varepsilon)d_W(q, N(q))$; of course, by definition, for this point p , $d_W(q, N(q)) \leq d_W(q, p)$.

Terminology and notation

We use $f(x) = \text{poly}(x)$ to indicate that there exists a constant $a > 0$ such that $f(x)$ is in $O(x^a)$; similarly, $g(x, y) = \text{poly}(x, y)$ means that there exist two constants $a, b > 0$ such that $g(x, y)$ is in $O(x^a y^b)$.

We say that an event holds *with high probability*, if it holds with probability $1 - 1/n^c$ for a suitably large $c > 0$.

Finally, we say that a quantity k' $(1 + \varepsilon)$ -*approximates* a quantity k , if $(1 - \varepsilon)k \leq k' \leq (1 + \varepsilon)k$.

Problem statement

DYNAMIC APPROXIMATE MULTIPLICATIVELY WEIGHTED EUCLIDEAN NEAREST NEIGHBORS IN THE PLANE

Input: an n -point set P in the plane with positive weights W as above and $\varepsilon > 0$

Output: a data structure

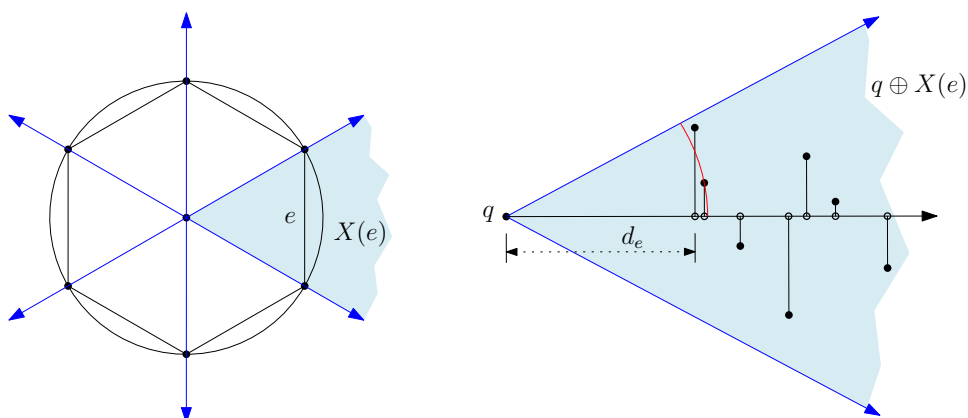
- of size $O(n \text{poly}(\log n, 1/\varepsilon))$,
- constructed in time $O(n \text{poly}(\log n, 1/\varepsilon))$,
- that supports approximate nearest-neighbor queries in time $\text{poly}(\log n, 1/\varepsilon)$, and
- insertions and deletions for P in amortized time $\text{poly}(\log n, 1/\varepsilon)$.

Outline of our approach

We use the standard approximation that replaces the unit disk U of the Euclidean norm by a regular k -gon U_k inscribed in it, for a suitable choice of $k = \Theta(\varepsilon^{-1/2})$. More precisely, we choose k , so that for any point $z \in U$, its closest point of U_k on the segment oz connecting z to the origin o is at Euclidean distance at most ε from z : $\forall z \in U: d(z, oz \cap U_k) \leq \varepsilon$.

The norm $\|\cdot\|_k$ defined by U_k as the unit disk is a $(1 + \varepsilon)$ -approximation of the Euclidean norm. We will denote the distance induced by this norm by $d_k(\cdot, \cdot)$ and its multiplicatively weighted version by $d_{k,W}(\cdot, \cdot)$. We thus have $d(q, P) \leq d_k(q, P) \leq (1 + \varepsilon)d(q, P)$, where $\text{dist}(q, P) := \min_{p \in P} \text{dist}(q, p)$ and dist stands for either d_k or d .

With each edge e of U_k we associate a wedge $X(e)$ centered at the origin and delimited by the rays from the origin through the endpoints of e ; see Figure 1 (left).



■ **Figure 1** Left: The regular k -gon U_k , for $k = 6$, and the wedges $X(e)$. Right: Determining the distance d_e for query point q .

Given a point q , $d_k(q, P)$ can be computed as follows: for each edge e of U_k , let $P_e := P \cap (q \oplus X(e))$, or, in words, consider the subset P_e of points of P (if any) that lie in the wedge $X(e)$ translated to q ; see Figure 1 (right). Now project the points of P_e onto the central ray of $q \oplus X(e)$ and compute d_e as the distance from q to the closest projected point (set $d_e = +\infty$ if $P_e = \emptyset$). Then $d_k(q, P) = \min_e d_e$. (Up to this point, our approach is quite similar to that of Kapoor and Smid [26], who studied the basic (i.e., unweighted Euclidean) version of dynamic approximate nearest neighbors.)

We will make use of the fact that computing $d_{k,W}(q, P)$ is a *decomposable* problem, since $d_{k,W}(q, P_1 \cup P_2) = \min\{d_{k,W}(q, P_1), d_{k,W}(q, P_2)\}$.

Approximately nearest in a wedge

We first examine a fundamental subproblem. Fix an edge e of U_k and denote its associated wedge $X(e)$ by X . Let $R \subseteq P$ be a subset of the given points and consider only those queries q for which $R \subseteq q \oplus X$. We will describe a data structure for determining, given q , its nearest neighbor $N(q; R, W)$ in R , which we will continue to denote $N(q)$, slightly abusing the notation.

After a suitable rotation, we may suppose that the central ray of X coincides with the positive x -axis; recall that the apex of X is the origin; refer to Figure 1.

By definition, if $q = (x_q, y_q)$, the function $d_{k,W}(q, p)$, for $p \in q \oplus X$, is simply $W(p) \cdot |x_q - x_p| = W(p)(x_p - x_q)$, since $p \in q \oplus X$ implies $x_p \geq x_q$. Therefore $N(q)$ is precisely the point $p \in R$ whose corresponding function achieves $\min_{p \in R} W(p)(x_p - x_q)$. One way to view this

11:6 Dynamic Approximate Multiplicatively-Weighted Nearest Neighbors

operation is to consider the graphs of functions $f_p(x) = W(p)(x_p - x)$, for $p \in R$, define the *lower envelope* $L: \mathbb{R} \rightarrow \mathbb{R}$ by $x \mapsto \min_{p \in R} f_p(x)$, and then perform *vertical ray-shooting* in (the graph of) L , namely given $x = x_q$, identify (the graph of) the function that achieves the value $L(x_q)$ at this x .

Being a lower envelope of a set of lines, the graph of L is the boundary of the intersection of a set of lower halfplanes in the plane, and so a monotone concave chain. In a static setting, it can be precomputed and stored in, say, an array, to facilitate vertical ray shooting via binary search on the x -coordinate of the query point q . In a dynamic setting, the intersection of lower halfplanes is dual to the upper convex hull of the dual points of the lines bounding the halfplanes. It can be stored, say, in the data structure of Overmars and van Leeuwen [29], that supports $O(\log^2 n)$ time updates and $O(\log n)$ time queries, where $n = |R|$.¹ A vertical ray shooting query corresponds in the dual to finding the extreme point of the hull in a given direction, one of the standard queries supported by the data structure.

The general case

We build k data structures, one for each wedge $X = X(e)$. For each X , let u and v be the directions of its bounding rays and m be the direction of its central ray. We build a three-level range-search tree structure on the points of P , where the first two levels sort points of P in the directions orthogonal to u and to v , respectively. The effect of this is that a query with a point q will return the points of $P \cap (q \oplus X)$ as a disjoint union of $O(\log^2 n)$ canonical subsets on the second level of the structure; here we use the decomposability of the queries we are interested in. On the bottommost level, for each canonical subset R , we build the nearest-in-a-wedge data structure described above, with the distinguished direction being m , the central direction of X (rather than that of the positive x -axis).

Among the $O(k \log^2 n)$ points returned from $O(\log^2 n)$ canonical subsets in each of the k structures, we pick the one that is closest to q in the distinguished direction as the approximate nearest neighbor $\tilde{N}(q)$; this produces the correct answer, since our query is decomposable.

Now to address the efficiency of the updates. Given m points, the corresponding nearest-in-a-wedge structure for them can be built from scratch in $O(m \log m)$ time. This structure is fully dynamic by construction and hence the only concern is maintaining the balance in the upper levels of the overall range tree. If upper levels of the range-search structure are implemented as $\text{BB}[\alpha]$ trees, amortized rebalancing costs are polylogarithmic, as implied by Theorem 5 in section III.5.1 of Mehlhorn's monograph [27, pages 198–199].

We summarize in the following theorem:

► **Theorem 1.** *For any n -point set P in the plane and $\varepsilon > 0$, we can construct a dynamic data structure for $(1 + \varepsilon)$ -approximate multiplicatively weighted Euclidean nearest-neighbor queries in P*

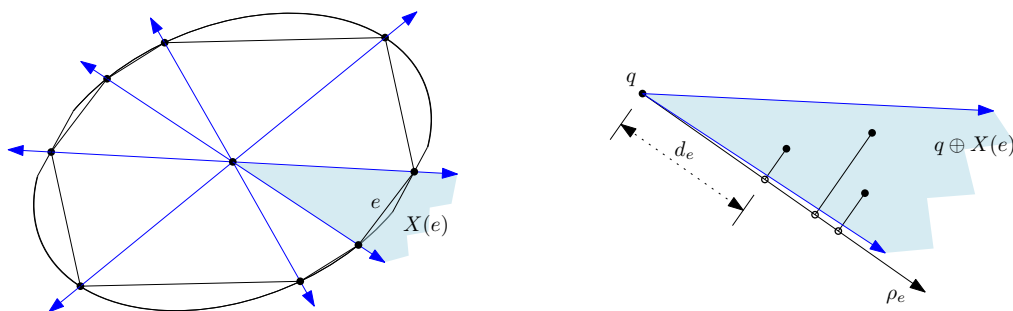
- *of size $O(n \text{poly}(\log n, 1/\varepsilon))$,*
- *in time $O(n \text{poly}(\log n, 1/\varepsilon))$,*
- *supporting $O(\text{poly}(\log n, 1/\varepsilon))$ -time queries and*
- *$O(\text{poly}(\log n, 1/\varepsilon))$ amortized time updates.*

¹ Recall that, in this work, a conservative choice of the data structure that guarantees performance polylogarithmic in n and polynomial in $1/\varepsilon$ but does not necessarily attempt to achieve optimal performance is sufficient.

► Remark. As an illustration, we calculate the actual performance characteristics using admittedly suboptimal “classical” building blocks: With the Overmars-van Leeuwen data structure for dynamic convex hulls with $O(\log n)$ time queries and $O(\log^2 n)$ time updates, and with $k = O(\varepsilon^{-1/2})$ top-level structures, we obtain space and time $O((n/\sqrt{\varepsilon}) \log^2 n)$, query time of $O(\varepsilon^{-1/2} \log^3 n)$ and amortized update time of $O(\varepsilon^{-1/2} \log^4 n)$.

3 Other norms

In this section we outline how the results of Section 2 generalize to an arbitrary norm on \mathbb{R}^2 . Indeed, for any norm $\|\cdot\|$, consider its unit disk $D := \{p \in \mathbb{R}^2 : \|p\| \leq 1\}$. D is a compact centrally symmetric convex set with non-empty interior.² We proceed as follows: by a theorem of John [24] there exists a pair of concentric ellipses, one contained in D and one containing it, which are scaled copies of each other, with a scaling factor of at most 2. Consider an affine transformation that turns the inner ellipse into the Euclidean unit disk and the outer one into a Euclidean disk of radius at most 2. Apply this transformation to the unit ball D , to all of \mathbb{R}^2 , and to P for the subsequent processing. It is easy to check that this transformation leaves the answer to the (both exact and approximate) nearest-neighbor problem unchanged. With a slight abuse of notation, we will continue to refer to objects after the transformation by the same symbols.



■ **Figure 2** Left: Approximating D by a centrally-symmetric convex k -gon U_k , and the wedges $X(e)$. Right: Determining the distance d_e for query point q .

After the transformation, the unit ball D is “fat” in the sense that it is sandwiched between two concentric disks with a bounded ratio of radii. It therefore can be approximated by a centrally-symmetric convex polygon $U_k \subseteq D$ with $k = \Theta(\varepsilon^{-1/2})$ sides, just as the Euclidean disk in Section 2 (except that now U_k is not necessarily a regular polygon) [14], see Figure 2 (left). U_k approximates D in the sense that, for any direction ρ , the ratio between the distances of the farthest points of D and U_k along the ray from the origin in direction ρ is at most $1 + \varepsilon$. This implies that replacing D by U_k as the unit disk distorts the distance by a factor of at most $1 + \varepsilon$, as desired.

We now proceed as before: We associate each of U_k ’s edges e with the wedge $X(e)$ formed by the rays from the origin passing through e ’s endpoints. We use a shifted version $q \oplus X(e)$ of $X(e)$ to compute the “approximately closest” point of $P_e = P \cap (q \oplus X(e))$ from q , where the distance from q is measured along a ray ρ_e emanating from q and orthogonal to e (ρ_e need not be the central ray of $X(e)$ any longer), see Figure 2 (right). That is, we project the points of P_e onto ρ_e , and measure the distance d_e from q to the first projected point along ρ_e . The remainder of the argument and the data structure remain unchanged.

² D with empty interior would allow non-zero vectors of zero norm, violating the standard norm definition.

We summarize in the following theorem.

► **Theorem 2.** *For any norm in \mathbb{R}^2 , there exists a dynamic data structure supporting $(1 + \varepsilon)$ -approximate multiplicatively weighted nearest-neighbor queries with the same performance as the data structure in Theorem 1.*

Notice that at no extra cost, we can support more general approximate nearest-neighbor queries where both multiplicative and positive additive weights are present. More precisely, let $A: P \rightarrow \mathbb{R}_0^+$ be non-negative additive weights on P and modify the distance from q to $p \in P$ to mean $A(p) + W(p) \cdot \|q - p\|$, where $\|\cdot\|$ is any norm as above. (Setting $A(p) \equiv 0$ recovers the multiplicative-weights-only version of the problem, and setting $W(p) \equiv 1$ recovers the familiar additive-weights-only version.) Indeed the functions $f_p(x)$ as defined above become $W(p)(x_p - x) + A(p)$, i.e., remain linear, and the argument goes through verbatim.

► **Theorem 3.** *For any norm in \mathbb{R}^2 , we can construct a dynamic data structure supporting $(1 + \varepsilon)$ -approximate additively and multiplicatively weighted nearest-neighbor queries with the same performance guarantees as the data structure in Theorem 1.*

► **Remark.** In fact, our result is slightly more general. It applies to any asymmetric norm, whose unit disk is a not necessarily centrally-symmetric convex set with non-empty interior, provided that this disk is sandwiched between two Euclidean disks centered at the origin, with a constant ratio c of the outer and inner radii. In this situation the unit disk is still approximable by a $D(c)\sqrt{\varepsilon}$ -gon where $D(c)$ is a constant that depends on c . The argument goes through essentially without modification, taking care that the distance from p to q need not be equal to the distance from q to p .

4 Higher dimensions

We now outline the fairly standard procedure to extend our results from Sections 2 and 3 from the plane to any fixed dimension $d \geq 2$; we assume d is a small constant. We proceed as in Section 3. Consider a norm $\|\cdot\|$ on \mathbb{R}^d and let D be its unit ball – a compact centrally symmetric convex set with non-empty interior. We consider the Löwner-John ellipsoids for D that approximate it up to a factor of d , in the following sense: The two ellipsoids are concentric, one is contained in D while the other contains it, and the outer ellipsoid is a scaled copy of the inner one, with a scaling factor of at most d [24]. We now apply an affine transformation to the entire space turning the inner ellipsoid into the Euclidean unit ball and proceed with the transformed problem; transforming the space, the input point set P , and the unit ball D does not affect distance measurements according to $\|\cdot\|$. Slightly abusing the notation, we use the same symbols referring to the objects after the transformation. After the transformation, D is “fat” in the sense that it is sandwiched between two concentric balls with radius ratio of at most d .

Bronshsteyn and Ivanov [14] proved that D in this situation (see also [21] for a compact self-contained proof) can be $(1 + \varepsilon)$ -approximated in the Hausdorff metric by a polytope with $O(1/\varepsilon^{(d-1)/2})$ vertices. Dudley [19] showed that there exists such an approximating polytope with $O(1/\varepsilon^{(d-1)/2})$ facets. Much more recently Arya et al. [8] proved that one can construct such an approximating polytope whose total number of faces of all dimensions is $k := O(1/\varepsilon^{(d-1)/2})$;³ moreover, one can assume that the resulting polytope U_k is simplicial,

³ This is the best possible answer, as $\Omega(1/\varepsilon^{(d-1)/2})$ faces are sometimes required. In fact, this is the case for the Euclidean norm.

that is each of its faces is a simplex. As in the plane, U_k approximates D in the sense that the norm $\|\cdot\|_k$ defined by U_k as the unit ball has the property that $\|p\| \leq \|p\|_k \leq (1 + \varepsilon)\|p\|$. Since U_k is a simplicial polytope with at most k facets, its boundary is already triangulated by at most k simplices.

We now repeat the reasoning of Section 3 for each facet Δ of U_k (which is a simplex): Δ is associated with a (simplicial) cone of directions $X = X(\Delta)$, and for any point of $q \in \mathbb{R}^d$, the distance $d_k(q, P(q, \Delta))$ from q to the closest point of $P(q, \Delta) := P \cap (q \oplus X)$ can be computed exactly by using a suitable range-searching data structure, as $P(q, \Delta)$ is the intersection of P with the simplicial cone $q \oplus X$ with fixed and known directions of its d bounding hyperplanes, and the distance $\|p - q\|_k$ for $p \in P(q, \Delta)$ is the distance from q to the projection of p onto the ray $\rho = \rho(\Delta)$ emanating from q and orthogonal to the hyperplane containing Δ . Employing a suitable dynamic range structure, *à la* Section 3, completes the description; the only difference is that there are d upper levels in the structure to handle the narrowing to the cone X . Note that the bottommost level still handles dynamic vertical ray shooting in *two* dimensions.

The query structure has to be built for each of the at most k cones.

► **Theorem 4.** *For any norm in \mathbb{R}^d , $d \geq 2$, any n -point set P , and $\varepsilon > 0$, there exists a dynamic data structure for approximate additively-and-multiplicatively weighted nearest-neighbor queries, with performance guarantees as follows:*

- *data structure has size $O(kn \text{ polylog } n)$ and it can be constructed in this time,*
- *updates in $O(k \text{ polylog } n)$ amortized time, and*
- *queries in $O(k \text{ polylog } n)$ time,*

where $k = \Theta(1/\varepsilon^{(d-1)/2})$. In particular, when $d = 2$, $k = \Theta(1/\sqrt{\varepsilon})$, and when $d = 3$, $k = \Theta(1/\varepsilon)$.

5 Dynamic SINR queries: An application

We now explain how to use the ANN data structure described in Section 2 to speed up dynamic approximate SINR (signal-to-interference-plus-noise ratio) queries for non-uniform power transmitters. We first formulate the problem, give some background and history, and then outline the application.

Problem setup and formulation

Let $S = \{s_1, \dots, s_n\}$ be a set of n transmitters (distinct points in the plane), and let $p_i \geq 0$ denote the transmission power of s_i , for $i = 1, \dots, n$. Let q be a receiver (a point in the plane). According to the SINR model, q receives s_i if and only if

$$\frac{\frac{p_i}{d(q, s_i)^\alpha}}{\sum_{j \neq i} \frac{p_j}{d(q, s_j)^\alpha} + N} =: \text{sinr}(q, s_i) \geq \beta,$$

where $\alpha \geq 1$ and $\beta > 1$ are given constants, N is a given constant representing the background noise, and $d(a, b)$ is the Euclidean distance between points a and b . The quantity $\frac{p_i}{d(q, s_i)^\alpha}$ is the strength of the signal of the i th transmitter (located at s_i with power p_i) as measured at the receiver q , where α is the *path-loss* parameter. In words, the above inequality states that q receives s_i if and only if s_i 's signal (at q) is at least β times stronger than the combined signal of all other transmitter and the noise; see, for example, [31].

Observe that, since $\beta > 1$, q may receive at most one transmitter – the one for which the value $\frac{1}{p_i^{1/\alpha}} d(q, s_i)$ is minimum. Thus, in the *uniform power* situation, where $p_1 = p_2 = \dots = p_n$, one needs to test reception at q for the (Euclidean) nearest neighbor of q in S , while in

11:10 Dynamic Approximate Multiplicatively-Weighted Nearest Neighbors

the *non-uniform power* situation, where transmitter powers vary, one needs to test reception for the multiplicatively weighted nearest neighbor of q . An *SINR query* is therefore: Given a receiver q , find the sole transmitter $s \in S$ that may be received by q and determine whether $\text{sinr}(q, s) \geq \beta$.

Since it seems unlikely that one can answer an SINR query *exactly* in significantly sublinear time with the help of a near-linear-size data structure, as the degree of the polynomials involved is high, the relevant research has focused on preprocessing to facilitate efficient *approximate* SINR queries.

Given $\varepsilon > 0$, an *approximate SINR query* is: Given a receiver q , find the sole transmitter s that may be received by q and return a value $\tilde{\text{sinr}}(q, s)$, such that $(1 - \varepsilon)\text{sinr}(q, s) \leq \tilde{\text{sinr}}(q, s) \leq (1 + \varepsilon)\text{sinr}(q, s)$. Thus, unless $(1 - \varepsilon)\beta \leq \tilde{\text{sinr}}(q, s) < (1 + \varepsilon)\beta$, the value $\tilde{\text{sinr}}(q, s)$ enables us to determine definitely whether or not s is received by q .

Given S , α, β , and N , as above, and $\varepsilon > 0$, there are four natural problems to consider, depending on whether the transmission powers are uniform or not, and whether the setting is static or dynamic (i.e., transmitters may be added to or deleted from S). In each of these problems, the goal is to devise efficient algorithms for handling approximate SINR queries, after some preprocessing of total near-linear time, where in the dynamic setting one also needs to handle updates (i.e., insertions and deletions of transmitters) efficiently.

Some history

For all but the most general problem, satisfactory solutions already exist, where by satisfactory we mean near-linear time preprocessing, polylogarithmic-time approximate queries and amortized polylogarithmic-time updates (in the dynamic setting) [4]; the dependence on $1/\varepsilon$ is polynomial (see also [6, 12, 25]).

In this section, we obtain a satisfactory solution also for the most general problem. That is, we show that even if the transmission powers are non-uniform, it is possible to construct in near-linear time a dynamic data structure that supports polylogarithmic-time approximate SINR queries and amortized polylogarithmic-time updates.

Tools required

A *static* structure supporting fast approximate SINR queries was presented in [4, section 6]. With the goal of generalizing it to support dynamic updates, inspecting the proposed algorithm, we discover that dynamizing the following two structures is sufficient for achieving our objective: The first one is a dynamic approximate multiplicatively weighted nearest-neighbor structure, which is precisely the problem in Section 2.

The second issue is handling *approximate halfplane range counting* queries in a dynamic setting; the problem is stated formally and a reasonably efficient solution is sketched in Section 6.

Examining the static algorithm of [4, section 6], we note that the approximate halfplane range counting data structure is used as the bottom level of a multilevel dynamic orthogonal range-searching structure, where approximate halfplane range counting queries are applied to bottommost canonical subsets of points. Replacing the static structure by the dynamic one from Section 6 completes the dynamization of the algorithm.⁴

⁴ In slightly more detail, if we use $\text{BB}[\alpha]$ trees as the basis of the orthogonal range counting structure, then, by Theorem 5 in section III.5.1 and subsequent discussion in [27, pages 198–199], the amortized cost of an update remains polylogarithmic.

We thus summarize our result.

► **Theorem 5.** *Given a set S of n transmitters in the plane, with their power transmission levels, parameters α , β , N , and an approximation constant $\varepsilon > 0$, one can preprocess S into a data structure of worst-case size $O(n \text{poly}(\log n, 1/\varepsilon))$ that supports approximate SINR queries in time $O(\text{poly}(\log n, 1/\varepsilon))$ and updates in amortized time $O(\text{poly}(\log n, 1/\varepsilon))$.*

The construction time is $O(n \text{poly}(\log n, 1/\varepsilon))$ with high probability, and query and update times are with high probability, assuming the number of queries is polynomial.

► **Remark.** As already noted in [4], the same approach generalizes to any fixed dimension $d > 2$ with increased overhead in the dependence on $1/\varepsilon$ and higher polylogarithmic factors. The algorithm still only needs dynamic approximate counting *in the plane* and results from Section 4 provide the machinery for finding nearest neighbors.

6 Dynamic approximate halfplane counting

In this section we outline a solution to the following problem which is required to complete our algorithm for efficient dynamic SINR queries in Section 5. No effort has been made to optimize the performance of the data structure; this may be an interesting question in itself.

DYNAMIC APPROXIMATE HALFPLANE RANGE COUNTING: Given a set P of n points in the plane and a parameter ε , $0 < \varepsilon < 1$, preprocess P in $O(n \text{poly}(\log n, 1/\varepsilon))$ time so that, given a query halfplane h , a $(1 + \varepsilon)$ -approximation of the number $|P \cap h|$ can be returned in time $O(\text{poly}(\log n, 1/\varepsilon))$. Moreover, insertions and deletions can be processed in $O(\text{poly}(\log n, 1/\varepsilon))$ amortized time.

We sketch a Monte Carlo algorithm for this problem, where the $(1 + \varepsilon)$ -approximation is correct *with high probability* (i.e., with probability $1 - 1/n^c$, for a sufficiently large $c > 0$), provided one makes a polynomial number of queries, assuming that the updates to the data structure do not depend on the random choices made by the algorithm.

The easiest, though perhaps not the most efficient method to achieve our goal is to use the “black-box” reduction of Aronov and Har-Peled [5], who observed that an approximate range counting structure can be obtained, at the cost of multiplicative overhead of $\text{poly}(\log n, 1/\varepsilon)$ in construction time and space and in query time, from a data structure for *emptiness testing*. In our context, emptiness testing is, given a set P of points in the plane, preprocess it so that, for a query halfplane h , one can quickly check whether or not $h \cap P = \emptyset$. The nature of the reduction is constructing a number of emptiness-testing structures on randomly chosen subsets of P . In our case, emptiness testing is easily dynamized by using, say, the classical dynamic convex hull data structure of Overmars-van Leeuwen [29]. Random subsets generated by the reduction involve picking each point of P independently with a given probability and so can be updated efficiently on insertion into and deletion from P . The number of such subsets is $\text{poly}(\log n, 1/\varepsilon)$, so an update cannot affect too many of them, even in the worst case. (Some additional aspects of the structure depend on the value of n , but these can be handled, as is standard, by periodic rebuilds when n changes substantially.)

If one is interested in improving the performance of this data structure, there are several natural avenues of improvement:

- One can replace the dynamic convex hull structure of Overmars-van Leeuwen [29] by a faster alternative, such as [16] or [13, 23]; refer to the introduction of the latter reference for a more thorough literature survey.

- Using the *black-box* reduction from [5] likely affects efficiency, see more elaborate non-black-box work in [7] (though that work focuses on the higher-dimensional case and does not address supporting updates).
- Afshani and Chan [2] describe a more efficient Monte Carlo black-box reduction from approximate counting to *small-count* range searching, where a query needs to check if $|P \cap h|$ is small for a halfplane h , and if so, produce the correct answer, and otherwise output TOO LARGE. The algorithm essentially builds small-count structures for random samples of the input, of various sizes, conceptually similar to the reduction from [5]. As the counts required in the reduction are essentially of size $O(\text{poly}(1/\varepsilon) \log n)$, one could implement such a dynamic data structure in (near-)linear space with efficient updates using classical tools such as [29] or their subsequent improvements. One could even employ a dynamic *reporting* structure in this situation, provided it can interrupt the reporting if the query answer is too large; most reporting structures have such an ability.
- We conclude with an open problem: All the reductions described so far produce a Monte Carlo algorithm for dynamic approximate halfplane range counting. Is there a (simple and straightforward) Las Vegas algorithm with comparable performance? Or even a deterministic one?

7 Discussion and open problems

- Our data structures in Sections 2 through 4 can handle *farthest* rather than *nearest* neighbors, with simple and obvious modifications.
- It might be interesting to determine the best performance that can be attained in Theorems 1 through 4 with the current state-of-the-art data structures.
- The static data structures of Har-Peled and Kumar [22] and of Abdelkader et al. [1] cover a large class of (weighted) distance measures. In particular, both structures allow assigning different norms to different sites, which our approach cannot accommodate.

Refer to the end of Section 6 for some discussion and open problems related to the approximate halfplane range counting data structure described there.

References

- 1 Ahmed Abdelkader, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Approximate nearest neighbor searching with non-Euclidean and weighted distances. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 355–372. SIAM, 2019. doi:10.1137/1.9781611975482.23.
- 2 Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discret. Comput. Geom.*, 42(1):3–21, 2009. doi:10.1007/s00454-009-9177-z.
- 3 Alexandr Andoni and Piotr Indyk. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*, pages 1135–1155. Chapman and Hall/CRC, 2017.
- 4 Boris Aronov, Gali Bar-On, and Matthew J. Katz. Resolving SINR queries in a dynamic setting. *SIAM J. Comput.*, 49(6):1271–1290, 2020.
- 5 Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- 6 Boris Aronov and Matthew J. Katz. Batched point location in SINR diagrams via algebraic tools. *ACM Transactions on Algorithms*, 14(4):41:1–41:29, 2018.
- 7 Boris Aronov and Micha Sharir. Approximate halfspace range counting. *SIAM J. Comput.*, 39:2704–2725, 2010.

- 8 Rahul Arya, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Optimal bound on the combinatorial complexity of approximating polytopes. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 786–805. SIAM, 2020. doi:10.1137/1.9781611975994.48.
- 9 Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Optimal approximate polytope membership. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 270–288. SIAM, 2017. doi:10.1137/1.9781611974782.18.
- 10 Sunil Arya, Theodoros Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1):1:1–1:54, 2009. doi:10.1145/1613676.1613677.
- 11 Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recogn.*, 17:251–257, 1984. URL: [https://dx.doi.org/10.1016/0031-3203\(84\)90064-5](https://dx.doi.org/10.1016/0031-3203(84)90064-5).
- 12 Chen Avin, Yuval Emek, Erez Kantor, Zvi Lotker, David Peleg, and Liam Roditty. SINR diagrams: Convexity and its applications in wireless networks. *J. ACM*, 59(4):18:1–18:34, 2012. doi:10.1145/2339123.2339125.
- 13 Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 617–626. IEEE, 2002. See also [23].
- 14 Efim M. Bronshteyn and L.D. Ivanov. The approximation of convex sets by polyhedra. *Siberian Mathematical Journal*, 16(5):852–853, 1975.
- 15 Timothy M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998. doi:10.1007/PL00009390.
- 16 Timothy M Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *Journal of the ACM (JACM)*, 48(1):1–12, 2001.
- 17 Timothy M. Chan. Applications of Chebyshev polynomials to low-dimensional computational geometry. *J. Comput. Geom.*, 9(2):3–20, 2018. doi:10.20382/jocg.v9i2a2.
- 18 Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988. doi:10.1137/0217052.
- 19 Richard M Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.
- 20 Sariel Har-Peled. A replacement for Voronoi diagrams of near linear size. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 94–103. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959884.
- 21 Sariel Har-Peled and Mitchell Jones. Proof of Dudley’s convex approximation, 2019. arXiv:1912.01977.
- 22 Sariel Har-Peled and Nirman Kumar. Approximating minimization diagrams and generalized proximity search. *SIAM J. Comput.*, 44(4):944–974, 2015.
- 23 Riko Jacob and Gerth Stølting Brodal. Dynamic planar convex hull, 2019. arXiv:1902.11169.
- 24 Fritz John. Extremum problems with inequalities as subsidiary conditions. *R. Courant Anniversary Volume*, pages 187–204, 1948.
- 25 Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg. The topology of wireless communication. In *Proceedings 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 383–392, 2011. URL: <https://doi.acm.org/10.1145/1993636.1993688>, doi:10.1145/1993636.1993688.
- 26 Sanjiv Kapoor and Michiel H. M. Smid. New techniques for exact and approximate dynamic closest-point problems. *SIAM J. Comput.*, 25(4):775–796, 1996. doi:10.1137/S0097539793259458.
- 27 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. doi:10.1007/978-3-642-69672-5.

11:14 Dynamic Approximate Multiplicatively-Weighted Nearest Neighbors

- 28 David M. Mount. New directions in approximate nearest-neighbor searching. In Sudebkumar Prasant Pal and Ambat Vijayakumar, editors, *Algorithms and Discrete Applied Mathematics - 5th International Conference, CALDAM 2019, Kharagpur, India, February 14-16, 2019, Proceedings*, volume 11394 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2019. doi:10.1007/978-3-030-11509-8_1.
- 29 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.
- 30 A. N. Papadopoulos and Y. Manolopoulos. *Nearest Neighbor Search: A Database Perspective*. Springer US, 2005.
- 31 Theodore S. Rappaport. *Wireless Communications – Principles and Practice*. Prentice Hall, 1996.

MaxSAT with Absolute Value Functions: A Parameterized Perspective

Max Bannach  

Institute for Theoretical Computer Science, Universität zu Lübeck, Germany

Pamela Fleischmann  

Department of Computer Science, Christian-Albrechts-Universität zu Kiel, Germany

Malte Skambath  

Department of Computer Science, Christian-Albrechts-Universität zu Kiel, Germany

Abstract

The natural generalization of the Boolean satisfiability problem to optimization problems is the task of determining the maximum number of clauses that can simultaneously be satisfied in a propositional formula in conjunctive normal form. In the *weighted maximum satisfiability problem* each clause has a positive weight and one seeks an assignment of maximum weight. The literature almost solely considers the case of *positive* weights. While the general case of the problem is only restricted slightly by this constraint, many special cases become trivial in the absence of *negative* weights. In this work we study the problem *with* negative weights and observe that the problem becomes computationally harder – which we formalize from a parameterized perspective in the sense that various variations of the problem become $W[1]$ -hard if negative weights are present.

Allowing negative weights also introduces new variants of the problem: Instead of maximizing the sum of weights of satisfied clauses, we can maximize the *absolute value* of that sum. This turns out to be surprisingly expressive even restricted to *monotone* formulas in *disjunctive normal form* with at most *two literals* per clause. In contrast to the versions without the absolute value, however, we prove that these variants are *fixed-parameter tractable*. As technical contribution we present a kernelization for an auxiliary problem on hypergraphs in which we seek, given an edge-weighted hypergraph, an induced subgraph that maximizes the absolute value of the sum of edge-weights.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases parameterized complexity, kernelization, weighted maximum satisfiability, absolute value maximization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.12

Related Version *Technical Report*: <https://arxiv.org/abs/2204.12614>

1 Introduction

Maximum satisfiability is the natural generalization of the satisfiability problem of propositional logic to optimization problems. In its decision version MAX-SAT, we seek a truth assignment for a given propositional formula in conjunctive normal form that satisfies at least $\alpha \in \mathbb{N}$ clauses. This problem is known to be NP-complete even in the restricted case that every clause contains at most two literals [16]. On the positive side, it is known that the problem can be decided in time $f(\alpha) \cdot n^c$ for some function $f: \mathbb{N} \rightarrow \mathbb{N}$ and constant $c \in \mathbb{N}$, which means that the problem is *fixed-parameter tractable* parameterized by α [26]. This also holds for the weighted version in which clauses have positive integer weights. In this paper we extend the previous scenario by allowing also *negative weights*. We denote the corresponding problem with MAX-CNF: decide for a given propositional formula with *arbitrary integer* weights whether there is an assignment of weight at least α . In the presence of negative weights, many non-trivial variants of the problem arise, for instance MAX-DNF (the propositional formula is in disjunctive normal form), MAX-MONOTONE-CNF



© Max Bannach, Pamela Fleischmann, and Malte Skambath;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 12; pp. 12:1–12:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

12:2 MaxSAT with Absolute Value Functions

and MAX-MONOTONE-DNF (all literals in the formulas are positive), and ABS-CNF as well as ABS-DNF (maximize the *absolute value* of the sum of the satisfied clauses). Even an obscure combination such as ABS-MONOTONE-DNF becomes surprisingly expressive if negative weights are allowed (we show that this version is NP-hard restricted to formulas in which every clause contains at most two literals). This complexity jump (many of these versions are trivially in P if negative weights are *not* allowed) motivates a systematic study of the parameterized complexity of ABS-DNF and its variants. The main problem we investigate is defined as:

► **Problem 1** (p_α - d -ABS-DNF).

Instance: A propositional formula ϕ in disjunctive normal form with clauses of size at most d , a target value $\alpha \in \mathbb{N}$, and a weight function $w: \text{clauses}(\phi) \rightarrow \mathbb{Z}$.

Parameter: α

Question: Is there an assignment β such that $|\sum_{c \in \text{clauses}(\phi), \beta \models c} w(c)| \geq \alpha$ (the absolute value of the sum of the weights of satisfied clauses is at least α)?

We investigate the complexity of this problem by varying the status of α and d as parameter. In the notation above (α as index of p and d in the problem description) we consider α as a parameter and d as a fixed constant. But we also consider α and d to be parameters at the same time, or both to be constant, or to be neither a parameter nor a constant. An overview of the complexity of ABS-DNF can be found in Table 1.

■ **Table 1** Summary of our results for the maximum satisfiability problem over formulas in disjunctive normal form with negative weights and an absolute value target function (ABS-DNF). We distinguish how α (the weight of the sought solution) and d (maximum size of a clause) are interpreted (as *constant*, as *parameter*, or as *unbounded*). The columns correspond to d , while the rows represent α . Upper bounds are highlighted in green and lower bounds in red.

| $\alpha \setminus d$ | <i>constant</i> | <i>parameter</i> | <i>unbounded</i> |
|----------------------|--------------------------|--------------------------------|-----------------------------|
| <i>constant</i> | P (Corollary 25) | FPT (Corollary 24) | unknown |
| <i>parameter</i> | FPT (Theorem 3) | FPT (Corollary 24) | W[1]-hard (Corollary 12) |
| <i>unbounded</i> | NP-hard (Corollary 9) | para-NP-hard (Corollary 10) | NP-hard (Corollary 9) |

As mentioned before, we will also study other variations of the problem, which we obtain by replacing the DNF by a CNF, by forcing the formula to be monotone, or by replacing the absolute value function by a normal sum. It turns out that without the absolute value function, many natural versions of the problem become W[1]-hard in the presence of negative weights. However, the problems become tractable if the aim is to maximize the absolute value of the target sum as they than can all be fpt-reduced to p_α - d -ABS-MONOTONE-DNF, which in turn is equivalent to the following hypergraph problem:

► **Problem 2** (p_α - d -UNBALANCED-SUBGRAPH).

Instance: A d -hypergraph $H = (V, E)$, a weight function $w: E \rightarrow \mathbb{Z}$, and a number $\alpha \in \mathbb{N}$.

Parameter: α

Question: Is there a set $X \subseteq V$ with $|w[X]| \geq \alpha$.

In words, we seek a subset X of the vertices of an edge-weighted hypergraph such that the absolute value of $w[X] = \sum_{e \in E[X]} w(e)$ is maximized. Here $E[X]$ are the hyperedges induced by the set X . Intuitively, that means we have to find a subgraph that contains either more positive than negative edges or more negative edges than positive ones. Thus, we seek an *unbalanced* subgraph.

Our Contribution I: Hardness Results. We present hardness results for the parameterized and non-parameterized variants of MAX-DNF and ABS-DNF (both with negative weights). In particular, we prove that MAX-DNF and ABS-DNF are NP-hard when restricted to monotone formulas with at most two literals per clause. However, the problems differ from a parameterized perspective: p_α - d -MAX-MONOTONE-DNF is W[1]-hard for all $d \geq 2$, while p_α - d -ABS-DNF \in FPT. We also prove a similar result for the CNF-versions: p_α - d -MAX-MONOTONE-CNF is W[1]-hard for all $d \geq 2$, but p_α - d -ABS-CNF \in FPT.

Our Contribution II: Satisfiability Based Optimization with Absolute Value Function.

We derive a kernelization algorithm for p_α - d -UNBALANCED-SUBGRAPH, which implies that the problem is in FPT parameterized by α . Using an fpt-reduction we show:

► **Theorem 3.** p_α - d -ABS-DNF \in FPT

Our Contribution III: Absolute Integer Optimization. Theorem 3 can be generalized to an *absolute integer optimization problem*. Given a multivariate polynomial $p: \mathbb{Z}^n \rightarrow \mathbb{Z}$ with integer coefficients and intervals $[(b_{min})_i; (b_{max})_i]$ for every $i \in \{1, \dots, n\}$, the task is to find a solution $(x_1, \dots, x_n) \in \mathbb{Z}^n$ with $(b_{min})_i \leq x_i \leq (b_{max})_i$ such that the non-linear objective function $|p(x_1, \dots, x_n)|$ is maximized. With the value of the sought solution α and the maximal degree d of the polynomial, we prove:

► **Theorem 4.** $p_{\alpha,d}$ -ABS-IO \in FPT

We emphasize that there are no restrictions on the number of variables or on the size of an interval $[(b_{min})_i; (b_{max})_i]$. There may be even infinitely large intervals like $[-\infty; \infty]$.

Related Work. Optimization problems based on satisfiability or constraint-satisfaction problems are usually NP-complete [23, 31]. The parameterized complexity of these problems is, thus, an active field of research [17, 18, 21, 32]. An overview over the current trends can be found in [8, 30]. In a propositional formula in conjunctive normal form, we can always satisfy at least $(1 - 2^{-d})m$ clauses if m is the number of clauses and d is the size of these clauses [7, Chapter 9.2]. Therefore, the problem is fixed-parameter tractable when parameterized by the solution size α . It is known that the problem remains in FPT parameterized above this lower bound, i. e., the parameter is the distance between $(1 - 2^{-d})m$ and the number of clauses that can actually be satisfied simultaneously [1]. The currently fastest algorithm for parameter α runs in time $O^*(1.3248^\alpha)$ and improving the base is an active field of research [3].

Another parameterized approach to (non-optimizing) satisfiability problems are *backdoors*, which are small sets of variables such that assigning values to these variables yields an easy formula, e. g., a KROM- or HORN-formula [5, 28]. Unfortunately, MAX-CNF remains NP-hard on such formulas, making the approach less appealing for optimization problems.

A third line of research on parameterized algorithms for satisfiability problems focuses on *structural parameters*, i. e., graph theoretic properties of a graph representation of the formula. Depending on the details of that representation, various tractability and intractability results can be derived for parameters such as the *treewidth* of the corresponding graph [29, 30]. Such algorithms usually can be obtained from general frameworks such as Courcelle's Theorem and carry over to the optimization versions [4]. Generalizing this approach to a broader range of graphs is an active field of research, see for instance [15] or [2, Chapter 17].

A related problem is MAX-LIN2 [1, 6, 27]: given variables x_1, \dots, x_n and a set of equations $\prod_{i \in I} x_i = b$ for $b \in \{-1, 1\}$ and $I \subseteq \{1, \dots, n\}$, find an assignment to $\{-1, 1\}$ that satisfies as many equations as possible. This can be seen as maximizing *parity constraints*, in contrast

to *or* constraints (MAX-CNF) or *and* constraints (MAX-DNF). The parameterized complexity of *linear* integer programming is well understood, see for instance the famous result of Lenstra [25] or recent developments in algorithms based on structural parameters [9, 14, 22, 24]. There are also recent results on *non-linear* programs, see for instance [10, 19, 20]. For separable convex functions, fixed-parameter tractability results are known for structural parameters depending on the constraint matrix [11]. There are also results that directly cover absolute values of polynomials – for instance, for maximizing convex functions, which include absolute values of convex polynomials [20].

Structure of this Work. After some brief preliminaries in Section 2, we present lower bounds for various versions of ABS-DNF and related problems in Section 3. Afterwards, we formulate our main result – an FPT-algorithm for p_α - d -ABS-DNF – in Section 4. We also present the underlying machinery, i. e., a kernelization-algorithm for p_α - d -UNBALANCED-SUBGRAPH. Section 5 concludes with further applications and an outlook is presented in Section 6. Due to space constraints, some proofs are only available in the technical report.

2 Preliminaries

A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$, where an instance (w, k) consists of a word w and a *parameter* k . We denote parameterized problems with a preceding “p-” with an index that indicates what the parameter is – for instance, p_k -VERTEX-COVER denotes the well-known vertex cover problem parameterized by the solution size k . A parameterized problem is *fixed-parameter tractable* (*fpt* or it is in FPT) if there is a computable function $f: \mathbb{N} \rightarrow \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that we can decide $(w, k) \in Q$ in time $f(k) \cdot |w|^c$. A *size- g kernelization* for a parameterized problem Q and a computable function $g: \mathbb{N} \rightarrow \mathbb{N}$ is a polynomial-time algorithm that, on input of an instance (w, k) , outputs an instance (w', k') such that $(w, k) \in Q \Leftrightarrow (w', k') \in Q$ and $|w'| + k' \leq g(k)$. The output of the algorithm is called a *kernel*. If g is a polynomial then the kernel is called a *polynomial kernel*. It is well-known that a decidable problem is in FPT iff it admits a kernelization [13, Theorem 1.39].

A *d -hypergraph* $H = (V, E)$ consists of a set V of vertices and a set E of edges with $e \subseteq V$ and $|e| \leq d$ for all $e \in E$. A hypergraph is *d -uniform* if all edges have size *exactly* d . For $d = 2$, they are just *graphs*. The *neighborhood* of a vertex v is the set $N(v) = \{w \mid (\exists e \in E)(v, w \in e)\}$ and its degree $\deg(v)$ is the number of edges containing it. The *link* of a set $c \subseteq V$ is defined as $\text{link}(c) = \{e \in E \mid c \subseteq e\}$. A *subedge* of some hyperedge $e \in E$ is any subset $s \subseteq e$. We use $\Delta(H) = \max_{v \in V} \deg(v)$ (or just Δ , if H is clear from the context) to denote the *maximum degree* of H . For $X \subseteq V$ and $w: E \rightarrow \mathbb{Z}$, let $E[X] = \{e \in E \mid e \subseteq X\}$ denote the edges of the *induced hypergraph* $H[X] = (X, E[X])$ over X , and define its *weight* as $w[X] = \sum_{e \in E[X]} w(e)$.

We need only little terminology from propositional logic. A formula ϕ is in *disjunctive normal form* if it is a disjunction of conjunctions, e. g., $\psi \equiv (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$. It is in *conjunctive normal form* if it is a conjunction of disjunctions, e. g., $\chi \equiv (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$. We refer to the variables of ϕ with $\text{vars}(\phi)$, and we call the conjunctions in a formula in disjunctive normal form and the disjunctions in a formula in conjunctive normal form *clauses*. These objects are addressed with $\text{clauses}(\phi)$. For instance, $\text{vars}(\psi) = \text{vars}(\chi) = \{x_1, x_2, x_3\}$, $\text{clauses}(\psi) = \{(x_1, x_2), (x_2, x_3)\}$, and $\text{clauses}(\chi) = \{(x_1, x_2), (x_2, \neg x_3)\}$. A *literal* is a variable (then it is *positive*) or its negation (then it is *negative*). Formulas that contain only positive literals are *monotone*, e. g., ψ is monotone but χ is not.

3 Lower Bounds for MaxSAT and AbsSAT Based Optimization

The problems MAX-DNF and ABS-DNF may seem similar on first sight. After all, if there are no negative weights they are clearly equivalent. However, if negative weights are present, a solution of ABS-DNF may either construct a sum of at least α or of at most $-\alpha$; in contrast, a solution for MAX-DNF does not have such freedom and has to find a weight- α solution. This results in an interesting complexity gap: p_α - d -MAX-MONOTONE-DNF is W[1]-hard for $d \geq 2$ while p_α -ABS-DNF is only W[1]-hard if d is unbounded (recall that d is the size of the clauses and that in p_α - d -MAX-MONOTONE-DNF the size of every clause is bounded by a constant d). We prove the hardness results in this section and complement them with a proof of p_α - d -ABS-DNF \in FPT in the next section.

We first consider p_α - d -MAX-MONOTONE-DNF. This problem is W[1]-hard by a reduction from the independent set problem (given a graph and an integer k , decide whether there is a set of at least k pairwise non-adjacent vertices):

► **Lemma 5.** p_α - d -MAX-MONOTONE-DNF is W[1]-hard for $d \geq 2$.

Proof. Let $G = (V, E)$ and $k \in \mathbb{N}$ be an instance of p_k -INDEPENDENT-SET. We build a formula ϕ that contains a variable x_v for every $v \in V$:

$$\phi \equiv \bigvee_{v \in V} (x_v) \vee \bigvee_{\{v,w\} \in E} (x_v \wedge x_w).$$

\downarrow
1

\downarrow
-1

The red labeling shows the corresponding weight function w . Clearly, a size- k independent set translates into an assignment of weight k . For the other direction consider an assignment of weight k . We can assume that the assignment satisfies only positively weighted clauses. Otherwise we can flip one variable of a satisfied and negatively weighted clause. This does not decrease the weight because there is at most one positively weighted clause containing this variable. Since k variables are set to *true* and no negatively weighted clauses are satisfied, the assignment translates back to an independent set. ◀

► **Corollary 6.** p_α - d -MAX-DNF is W[1]-hard for $d \geq 2$.

On the other hand, the absolute value version of the problem remains intractable in the classical sense, which motivates our parameterized perspective. In detail, the (unparameterized) problem d -ABS-DNF is NP-hard by a reduction from INDEPENDENT-SET. Note that in contrast to the previous lemma, the reduction in the following is not parameter-preserving. Thus, the theorem does *not* imply that p_α - d -ABS-MONOTONE-DNF is W[1]-hard.

► **Theorem 7.** d -ABS-MONOTONE-DNF is NP-hard for $d \geq 2$.

Proof. We reduce from INDEPENDENT-SET and let $G = (V, E)$ with $k \in \mathbb{N}$ be a corresponding instance. Construct the following formula ϕ that contains four variables $v_1^+, v_2^+, v_1^-, v_2^-$ for every vertex $v \in V$ (the red arrows illustrate the weight function):

$$\phi \equiv \bigvee_{v \in V} (v_1^+ \wedge v_2^+) \vee (v_1^- \wedge v_2^-) \vee \bigvee_{\{v,w\} \in E} (v_1^+ \wedge w_1^+) \vee (v_1^- \wedge w_1^-).$$

\downarrow
-1

\downarrow
1

\downarrow
1

\downarrow
-1

The intuition (also illustrated in Example 8) behind the formula is as follows: Think of two weighted copies of G called G^+ and G^- such that all edges in G^+ have weight $+1$ and all edges in G^- have weight -1 . This is the second part of the formula. Observe that by taking either all edges of G^+ or all edges of G^- , a solution would always have absolute value $|E|$.

12:6 MaxSAT with Absolute Value Functions

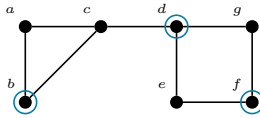
To encode the independent set problem, we introduce an extra gain for every vertex, these are the variables v_2^+ and v_2^- . Observe that in the first part of ϕ , we can add -1 or $+1$ to the solution by setting $v_1^{+/-}$ and $v_2^{+/-}$ to *true* at the same time. Finally, note that the signs in the first part of the formula are exactly the opposite of the signs in the second part. We claim that G has a size- k independent set iff ϕ has a solution of weight $k + |E|$. Let S be a size- k independent set in G , then the following assignment has an absolute value of $k + |E|$:

$$\beta(v_i^\sigma) = \begin{cases} \text{true} & \text{if } (v \in S \text{ and } \sigma = -) \text{ or } (\sigma = + \text{ and } i = 1); \\ \text{false} & \text{else.} \end{cases}$$

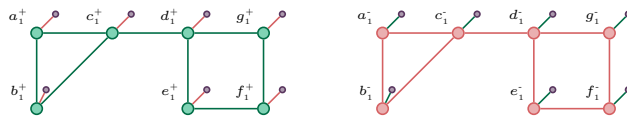
For the other direction let β be an assignment that achieves an absolute value $|\nu|$ of at least $k + |E|$. Assume for simplicity that $\nu > 0$ (the case $\nu < 0$ is symmetric). We may assume $\beta(v_1^+) = \text{true}$ and $\beta(v_2^+) = \text{false}$ – otherwise we can increase the solution by modifying the assignment locally. Observe that by setting only variables v_i^+ to *true*, we have $\nu \leq |E|$ and, since we assumed $|\nu| \geq k + |E|$, β sets further variables v_j^- to *true*. We may assume $\beta(v_1^-) = \beta(v_2^-)$ for all $v \in V$, as setting both variables to *true* is the only way to increase ν (the clauses of the form $(v_1^- \wedge w_1^-)$ all have a negative weight and we assumed $\nu > 0$).

Let $\{v, w\} \in E$ be an edge such that $\beta(v_1^-) = \beta(w_1^-) = \text{true}$. Observe that changing $\beta(w_1^-)$ to $\beta(w_1^-) = \text{false}$ does *not* decrease the value of the solution (it decreases the number of satisfied clauses with negative weight by exactly one, as the positively weighted clause $(w_1^- \wedge w_2^-)$ is not satisfied anymore; but it increases the number of satisfied clauses with positive weight by at least one as the negatively weighted clause $(v_1^- \wedge w_1^-)$ is also no longer satisfied). Hence, we may adapt β such that $S = \{v \mid \beta(v_1^-) = \text{true}\}$ is an independent set. Since the absolute value of β is at least $|E| + k$ and since all clauses containing variables of the form v_i^+ can contribute at most $|E|$ to this sum, we have $|S| \geq k$. ◀

► **Example 8.** Let us illustrate the reduction used within the theorem with an example. Consider the following graph G with an independent set of size three:



Within the reduction, we virtually generate the following edge-weighted graph that contains two copies of G , namely G^+ and G^- . In both copies, every vertex has a mirror vertex attached to it. Edges have either a positive weight of $+1$ or a negative weight of -1 :



The reduction continues by defining a weighted formula that contains a clause for every edge with the same weight as that edge, e. g., we would have $(a_1^+ \wedge b_1^+)$ of weight $+1$ and $(a_1^- \wedge b_1^-)$ of weight -1 . By selecting from one copy (i. e., setting the corresponding variables to *true*) all vertices except the mirrors, we obtain an assignment of weight $\pm|E|$; selecting the independent set together with its mirrors in the other copy provides another $\pm k$, yielding a solution of absolute value $|E| + k$. In this example we could take all green vertices (providing a weight of $+8$) and $\{b_1^-, d_1^-, f_1^-\}$ together with their mirrors (for another $+3$); yielding a solution with absolute value 11. ▽

► **Corollary 9.** d -ABS-DNF is NP-hard for $d \geq 2$.

► **Corollary 10.** p_d -ABS-DNF is para-NP-hard.

Due to the previous theorem, we cannot hope to achieve parameterized tractability with respect to the sole parameter d . The following lemma shows that this is also not possible for the parameter α . In the upcoming sections, we thus rely on the combined parameter $\alpha + d$.

► **Lemma 11.** p_α -ABS-MONOTONE-DNF is W[1]-hard.

Proof. Let $G = (V, E)$ and $k \in \mathbb{N}$ be an instance of p_k -INDEPENDENT-SET. Consider the following formula ϕ that contains a variable x_v for every vertex $v \in V$:

$$\phi \equiv \bigvee_{v \in V} \left(\neg x_v \wedge \bigwedge_{\{v,w\} \in E} x_w \right).$$

Let $\alpha = k$ and w be a weight function that maps all clauses of ϕ to 1. We show that (G, k) is a yes-instance of INDEPENDENT-SET iff (ϕ, w, α) is a yes-instance of ABS-DNF. For the first direction let $I \subseteq V$ be a size- k independent set and consider the assignment:

$$\beta(x_v) = \begin{cases} \text{false} & \text{if } v \in I; \\ \text{true} & \text{otherwise.} \end{cases}$$

Since I is an independent set we have $\beta(x_w) = \text{true}$ for all vertices w that have a neighbor $v \in I$. Therefore, β satisfies every clause that corresponds to a vertex of I and, hence, β is an assignment of weight at least k .

For the other direction let β be an assignment of weight at least k , i. e., one that satisfies at least k clauses. For each of these clauses there is a unique vertex v such that x_v appears negatively only in that clause and, thus, at least k variables must be assigned to *false* by β . Furthermore, for all neighbors w of v we have $\beta(x_w) = \text{true}$ by the second part of the clause. We conclude that $\{v \mid \beta(x_v) = \text{false}\}$ is an independent set of size at least k .

The formula ϕ is not monotone, however, in the proof of Lemma 17 we describe how an arbitrary instance can be turned into an equivalent one with a monotone formula. In this case we obtain the following monotone formula ϕ' , which (instead of ϕ) can be constructed directly for this reduction.

$$\phi' \equiv \bigvee_{v \in V} \left(\bigwedge_{\{v,w\} \in E} x_w \right) \vee \bigvee_{v \in V} \left(x_v \wedge \bigwedge_{\{v,w\} \in E} x_w \right).$$

\downarrow
1

\downarrow
-1

The red labeling shows the corresponding weight function w' . As we describe in the proof of Lemma 17 the instances (ϕ, w, k) and (ϕ', w', k) are equivalent. This completes the proof. ◀

► **Corollary 12.** p_α -ABS-DNF is W[1]-hard.

3.1 From Disjunctive Normal Form to Conjunctive Normal Form

So far, we have seen that the parameterized DNF maximization variants are W[1]-hard. We can use this to show that the same holds also for the conjunctive normal forms:

► **Lemma 13.** p_α - d -MAX-MONOTONE-CNF is W[1]-hard for $d \geq 2$.

► **Corollary 14.** p_α - d -MAX-CNF is W[1]-hard for $d \geq 2$.

3.2 Finding an Assignment with a Certain Weight

We finish this section with a variant of the problem in which we seek an assignment of a certain absolute value. In detail, $p_{\alpha,d}$ -EXACT-ABS-MONOTONE-DNF asks, given a weighted propositional formula in disjunctive normal form, whether there is an assignment such that the absolute value of the weighted sum is *exactly* α (and not *at least* α).

► **Lemma 15.** d -EXACT-ABS-MONOTONE-DNF is NP-hard for any constant $d \geq 2$ and $\alpha = 0$.

Proof. Since $p_{\alpha,d}$ -MAX-MONOTONE-DNF is W[1]-hard by Lemma 5, it is not surprising that d -EXACT-MAX-MONOTONE-DNF is NP-hard. We just use the same polynomial-time-computable reduction, which is possible since the independent set problem is monotone in the sense that the existence of a solution of size $\geq k$ implies the existence of an independent set of size $= k$.

We can turn a d -EXACT-MAX-MONOTONE-DNF instance into a d -EXACT-ABS-MONOTONE-DNF instance: Just add the empty clause with weight $-\alpha$ and set the target value to 0. Every assignment of weight α in the original instance will have weight 0, as the empty clause is always satisfied. On the other hand, any weight-0 assignment in the new formula has to satisfy clauses of weight exactly α to compensate for the empty clause. ◀

► **Corollary 16.** $p_{\alpha,d}$ -EXACT-ABS-MONOTONE-DNF is para-NP-hard.

Proof. A parameterized problem is para-NP-hard if there are finitely many slices whose union is NP-hard [13, Theorem 2.14]. The claim follows as already the slice $\alpha = 0$ alone is NP-hard by Lemma 15. ◀

4 Satisfiability Based Optimization with Absolute Value Function

In the previous section we showed that various SAT-based optimization problems are presumably not in FPT with respect to the natural parameter α . There are various ways to deal with these negative results. For instance, we could restrict ourselves to structured instances – such as formulas of bounded incidence-treewidth [8]. For the absolute version of the problem, we fortunately do not need such a strong structural parameter. Instead, we provide a reduction to an auxiliary hypergraph problem for which we present a polynomial kernel. This results in the main theorem of this section:

► **Theorem 3.** $p_{\alpha,d}$ -ABS-DNF \in FPT

It is well-known that various optimization problems over CNFs and DNFs are equal to problems on hypergraphs. For instance, it is easy to see that the question of whether a given *monotone* d -CNF can be satisfied by setting at most k variables to *true* equals the hitting set problem on d -hypergraphs. Notice that “monotone” here is a crucial key word, as this property allows us to encode clauses as edges. Such an approach does *not* work directly for $p_{\alpha,d}$ -MAX-DNF. Fortunately, $p_{\alpha,d}$ -ABS-DNF reduces to its monotone version:

► **Lemma 17.** $p_{\alpha,d}$ -ABS-DNF \leq $p_{\alpha,d}$ -ABS-MONOTONE-DNF

Proof. Let ϕ be a d -DNF and $w: \text{clauses}(\phi) \rightarrow \mathbb{N}$ be a weight function on its clauses. We argue how we can remove one occurrence of a negative literal – an iterated application of this idea then leads to the claim.

Let $c \in \text{clauses}(\phi)$ be a clause containing a negative literal \bar{x} . We remove c from ϕ and add two new clauses: c^+ and c^- . The clause c^+ equals c without \bar{x} , and we set $c^- = c^+ \wedge x$. Define $w(c^+) = w(c)$ and $w(c^-) = -w(c)$, and observe that an assignment that satisfies c does satisfy c^+ but not c^- and, thus, yields the same value. On the other hand, every assignment that satisfies c^- does also satisfy c^+ and, hence, the values cancel each other. ◀

Given a monotone DNF and a weight function on its clauses, we can encode the problem p_α - d -ABS-MONOTONE-DNF as a hypergraph by introducing one vertex per variable and by representing clauses as edges. The task then is to identify a set of vertices such that the absolute value of the total weight of the edges appearing in the subgraph induced by the vertices is as large as possible. Formally this is Problem 2 from the introduction.

► **Corollary 18.** p_α - d -ABS-DNF \leq p_α - d -UNBALANCED-SUBGRAPH

4.1 A Polynomial Kernel for Unbalanced Subgraph

In this section we develop a polynomial kernel for p_α - d -UNBALANCED-SUBGRAPH. The kernelization consists of a collection of *reduction rules* which we just call *rules*. Every rule obtains as input an instance of p_α - d -UNBALANCED-SUBGRAPH and either (i) is not applicable, or (ii) outputs an equivalent smaller instance. The rules are numbered and we assume that whenever we invoke a rule, all rules with smaller numbers have already been applied exhaustively. It will always be rather easy to show that the rules can be implemented in polynomial time, so we only prove that they are safe. To get started, we consider the following simple rules, where Δ is the maximum vertex degree in the hypergraph:

- **Rule 1.** *If H contains an isolated vertex v , delete v .*
- **Rule 2.** *If H contains an edge e with $w(e) = 0$, delete e .*
- **Rule 3.** *If H has at least $2\alpha \cdot d^3 \Delta^2$ vertices, reduce to a trivial yes-instance.*
- **Lemma 19.** *Rule 1–3 are safe.*

Proof. For the first two rules observe that neither isolated vertices nor weight-0 edges can contribute to any solution and, hence, may be discarded.

For Rule 3 we argue that H contains a set $X \subseteq V$ with $|w[X]| \geq \alpha$. Consider a maximal set $M \subseteq E$ such that (i) all edges in M are pairwise disjoint and (ii) $H[\cup_{e \in M} e] = (\cup_{e \in M} e, M)$, that is, M is a set packing that induces itself. Such a set always exists and can also be computed as the following greedy algorithm shows: Start with $M = \emptyset$ and repeatedly pick a remaining nonempty edge e from H that is no superset of any edge in H (except \emptyset) and add it to M . Since Rule 1 has already removed isolated vertices, there must be such an edge as long as V is not empty. After picking an edge e , remove the set E_e of all edges that intersect e . Observe that $|E_e| \leq d\Delta$. Clearly, if we just remove E_e , the algorithm would already generate a maximal set packing – however, it would not necessarily be self-induced. Let therefore $V_e = \{v \in V \mid (\exists e' \in E_e)(v \in e')\}$ be the set of vertices covered by the to-be-removed edges. Since every edge has size at most d , we have $|V_e| \leq d^2 \Delta$. To ensure that M will be self-induced, we remove all edges that contain a vertex of V_e – which are at most $d^2 \Delta^2$ edges as H has maximum degree Δ .

For every edge that is picked to be in M , at most $d^2 \Delta^2$ edges are removed from $|E|$. Therefore we have $|M| \geq |E| / (d^2 \Delta^2)$. Furthermore, we have $|E| \geq |V| / d$ because Rule 1 has already removed isolated vertices. Given that $|V| \geq 2\alpha \cdot d^3 \Delta^2$, we deduce $|M| \geq 2\alpha$. There are at least α edges in M that have the same sign (no edge has a zero-weight by Rule 2). Therefore, the set X of vertices induced by the larger set satisfies $|w[X]| \geq \alpha$. ◀

12:10 MaxSAT with Absolute Value Functions

The rules yield a kernel for the combined parameter $\alpha + \Delta$, which means that an exhaustive application of these rules leaves an instance of size bounded in α and Δ .

► **Corollary 20.** $\text{P}_{\alpha, \Delta}$ - d -UNBALANCED-SUBGRAPH has a kernel with $O(\alpha \cdot \Delta^2)$ vertices.

If we only consider $\alpha + d$ as parameter, rules 1–3 are not applicable, and the graph is still large, we know that the input graph has no isolated vertices, all edges have a non-zero weight, and we know that there is at least one vertex of high degree. Our final reduction rule states that in this case we can as well reduce to a trivial yes-instance, since the high degree vertex induces a solution. Intuitively, this is similar to an application of the well-known Sunflower Lemma [12], which states that large enough hypergraphs contain a set of hyperedges that pairwise all intersect in the same set of vertices (i. e., the “hypergraph version” of a high-degree vertex). Unfortunately, a direct application of the lemma is tricky as other hyperedges may intersect arbitrarily with such sunflowers. To circumnavigate this difficulties, we need objects that are more like a *self-induced sunflower*. This leads to the following reduction rule; details of these concepts are postponed to the proof of Lemma 21.

► **Rule 4.** Let $g(i) = (i^i 2\alpha \cdot 2^{2^d})^{2^i - 1}$ for $i > 0$ and $g(0) = 1$. If H contains a subedge $c \subseteq e \in E$ such that (i) $|\text{link}(c)| \geq g(i)$ where $i = d - |c|$ and (ii) for every superset $f \supseteq c$ it holds that $|\text{link}(f)| < g(j)$ where $j = d - |f|$, then reduce to a trivial yes-instance.

Before we prove the safeness of the rule in the general case, let us quickly sketch why the rule is correct for 2-uniform hypergraphs, i. e., normal graphs. A vertex v with $\deg(v) \geq 4\alpha$ is incident to at least 2α edges of the same sign. For the sake of an example let there be a set $P(v) \subseteq N(v)$ with $|P(v)| \geq 2\alpha$ and $w(\{v, u\}) > 0$ for all $u \in P(v)$. Either $P(v)$ is a solution (i. e., $|w[P(v)]| > \alpha$) or $P(v) \cup \{v\}$ is a solution (since $w[P(v) \cup \{v\}] \geq 2\alpha - |w[P(v)]| \geq \alpha$).

► **Lemma 21.** Rule 4 is safe.

Proof. Let $c \subseteq V$ be a set for which the rule is applicable. We argue that H contains a set $X \subseteq V$ with $|w[X]| \geq \alpha$. Let $i = d - |c|$, $E(c) = \text{link}(c) = \{e \in E \mid c \subsetneq e\}$, $N(c) = \bigcup E(c) \setminus c$, and $\deg_{E(c)}(v) = |\{e \in E(c) \mid v \in e\}|$. We can assume that $|c| < d$ and $i > 0$ because otherwise $|\text{link}(c)| = 0 < g(i)$. Since $|\text{link}(f)| < g(j)$ holds for any superset $f \supseteq c$ with $j = d - |f|$, it also holds that $\deg_{E(c)}(v) \leq g(i - 1)$ for every vertex $v \in N(c)$. Otherwise we have a contradiction with $f = c \cup \{v\}$.

Consider a maximal set $M \subseteq E(c)$ such that (i) all edges in M contain c as a subedge, (ii) c is the intersection of any pair of different edges in M , and (iii) for every edge $e \in E(c)$ it holds that either $e \in M$, or $e \not\subseteq (\bigcup_{e' \in M} e')$, that is, M is a set of edges that induces only itself in $H_c = (V, E(c))$. By (i) and (ii) M is also known as a *sunflower* with *core* c . Such a set always exists as the following greedy algorithm shows: Start with $M = \emptyset$ and observe that all three properties hold. Repeatedly pick an edge $e \in E(c)$ that has no smaller subedge in $E(c)$ and add it to M as long as the properties are preserved.

We will use M to identify the sought set X . For this, we require a lower bound for the size of M . Therefore, let us carefully analyze which edges cannot be chosen after an edge has been added to M . After picking an edge $e \in E(c)$, we can remove the whole set $E_e \subseteq E(c)$ of all edges that intersect e not only in c (a proper superset of c) because picking one of those edges contradicts property (ii). There may be also an edge $e' \in E(c) \setminus E_e$ that has not been added yet, but for which there is another edge $e'' \in E(c) \setminus M$ with $e'' \in \bigcup M \cup \{e'\}$. This implies that e' may not be added to M to preserve property (iii). Furthermore, there must be an edge $e \in M$ such that e' intersects either e or some edge in E_e outside in the core. Let $E'_e \subseteq E(c)$ be the set of all edges that intersect E_e not only in c . It follows

that any edge in $E(c)$ will either be inserted to M , or is in E_e or E'_e for an edge $e \in M$. Observe that $|E_e| \leq ig(i-1)$ because $e \setminus c$ contains at most i vertices that are in at most $g(i-1) - 1$ edges from $E(c)$. For the same reason, we have $|E'_e| \leq |E_e| \cdot (i-1) \cdot g(i-1)$. Therefore, $|\{e\} \cup E_e \cup E'_e| \leq i^2(g(i-1))^2$ edges are removed for every edge in M . This implies $|M| \geq |E(c)| / (i^2 g(i-1)^2)$. Given that $E(c) \geq g(i)$, we deduce $|M| \geq 2\alpha \cdot 2^{2^d}$: For $i = 1$, we have $|M| \geq |E(c)|$ because of $g(0) = 1$. For $i > 1$ we use that $x^{2^i-1} / x^{2 \cdot 2^{i-1}-2} = x$ and $g(i) = x^{2^i-1}$ for $x = 2\alpha \cdot 2^{2^d}$.

We show that we can find a set $X \subseteq \bigcup_{e \in M} e$ with $|w[X]| \geq \alpha$. Let M^+ denote all positively weighted edges in M and let M^- denote the other ones. Let us consider three cases based on the size of the core.

First Case ($|c| = 0$). We have $c = \emptyset$ and, without loss of generality, let $w[\emptyset] = |w[\emptyset]| \geq 0$. Otherwise we could flip the sign of all weights. Note that $E(\emptyset) = E \setminus \{\emptyset\}$. By Property (iii), any subset of $\bigcup M$ induces only edges in $M \cup \{\emptyset\}$. It follows that $w[\bigcup M^+] = |M^+| + w[\emptyset]$. We are done for $w[\bigcup M^+] \geq \alpha$, otherwise we obtain

$$|w[\bigcup M^-]| = |M^-| - w[\emptyset] = |M| - (|M^+| + w[\emptyset]) > |M| - \alpha.$$

The claim follows with $|M| \geq 2\alpha$.

Second Case ($|c| = 1$). In this case we have $c = \{v\}$ for some vertex $v \in V$. Without loss of generality, let $w[c] = |w[c]| \geq 0$ – otherwise we can flip the sign of all weights again. In contrast to the previous case, a subset of $\bigcup M \cup c$ might induce edges in M and subedges of c . Those edges are induced by $V' = \bigcup M \setminus c$. For any $X' \subseteq V'$ we have $|w[X']| < \alpha$, otherwise we are done. Observe that $w[\bigcup M^+] = |M^+| + w[c] + w[\bigcup M^+]$. The statement follows immediately for $w[\bigcup M^+] \geq \alpha$. Otherwise we obtain

$$\begin{aligned} |M^+| + w[c] &< \alpha + w[\bigcup M^+] < 2\alpha \quad \text{and} \\ |w[\bigcup M^-]| &= |M^-| - w[c] + w[\bigcup M^-] \\ &= |M| - |M^+| - w[c] - w[\bigcup M^-] > |M| - 3\alpha. \end{aligned}$$

The claim follows with $|M| \geq 4\alpha$.

Third Case ($|c| \geq 2$). Observe that in this case $\bigcup M$ may induce edges that are neither in $E(c)$ nor subedge of c , but that intersect c (and other vertices covered by M). This is because M contains only edges of $E(c)$. Let us recursively define the following subsets of $E[\bigcup M^\sigma]$ for $\sigma \in \{-, +\}$:

$$\begin{aligned} E_\emptyset^\sigma &= E \left[\left(\bigcup M^\sigma \setminus c \right) \setminus E[c]; \right. \\ E_{c'}^\sigma &= E \left[c' \cup \left(\bigcup M^\sigma \setminus c \right) \setminus \bigcup_{f \subsetneq c'} E_f^\sigma \setminus E[c] \quad \text{for every } c' \subseteq c. \right. \end{aligned}$$

For all $c' \subseteq c$ these sets are disjoint and every edge $e \in E[\bigcup M^\sigma]$ is either in one of these sets or in $E[c]$. To be precisely, an edge $e \in E[\bigcup M^\sigma]$ is in $E_{e \cap c}^\sigma$ if it is a proper superset of c . Otherwise $e \subseteq c$ (if $e \subseteq c$). Note that $M^\sigma = E_c^\sigma$. We will now show that there is a set $c' \subseteq c$ and such that the absolute value of the induced weight of $c' \cup (\bigcup M^- \setminus c)$ or $c' \cup (\bigcup M^+ \setminus c)$ is at least α . Assume this is not the case and that for all subsets $X \subseteq \bigcup M$ we have

$$|w[X]| < \alpha \tag{1}$$

12:12 MaxSAT with Absolute Value Functions

for any $c' \subseteq c$ and $\sigma \in \{-, +\}$. Observe that

$$E \left[c' \cup \left(\bigcup M^\sigma \setminus c \right) \right] = E_{c'}^\sigma \dot{\cup} E[c'] \dot{\cup} \bigcup_{f \subsetneq c'} E_f^\sigma. \quad (2)$$

Let $w_{c'}^\sigma = \sum_{e \in E_{c'}^\sigma} w(e)$ and note that this is the total weight of the edges in $E_{c'}^\sigma$. Hence:

$$w \left[c' \cup \left(\bigcup M^\sigma \setminus c \right) \right] = w_{c'}^\sigma + w[c'] + \sum_{f \subsetneq c'} w_f^\sigma. \quad (3)$$

For any $k \in \{0, \dots, |c|\}$ let $w_k = 2^{2^k-1} \cdot 2\alpha$. We show that w_k is an upper bound in the sense that $|w_{c'}^\sigma| < w_k$ for $c' \subseteq c$ and $k = |c'|$. By (1) and (3) we deduce that

$$|w_\emptyset^\sigma| = \left| w \left[\left(\bigcup M^\sigma \setminus c \right) \right] - w[\emptyset] \right| < 2\alpha = w_0.$$

Otherwise either $|w[\emptyset]| \geq \alpha$ or $|w[(\bigcup M^\sigma \setminus c)]| \geq \alpha$. Observe that $2\alpha + (2^{k-1} - 1)w_{k-1} \leq w_k$ for $k > 0$. Then for $\emptyset \neq c' \subseteq c$ it follows by induction that

$$\begin{aligned} |w_{c'}^\sigma| &= \left| w \left[c' \cup \left(\bigcup M^\sigma \setminus c \right) \right] - w[c'] - \sum_{f \subsetneq c'} w_f^\sigma \right| \\ &< \alpha + \alpha + \sum_{k=0}^{|c'|-1} \binom{|c'|}{k} w_k \\ &\leq 2\alpha + (2^{|c'} - 1)w_{|c'|-1} \\ &\leq w_{|c'|}. \end{aligned}$$

Since $|M| \geq 2\alpha \cdot 2^{2^d} \geq 4\alpha \cdot 2^{2^{d-1}-1} = 2 \cdot w_{|c|}$ either $|M^+| \geq w_{|c|}$ or $|M^-| \geq w_{|c|}$. Recall that $M^\sigma = E_c^\sigma$ and, thus, with $|w_c^\sigma| \geq |M^\sigma|$ we get that our assumption must be wrong. ◀

► **Corollary 22.** p_α - d -UNBALANCED-SUBGRAPH has a kernel with $O(\alpha^{2^d})$ vertices.

Proof. If Rule 4 cannot be applied, there is no such subedge $c \subseteq e \in E$ that meets the two properties of the rule. However, this also implies that there is no subedge c that only meets the first property because otherwise there would be an inclusionwise largest superset (at last a size- $(d-1)$ edge), which does. Therefore, for any subedge $c \subseteq e \in E$ it holds that $|\text{link}(c)| < (i^i 2\alpha \cdot 2^{2^d})^{2^i-1}$ where $i = d - |c|$. Especially for $c = \emptyset$ we have $|\text{link}(c)| < (2d^d 2^{2^d} \alpha)^{2^d-1}$. By the definition of the link we have $|E| \leq |\text{link}(\emptyset)| + 1$. Thus, the total number of edges is at most $(2d^d 2^{2^d} \alpha)^{2^d-1}$. Hence, we get a kernel of size $O(\alpha^{2^d})$. ◀

Proof of Theorem 3. Combine Corollary 18 with Corollary 22. ◀

If d is not a constant but a parameter, then, by an exhaustive application of the rules above, we obtain reduced instances of size $O(d(2d^d 2^{2^d} \alpha)^{2^d-1})$. Note that there could be up to n^d possible subedges, therefore it is not clear that we obtain a kernel. However, a closer look at Rule 4 reveals that we are not forced to consider these subedges – we can directly return a trivial yes-instance if $|E| \geq (2d^d 2^{2^d} \alpha)^{2^d-1}$.

► **Corollary 23.** $p_{\alpha,d}$ -UNBALANCED-SUBGRAPH \in FPT

Proof. A simple induction shows that when Rule 1 and Rule 2 were applied exhaustively and $|E| \geq (2d^d 2^{2^d} \alpha)^{2^d-1}$, Rule 4 can be applied on $c = \emptyset$ because $E \setminus \{\emptyset\} = \text{link}(\emptyset)$ or on some larger subedge. Hence we can return a trivial yes-instance whenever $|E| \geq (2d^d 2^{2^d} \alpha)^{2^d-1}$. ◀

► **Corollary 24.** $p_{\alpha,d}$ -ABS-DNF \in FPT

► **Corollary 25.** α - d -ABS-DNF \in P

4.2 From Disjunctive Normal Form to Conjunctive Normal Form

Similar to Lemma 13 in Section 3, the goal of this section is to companion Corollary 24 with a CNF-version of the claim.

► **Lemma 26.** $p_{\alpha,d}\text{-ABS-CNF} \leq_{\text{fpt}} p_{\alpha,d}\text{-ABS-DNF}$

Proof. Let (ϕ, w, α) be an instance of $p_{\alpha,d}\text{-ABS-CNF}$ and let $\Phi = \text{clauses}(\phi)$. The idea of the reduction is to successively replace every disjunction in Φ by a set of conjunctions with new weights such that we have for the resulting pair (Φ', w') :

$$\sum_{\sigma \in \Phi, \beta \models \sigma} w(\sigma) = \sum_{\sigma \in \Phi', \beta \models \sigma} w'(\sigma).$$

Let c be some disjunction in Φ . Recall that c contains at most d variables. There is a set X of at most $2^{|\text{vars}(c)|}$ conjunctions such that we have for every truth assignment $\beta: \text{vars}(\phi) \rightarrow \{\text{true}, \text{false}\}$ (i) $\beta \models c$ iff β satisfies exactly one formula in X , and (ii) $\beta \not\models c$ iff β satisfies no formula in X . Such a set X can easily be computed using the truth table of c , which is possible in the reduction as d is a parameter. Let $w'(c') = w(c)$ for any $c' \in X$, and $w'(\tilde{c}) = w(\tilde{c})$ for any $\tilde{c} \in \Phi \setminus \{c\}$.

For the reduction, we replace Φ by $\Phi' = \Phi \setminus \{c\} \cup X$ and update w to w' accordingly as long as there is a disjunction $c \in \Phi$. Note that this will replace exactly $|\text{clauses}(\phi)|$ disjunctions with at most 2^d conjunctions each, which implies that the reduction can be carried out by an fpt-algorithm. The resulting pair (Φ', w') satisfies for every truth assignment β :

$$\begin{aligned} & \sum_{\sigma \in \Phi, \beta \models \sigma} w(\sigma) \\ &= \sum_{\sigma \in \Phi \setminus \{c\}, \beta \models \sigma} w(\sigma) + \begin{cases} w(c) & \text{if } \beta \models c; \\ 0 & \text{otherwise;} \end{cases} \\ &= \sum_{\sigma \in \Phi \setminus \{c\}, \beta \models \sigma} w(\sigma) + \sum_{\sigma \in X, \beta \models c} w(c) \\ &= \sum_{\sigma \in \Phi \setminus \{c\}, \beta \models \sigma} w'(\sigma) + \sum_{\sigma \in X, \beta \models c} w'(\sigma) \\ &= \sum_{\sigma \in \Phi \setminus \{c\} \cup X, \beta \models \sigma} w'(\sigma). \end{aligned}$$

► **Corollary 27.** $p_{\alpha,d}\text{-ABS-CNF}, p_{\alpha,d}\text{-ABS-MONOTONE-CNF} \in \text{FPT}$

5 Application: Absolute Integer Optimization

We have seen that $p_{\alpha,d}\text{-ABS-MONOTONE-DNF} \in \text{FPT}$. We can generalize this problem to an algebraic optimization problem: Given a sum of products (a *multivariate polynomial* over binary variables), find an assignment such that the absolute value of the sum is at least α . Using binary variables, this problem is essentially the same problem as ABS-MONOTONE-DNF (in fact, optimization over DNFs is sometimes called sum of products). However, what happens if we do not have binary variables, but arbitrary integers from some given domain? We prove in this section that the problem remains tractable for similar parameters.

Let us represent a multivariate polynomial $p_{A,w}(z_1, \dots, z_n)$ over n variables z_1, \dots, z_n by an $n \times m$ matrix A and a vector w of length m as follows: Define $A\langle z, i, j \rangle$ to be $z^{A_{i,j}}$ if $z \neq 0$ or $A_{i,j} \neq 0$ and 1 otherwise; then write $p_{A,w}(z_1, \dots, z_n) = \sum_{j=1}^m w_j \cdot \prod_{i=1}^n A\langle z_i, i, j \rangle$.

► **Problem 28** ($p_{\alpha,d}$ -ABS-IO).

Instance: An $n \times m$ matrix $A \in \mathbb{N}_{\geq 0}^{n \times m}$, a weight vector $w \in \mathbb{Z}^m$, two bounding vectors $b_{min} \in (\mathbb{Z} \cup \{-\infty\})^n$ and $b_{max} \in (\mathbb{Z} \cup \{\infty\})^n$, and a target value $\alpha \in \mathbb{N}$.

Parameter: $\alpha, d = \max_{j \in \{1, \dots, m\}} \sum_{i=1}^n A_{i,j}$

Question: Is there a solution $x = (x_1, x_2, \dots, x_n) \in \mathbb{Z}^n$ such that $|p_{A,w}(x)| \geq \alpha$ and for all $i \in \{1, \dots, n\}$ we have $(b_{min})_i \leq x_i \leq (b_{max})_i$?

Here the parameter d is the maximal number of variables that do occur in any product of the polynomial. If there is no exponent larger than 1, then d is the number of variables in every product or *monomial*.

In this problem we restrict the domain of each variable to an interval. One might ask whether we could allow arbitrary linear inequations instead of upper and lower bounds, which is common in linear programming. However, in this case the problem becomes W[1]-hard:

► **Lemma 29.** $p_{\alpha,d}$ -ABS-IO with linear inequation constraints is W[1]-hard.

Proof. To prove this we turn an instance of p_k -INDEPENDENT-SET into an instance of $p_{\alpha,d}$ -ABS-IO with additional inequations. For a graph $G = (\{1, \dots, n\}, E)$ and a number $k \in \mathbb{N}$, let $A \in \mathbb{Z}^{n \times n}$ be the identity matrix, $w = (1, \dots, 1) \in \mathbb{Z}^n$, and $\alpha = k$. Further set $b_{min} = (0)^n$, $b_{max} = (1)^n$, and let $x_u + x_v \leq 1$ for every edge $\{u, v\} \in E$ be additional constraints.

Since all weights in w are positive, the absolute value is the exact value of the sum. Every size- k independent has now a corresponding variable assignment and vice versa. ◀

Observe that variables may have a huge or possibly infinite domain of possible values and, hence, it is not obvious that $p_{\alpha,d}$ -ABS-IO is in FPT. However, we already mentioned that $p_{\alpha,d}$ -ABS-MONOTONE-DNF is equivalent to $p_{\alpha,d}$ -ABS-IO if (i) the domain of all variables is $\{0, 1\}$ (i. e., $b_{min} = (0)^n$ and $b_{max} = (1)^n$), and if (ii) we have no exponents (i. e., $A \in \{0, 1\}^{m \times n}$).

Therefore, we only have to consider the remaining cases. If (i) holds but (ii) does not, we can simply replace all nonzero-values in A by 1. This does not change the value of $p_{A,w}(x_1, \dots, x_n)$ as by (i) we have only binary variables and w remains the same.

If (i) does not hold, we use a set of reduction rules to transform an instance over an arbitrary domain into one in which the domain is a superset of $\{0, 1\}$. This is, of course, not the same as property (i). However, while adapting the domain, we can transform the instance such that it is reducible to an instance of $p_{\alpha,d}$ -UNBALANCED-SUBGRAPH with the following property: Rule 4 either identifies it as a yes-instance or does nothing. This property can be used by an algorithm for $p_{\alpha,d}$ -ABS-IO with the following trick: First modify the domain such that it contains $\{0, 1\}$; then virtually shrink the domain to $\{0, 1\}$ (this restricts the solution space and, thus, may turn a yes-instance to a no-instance, but may not turn a no-instance to a yes-instance); perform the reduction and apply Rule 4; either deduce that we are dealing with a yes-instance (trivial decision of Rule 4) or that the instance is small (as the rule did nothing). In the latter case restore the larger domain and explore a search tree to solve the problem (the size of the search tree is bounded, as Rule 4 did not trigger).

Let us first describe the reduction rules. As before, we assume that a rule may only be applied if rules with smaller numbers were applied exhaustively. The first rule removes unnecessary variables, products, and constraints; or detects that the instance has no solution.

► **Rule 5.** Apply the following modifications if possible:

1. If $w_j = 0$ for some j , then remove the j -th column from A and w .
2. If there is some i such that $A_{i,j} = 0$ for every j , then remove the i -th row from A , $(b_{min})_i$, and $(b_{max})_i$.

3. If there is some i with $(b_{max})_i < (b_{min})_i$, then return a trivial no-instance.
4. If there is some i with $(b_{max})_i = (b_{min})_i$, then for every j where $A_{i,j} > 0$ replace w_j by $w_j \cdot (b_{max})_i^{A_{i,j}}$ and set $A_{i,j} = 0$.
5. If there are two equal columns j_0, j_1 in A , then replace w_{j_0} by $w_{j_0} + w_{j_1}$ and remove the j_1 th column from A and w .

► **Lemma 30.** *Rule 5 is safe.*

Proof. For Item 1 and 2 observe that neither variables that do not occur in the represented polynomial nor products that contain zero as a factor contribute to the value of the polynomial. Item 3 follows from the fact that there cannot be a solution if the domain of a variable is the empty set. Finally, Item 4 follows by the distributivity of addition and multiplication. ◀

If this rule cannot be applied, the domain of any variable contains at least two consecutive values. With the next rule we change and shift the domains of all variables such that the domain of every variable contains 0 and 1.

► **Rule 6.** *Apply the following modifications if possible:*

1. If there is an i with $\{0, 1\} \not\subseteq [(b_{min})_i, (b_{max})_i]$ and $(b_{min})_i \in \mathbb{Z}$, then for every column j_0 where $A_{i,j_0} > 0$ add $c := A_{i,j_0}$ new j_1, j_2, \dots, j_c -th columns such that

$$A_{i',j_k} = \begin{cases} A_{i',j_0} & \text{if } i' \neq i \\ A_{i,j_0} - k & \text{if } i' = i \end{cases} \quad \text{and} \quad w_{j_k} = \binom{c}{k} \cdot w_{j_0} \cdot (b_{min})_i^k \quad \text{for } k \in \{1, \dots, c\}.$$

Finally set $(b_{min})_i$ to 0 and $(b_{max})_i$ to $(b_{max})_i - (b_{min})_i$.

2. If there is an i with $(b_{min})_i = -\infty$ and $(b_{max})_i \leq 0$, then replace w_j by $-w_j$ for every j where $A_{i,j}$ is odd, and set $(b_{min})_i$ to $-(b_{max})_i$ and $(b_{max})_i$ to ∞ .

► **Lemma 31.** *Rule 6 is safe.*

The first subrule shifts the domain of variables that have a finite lower bound. Variables without such a bound are turned into variables with a finite lower bound by the second subrule. The first subrule may then be applied again.

As soon as none of the rules can be applied, every variable that occurs in the polynomial $p_{A,w}$ has a domain that is a superset of $\{0, 1\}$. We can turn our reduced instance into an instance of $p_{\alpha,d}$ -ABS-DNF, which directly translates into an equivalent instance of $p_{\alpha,d}$ -UNBALANCED-SUBGRAPH.

Thereby we might turn an ABS-IO yes-instance into a no-instance of UNBALANCED-SUBGRAPH. Recall that running the kernelization algorithm from Section 4 (especially Rule 4) either returns a trivial yes-instance or does nothing. After applying Rule 5 on the initial instance, rules 1 and 2 cannot be applied on the hypergraph. Therefore, the initial instance is a yes-instance or the size of the matrix is bounded by the parameters (because (H, w) is a kernel). Note that this does not imply that we obtain a kernel: The values in the bounding vectors are eventually not bounded yet. However, we show that such problematic instances can be reduced to a set of equivalent instances in which we have control over all these values.

► **Theorem 4.** $p_{\alpha,d}$ -ABS-IO \in FPT

Proof. Let us firstly present some branching rule for our problem:

► **Branching Rule 1.** *Let there be some i^* where $(b_{max})_{i^*} - (b_{min})_{i^*} \geq 2e\alpha$ with $e = \max_j A_{i^*,j}$. Let $w^{(k)} \in \mathbb{Z}^m$ with $w_j^{(k)} = w_j$ if $A_{i^*,j} = k$ and $w_j^{(k)} = 0$ otherwise, and let A' be the matrix obtained by setting all values in the i^* -th row of A to 0. Then $(A, w, b_{min}, b_{max}, \alpha) \in \text{ABS-IO}$ iff there is some $k \in \{0, \dots, e\}$ such that $(A', w^{(k)}, b_{min}, b_{max}, \alpha_k) \in \text{ABS-IO}$ where $\alpha_k = 1$ for $k > 1$ and $\alpha_0 = \alpha$.*

▷ **Claim 32.** Branching Rule 1 is safe.

Proof. Let (x_1, \dots, x_n) be an arbitrary solution for the input instance. By term rewriting it follows that

$$\begin{aligned} p_{A,w}(x_1, \dots, x_n) &= \sum_{j=1}^m w_j \cdot \prod_{i=1}^n A\langle x_i, i, j \rangle \\ &= \sum_{k=0}^e \left(\sum_{\substack{j \in \{1, \dots, m\} \\ A_{i^*,j} = k}} w_j \cdot \prod_{i \in \{1, \dots, n\}} A\langle x_i, i, j \rangle \right) \\ &= \sum_{k=0}^e \left(\sum_{\substack{j \in \{1, \dots, m\} \\ A_{i^*,j} = k}} w_j^{(k)} \cdot \prod_{\substack{i \in \{1, \dots, n\} \\ i \neq i^*}} A'\langle x_i, i, j \rangle \cdot A\langle x_{i^*}, i^*, j \rangle \right) \\ &= \sum_{k=0}^e (p_{A',w^{(k)}}(x_1, \dots, x_n) \cdot A\langle x_{i^*}, i^*, k \rangle). \end{aligned}$$

We show that one of the branching instance has a solution, too. If $p_{A',w^{(k)}}(x_1, \dots, x_n) > 0$ for some $k > 0$, we are done. Otherwise $p_{A',w^{(k)}}(x_1, \dots, x_n) = 0$ holds for every $k > 0$. Then the equation above implies $p_{A',w^{(0)}}(x_1, \dots, x_n) = p_{A,w}(x_1, \dots, x_n)$. Since $|p_{A,w}(x_1, \dots, x_n)| \geq \alpha$, we get that $|p_{A',w^{(0)}}(x_1, \dots, x_n)| \geq \alpha$, which means $(A', w^{(0)}, b_{min}, b_{max}, \alpha) \in \text{ABS-IO}$.

It remains to show that the input instance is a yes-instance if one of the branching instances is a yes-instance. Firstly, consider $(A', w^{(0)}, b_{min}, b_{max}, \alpha)$. Note, that $p_{A',w^{(0)}}(x_1, \dots, x_n)$ does not depend on x_{i^*} because $w_j^{(0)} = 0$ for every product that does include x_{i^*} . For the same reason it also holds that $p_{A,w}(x_1, \dots, x_{i^*-1}, 0, x_{i^*+1}, \dots, x_n) = p_{A',w^{(0)}}(x_1, \dots, x_n)$. Therefore, we can turn any solution of $(A', w^{(0)}, b_{min}, b_{max}, \alpha)$ into a solution for the input instance $(A, w, b_{min}, b_{max}, \alpha)$ by replacing x_{i^*} by 0.

Now assume that $(A', w^{(0)}, b_{min}, b_{max}, \alpha)$ is a no-instance and consider the largest number $k \in \{1, \dots, e\}$ such that $(A', w^{(k)}, b_{min}, b_{max}, 1)$ is a yes-instance. Let (y_1, \dots, y_n) be one of its solutions. Define the polynomial $p(z) := p_{A,w}(y_1, \dots, y_{i^*-1}, z, y_{i^*+1}, \dots, y_n)$. It follows that $p(z) = \sum_{\ell=0}^k c_\ell \cdot z^\ell$ with $c_\ell = p_{A',w^{(\ell)}}(y_1, \dots, y_n)$. Note that for every z we have $|p_{A',w^{(\ell)}}(y_1, \dots, y_{i^*-1}, z, y_{i^*+1}, \dots, y_n)| \geq 1$ as $A'_{i^*,j} = 0$ for every column j and, thus, $c_k \neq 0$. We will now argue by curve sketching over \mathbb{R} that there is a $z \in \{(b_{min})_{i^*}, \dots, (b_{max})_{i^*}\} \cap \mathbb{Z}$ with $|p(z)| \geq \alpha$. Any interval $[a, b]$ with $b - a > 2\alpha$ where either (a, b) or $[a, b)$ contains no extremum of the polynomial contains at least one value $z \in \mathbb{Z}$ where $|p(z)| \geq \alpha$. This is because the polynomial is strictly increasing or decreasing on such an interval and the domain and image are restricted to \mathbb{Z} . The polynomial p has a degree of k , which is at most e , and, thus, it has at most $e - 1$ extreme points. Since $(b_{max})_{i^*} - (b_{min})_{i^*} \geq 2e\alpha$, there is such an interval within $[(b_{min})_{i^*}, (b_{max})_{i^*}]$. Hence, there is a $z \in [(b_{min})_{i^*}, (b_{max})_{i^*}]$ with $|p(z)| \geq \alpha$. By the definition of $p(z)$, we finally get that $(y_1, \dots, y_{i^*-1}, z, y_{i^*+1}, \dots, y_n)$ is a solution for the input instance. \triangleleft

Note that this rule reduces the number of variables on which the value of the polynomial depends. Recall Rule 5: As soon as a row contains only zeroes, it can be safely removed. Since the rules output at most $e \leq d$ branches, an exhaustive application of all the reduction rule results in a search tree of size $O(d^n)$. This search tree computes $O(d^n)$ instances in total, all of which have no variables with large domain (more than $2d\alpha$ possible values).

We partition the remaining part of the proof into three parts: First, we present the algorithm sketched in the main text in detail; second, we prove that this algorithm correctly solves $p_{\alpha,d}$ -ABS-IO; and third, we argue that the algorithm in fpt-time.

The Algorithm. Our algorithm has four phases. In the first phase it exhaustively applies the reduction rules 5 and 6 to obtain an instance $(A, w, b_{min}, b_{max}, \alpha)$. If Rule 5 returns a trivial no-instance, we reject the initial input. As soon as no reduction rule can be applied, it follows from Rule 5 that every variable in $p_{A,w}$ has a domain of size at least two. By Rule 6 every domain must be a superset of $\{0, 1\}$.

In the second phase we use the kernelization algorithm for $p_{\alpha,d}$ -UNBALANCED-SUBGRAPH: For the instance $(A, w, b_{min}, b_{max}, \alpha)$ obtained from the previous phase, let $H = (V, E)$ be a d -hypergraph and let $w: E \rightarrow \mathbb{Z}$ be a weight function with $V = \{1, \dots, n\}$, $E = \{e_1, \dots, e_m\}$, $e_j = \{i \mid A_{i,j} > 0\}$, and $w(e_j) = w_j$. We test whether $|E| \geq (2d^d 2^{2^d} \alpha)^{2^d - 1}$ and accept if so.

Otherwise $m = |E| \leq (2d^d 2^{2^d} \alpha)^{2^d - 1}$ and $n \leq d \cdot m$. Then we continue with the third phase and exhaustively apply Branching Rule 1 as well as the reduction rules (Rule 5 and Rule 6). This gives us a search tree that computes a set of instance on which neither a reduction rule nor the branching rule is applicable. The domain of all variables in all of these instances is bounded by the parameters and, hence, we can solve them via “brute-force.”

Proof of Correctness. Consider the first phase of the algorithm where rules 5 and 6 are applied in the given order as long as possible. Since the rules are safe, it is correct to stop and reject if Rule 5 returns a trivial no-instance. Otherwise we obtain an instance that has a solution if, and only if, the initial input instance does.

Since the reduction rules have been applied exhaustively in the first phase, the domain for every variable is a superset of $\{0, 1\}$ afterwards. Therefore, any solution for the constructed UNBALANCED-SUBGRAPH instance gives us a valid solution. By Rule 5 there is no isolated vertex in V nor an edge $e \in E$ with $w(e) = 0$ in the constructed hypergraph. This means the only rules that modify the hypergraph (Rule 1 and Rule 2) are not applicable. Because of that, from the proof of Corollary 23, and from Lemma 21, the test for the size of the hypergraph correctly identifies trivial yes-instances or does nothing. We can derive a solution $(x_1, \dots, x_n) \in \mathbb{Z}^n$ with $|p_{A,w}(x_1, \dots, x_n)| \geq \alpha$ from a solution X for our hypergraph where $|w[X]| \geq \alpha$ as follows by setting $x_i = 1$ if $i \in X$, and by setting $x_i = 0$ otherwise. This is correct as the term that we maximize for the UNBALANCED-SUBGRAPH instance is the same as $|p_{A,w}(x_1, \dots, x_n)|$, and because the domain of every variable contains 0 and 1.

In the third phase, the algorithm reduces the instance to a set of instances in which the domain of every variable is bounded by the parameter. Variables with a too large domain will be eliminated by the branching rule and a following application of Rule 5 to remove the row containing only zeros. From the safeness of the branching rule it directly follows that there is one yes-instance if, and only if, one of the final branching instances is a yes-instance. Since the number of possible solutions for each instance is finite, a brute-force algorithm will decide for each of these instances correctly whether it is a yes-instance.

Runtime Analysis. We first argue that applying the reduction rules exhaustively can be done in time $f(\alpha, d) \cdot \mu^c$, where μ is the encoding length of the input, f some computable function, and $c \in \mathbb{N}$ a constant.

Rule 6 removes a variable with a domain that is not a superset of $\{0, 1\}$ and no rule introduces new variables. Hence, Rule 6 is applied at most $O(n)$ times. Rule 5 can only be applied $O(n + m)$ times because subrules 1–4 can be applied only once initially; the fifth subrule can be applied after Rule 6. Hence the rules can only be applied a polynomial number of times. Observe that almost all reduction rules are computable in polynomial time with respect to the size of the input instance. The sole exceptions are Rule 6.1 and Rule 5.4, which still run in fpt-time.

We also have to show that the size of subproblems does not exceed the algorithm's time bound of $f(\alpha, d) \cdot \mu^c$. The critical part is Rule 6.1 – in all other cases the instance becomes only smaller. Each time we apply the rule for some i , the number of columns j for which $A_{i,j} > 0$ is increased by a factor of at most d as $A_{i,j} \leq d$. Every new column arises directly or indirectly from one column that was in the initial instance and, hence, the size of the instance increases at most by a factor of d^d in total.

For the search tree note that once the kernelization was used in the second phase, n and m are bounded by the parameters. The height and size of the search tree is therefore bounded, too. Note that the kernelization itself runs in polynomial time by the definition of a kernelization algorithm. Finally, whenever Branching Rule 1 cannot be applied, we have to solve the instance at the leaves of the search tree. Since neither a reduction rule nor the branching rule are applicable, it holds that n , m , all values in the matrix, and all values in the bounding vectors are bounded by the parameters for these instances. Thus, the brute-force algorithm's runtime is bounded by some computable function that depends only on the parameters (note that the weights are not necessarily bounded, but this is no problem for a combinatorial algorithm that enumerates the solutions). ◀

6 Conclusion and Outlook

We considered several variants of the maximum satisfiability problem and showed that, as soon as we allow negative weights, those variants become W[1]-hard parameterized by the solution size α – even for monotone clauses of fixed size d . On the other hand, we obtained fixed-parameter tractability results parameterized by $\alpha + d$ if we optimize the absolute value of the target function. The latter result was obtained via a kernelization for the auxiliary hypergraph problem $p_{\alpha,d}$ -UNBALANCED-SUBGRAPHS, which we think may be of independent interest. Using these techniques, we were able to almost completely resolve the complexity of ABS-DNF, see Table 1 in the introduction for an overview. The only remaining case is α -ABS-DNF, i. e., the version in which the sought solution size α is constant while the size of the clauses is unbounded. We do not see how our techniques can be used for this version, as our algorithms for the unbalanced subgraph problem rely on hyperedges of bounded size.

Using a collection of additional reduction rules, we were able to generalize the results from $p_{\alpha,d}$ -ABS-DNF to $p_{\alpha,d}$ -ABS-IO, which tries to optimize the absolute value of the target function of a restricted integer optimization problem.

An interesting line of further research could be to study the *minimization* version of the problems presented within this paper. Usually, minimization and maximization problems have similar complexity, as one can perform some easy modifications such as multiplying all weights with -1 . This is, however, not the case if we optimize the absolute value of the target function, as the following observation illustrates: Let d -MIN-ABS-MONOTONE-DNF be defined as d -ABS-MONOTONE-DNF, but with $\geq \alpha$ being replaced by $\leq \alpha$, then:

► **Observation 33.** $\text{INDEPENDENT-SET} \leq d\text{-MIN-ABS-MONOTONE-DNF}$

Sketch of Proof. We use the same reduction as in Lemma 5 to reduce INDEPENDENT-SET to $d\text{-MAX-MONOTONE-DNF}$. Hence, we obtain weighted formula (ϕ, w) and seek an assignment of weight at least $\geq \alpha$ (without absolute values). Note that if such an assignment exists, then there is also one with weight $= \alpha$ as the independent set problem is monotone. We continue by adding the empty clause (which is always true in a DNF) and set its weight to $-\alpha$. Finally, we seek a solution with absolute value ≤ 0 . ◀

Note that this reduction works for constant d and produces an instance with $\alpha = 0$, i. e., a parameterization by α and d alone is not enough. However, restricting, for instance, the weights may yield tractable subproblems that should be explored further.

References

- 1 N. Alon, G.Z. Gutin, E.J. Kim, S. Szeider, and A. Yeo. Solving max- r -SAT above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.
- 2 A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability, Second Edition*, volume 185 of *Front. Artif. Intell. Appl.* IOS Press, 2021.
- 3 J. Chen, C. Xu, and J. Wang. Dealing with 4-variables by resolution: An improved maxsat algorithm. *Theor. Comput. Sci.*, 670:33–44, 2017.
- 4 B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- 5 Y. Crama, O. Ekin, and P.L. Hammer. Variable and term removal from boolean formulae. *Discret. Appl. Math.*, 75(3):217–230, 1997.
- 6 R. Crowston, M. Fellows, G. Gutin, M. Jones, F. Rosamond, S. Thomassé, and A. Yeo. Simultaneously Satisfying Linear Equations Over F_2 : MaxLin2 and Max- r -Lin2 Parameterized Above Average. In *FSTTCS*, volume 13, pages 229–240, 2011.
- 7 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 8 H. Dell, E.J. Kim, M. Lampis, V. Mitsou, and T. Mömke. Complexity and approximability of parameterized max-csps. *Algorithmica*, 79(1):230–250, 2017.
- 9 E. Eiben, R. Ganian, D. Knop, and S. Ordyniak. Unary integer linear programming with structural restrictions. In *IJCAI*, pages 1284–1290. ijcai.org, 2018.
- 10 E. Eiben, R. Ganian, D. Knop, and S. Ordyniak. Solving integer quadratic programming via explicit and structural restrictions. In *AAAI*, pages 1477–1484. AAAI Press, 2019.
- 11 F. Eisenbrand, C. Hunkenschröder, K.M. Klein, M. Koutecký, A. Levin, and S. Onn. An algorithmic theory of integer programming, 2019. [arXiv:1904.01361](https://arxiv.org/abs/1904.01361).
- 12 P. Erdős and R. Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 1(1):85–90, 1960.
- 13 J. Flum and M. Grohe. *Parameterized Complexity Theory*. EATCS. Springer, 2006.
- 14 R. Ganian and S. Ordyniak. Solving integer linear programs by exploiting variable-constraint interactions: A survey. *Algorithms*, 12(12):248, 2019.
- 15 R. Ganian and S. Szeider. New width parameters for model counting. In *SAT*, volume 10491 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 2017.
- 16 M.R. Garey, D.S. Johnson, and L.J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.
- 17 S. Gaspers and S. Szeider. Kernels for global constraints. In *IJCAI*, pages 540–545. IJCAI/AAAI, 2011.
- 18 S. Gaspers and S. Szeider. Guarantees and limits of preprocessing in constraint satisfaction and reasoning. *Artif. Intell.*, 216:1–19, 2014.
- 19 T. Gavenčiak, D. Knop, and M. Koutecký. Integer Programming in Parameterized Complexity: Three Miniatures. In *IPEC*, volume 115, pages 21:1–21:16, 2019.

- 20 T. Gavenčiak, M. Koutecký, and D. Knop. Integer programming in parameterized complexity: Five miniatures. *Accepted for Discrete Optimization*, 2020. doi:10.1016/j.disopt.2020.100596.
- 21 M. Grohe. The structure of tractable constraint satisfaction problems. In *MFCS*, volume 4162, pages 58–72. Springer, 2006.
- 22 K. Jansen, A. Lassota, and L. Rohwedder. Near-linear time algorithm for n-fold ilps via color coding. *SIAM J. Discret. Math.*, 34(4):2282–2299, 2020.
- 23 S. Khanna, M. Sudan, and D.P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In *STOC*, pages 11–20, 1997.
- 24 M. Koutecký, A. Levin, and S. Onn. A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs. In *ICALP*, volume 107 of *LIPICs*, pages 85:1–85:14, 2018.
- 25 H.W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.
- 26 M. Mahajan and V. Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *J. Algorithms*, 31(2):335–354, 1999.
- 27 M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *J. Comput. Syst. Sci.*, 75(2):137–153, 2009.
- 28 N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to horn and binary clauses. In *SAT*, 2004.
- 29 M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.
- 30 M. Samer and S. Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.
- 31 T.J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.
- 32 S. Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discret. Optim.*, 8(1):139–145, 2011.

Dense Graph Partitioning on Sparse and Dense Graphs

Cristina Bazgan ✉ 🏠 

Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE,
75016 Paris, France

Katrin Casel ✉ 🏠 

Hasso Plattner Institut, Universität Potsdam, Germany

Pierre Cazals ✉ 

Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE,
75016 Paris, France

Abstract

We consider the problem of partitioning a graph into a non-fixed number of non-overlapping subgraphs of maximum density. The density of a partition is the sum of the densities of the subgraphs, where the density of a subgraph is half its average degree, that is, the ratio of its number of edges and its number of vertices. This problem, called Dense Graph Partition, is known to be NP-hard on general graphs and polynomial-time solvable on trees, and polynomial-time 2-approximable.

In this paper we study the restriction of Dense Graph Partition to particular sparse and dense graph classes. In particular, we prove that it is NP-hard on dense bipartite graphs as well as on cubic graphs. On dense graphs on n vertices, it is polynomial-time solvable on graphs with minimum degree $n - 3$ and NP-hard on $(n - 4)$ -regular graphs. We prove that it is polynomial-time $4/3$ -approximable on cubic graphs and admits an efficient polynomial-time approximation scheme on graphs of minimum degree $n - t$ for any constant $t \geq 4$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases NP-hardness, approximation, density, graph partitioning, bipartite graphs, cubic graphs, dense graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.13

Related Version *Full Version:* <https://arxiv.org/abs/2107.13282>

1 Introduction

The research around communities in social networks can be seen as a contribution to the well establish research of clustering and graph partitioning. Graph partitioning problems have been intensively studied with various measures in order to evaluate clustering quality, see e.g. [17, 18, 10, 6] for an overview. In the context of social networks, a ‘community’ is a collection of individuals who are relatively well connected compared to other parts of the social network graph. A ‘community structure’ then corresponds to a partition of the whole social network into communities.

We consider a classical definition of the density of a (sub)graph (see, for example, [12, 15, 8]) given by half its average degree, that is, the ratio between its number of edges and its number of vertices. For this definition of density, there are several papers on finding the densest subgraph. This problem was shown solvable in polynomial time by Goldberg [12] but if the size of the subgraph is a part on the input, the problem called k -DENSEST SUBGRAPH becomes NP-hard even restricted to bipartite or chordal graphs [7]. The approximability of k -DENSEST SUBGRAPH was also studied, see [14, 9, 4].



© Cristina Bazgan, Katrin Casel, and Pierre Cazals;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 13; pp. 13:1–13:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study the problem MAX DENSE GRAPH PARTITION that models finding a community structure, that is, finding a dense *partition*. More precisely, given an undirected graph G , we aim to find a partition $\mathcal{P} = \{V_1, \dots, V_k\}$, $k \geq 1$, of the vertices of G , such that sum of the densities of the subgraphs $G[V_i]$ is maximized. We denote the sum of the densities of the subgraphs $G[V_i]$ by $d(\mathcal{P})$, and call this the density of the partition \mathcal{P} .

Note that the general concept of a community structure does not put any restriction on the number of communities. We therefore address the problem MAX DENSE GRAPH PARTITION of finding a partition of maximum density, without fixing the number of classes of the partition. Indeed, when the number of classes is given, the problem is a generalization of a partition into k cliques. By not fixing the number of classes, MAX DENSE GRAPH PARTITION differs from partitioning into cliques: observe that while there exists a partition into exactly k sets of density $(n - k)/2$ if and only if the input graph can be partitioned into k cliques (see Lemma 2), there can be a partition into less than k sets with a density even higher than $(n - k)/2$ even if the input cannot be partitioned into k cliques. As an example, consider a complete graph of an even number n of vertices and turn four of the vertices into an independent set by removing all edges among them. The resulting graph cannot be partitioned into 3 cliques (at least one set contains two of the four independent vertices), but it has a partition into two sets of equal cardinality with density $(n - 2)/2 - 4/n$.

Darley et al. [8] studied MAX DENSE GRAPH PARTITION, and its complement MIN SPARSE GRAPH PARTITION. They defined the sparsity of a partition \mathcal{P} as $F(\mathcal{P}) = \frac{|\mathcal{P}|}{2} + d(\mathcal{P})$ and the problem MIN SPARSE GRAPH PARTITION as finding a partition of a given undirected graph G such that the sparsity of the partition is minimized. Observe that MAX DENSE GRAPH PARTITION and MIN SPARSE GRAPH PARTITION are dual in the sense that solving the first one on a graph G is the same as solving the second one on the complement of G . In [8] it is shown that both problems are NP-complete, and that there is no constant factor approximation for MIN SPARSE GRAPH PARTITION unless $P = NP$. Moreover, a polynomial time algorithm for MAX DENSE GRAPH PARTITION on trees is given. We point out that their proof of NP-completeness is a polynomial-time reduction from k -COLORING. By construction, the same reduction when starting from 3-COLORING on graphs of degree at most 4 (proved NP-complete in [11]) yields as instance of MAX DENSE GRAPH PARTITION a graph on n vertices and of minimum degree greater than $n - 4n^{4/5}$. Thus it follows that MAX DENSE GRAPH PARTITION is NP-complete restricted to graphs of minimum degree $n - 4n^{4/5}$.

Aziz et al. [2] studied the problem FRACTIONAL HEDONIC GAME, and more particularly the MAX UTILITARIAN WELFARE problem as the simple symmetric version of the game defined as follows. Let N be a set of agents, the utility of $i \in N$ in a coalition $S \subseteq N$ is $u_i(S) = \frac{1}{|S|} \sum_{j \in S} u_i(j)$ where $u_i(j)$ is such that $u_i(j) \in \{0, 1\}$ for a simple game and $u_i(j) = u_j(i)$ for a symmetric one. For MAX UTILITARIAN WELFARE one tries to find a partition C of N into coalitions that maximizes $\sum_{S \in C} \sum_{i \in S} u_i(S)$. This game can be seen as a graph G where agents are vertices and there is an edge between two agents i and j if and only if $u_i(j) = 1$. In this context, $u_i(S) = \frac{1}{|S|} \sum_{j \in S} u_i(j) = \frac{1}{|S|} \deg_{G[S]}(i)$. We deduce that $\sum_{S \in C} \sum_{i \in S} u_i(S) = \frac{1}{|S|} \sum_{S \in C} \sum_{i \in S} \deg_{G[S]}(i) = \frac{1}{|S|} \sum_{S \in C} 2|E(S)| = 2 \cdot d(C)$. Hence, the problems MAX UTILITARIAN WELFARE and MAX DENSE GRAPH PARTITION are equivalent to within a constant, which means that the 2-approximation for the former given in [2] directly translates to the latter.

Our contributions. The following overview summarises the results achieved in this paper concerning MAX DENSE GRAPH PARTITION (MDGP).

- MDGP is trivially solvable on graphs of maximum degree 2, we prove its NP-hardness for 3-regular (cubic) graphs.

- We establish that on bipartite complete graphs an optimal partition consists of one part, that is the whole graph. Moreover if the size of the two independent sets are relatively prime numbers then this optimal solution is unique. We use this result to show that MDGP is NP-hard on dense bipartite graphs.
- MDGP is trivial on complete graphs since the optimal solution is the whole graph as one part of the partition. Moreover, as we previously explained, it is NP-hard on graphs of minimum degree $n - 4n^{4/5}$. We show that for graphs of minimum degree $\geq n - 3$, the problem is solvable in polynomial time and any optimal solution has two parts. Moreover on $(n - 4)$ -regular graphs, the problem becomes NP-hard.
- We further improve on the 2-approximation for MDGP [2] for specific sparse and dense graph classes. In particular, we show that MDGP admits a polynomial-time $4/3$ -approximation on cubic graphs. Moreover we establish a polynomial-time $\frac{n-1}{\delta+1}$ -approximation, where δ is the minimum degree of the input graph (note that this improves on the ratio of 2 for all $\delta > \frac{n-3}{2}$). Also, we give an eptas (i.e. a $(1 + \varepsilon)$ -approximation for any $\varepsilon > 0$) on graphs of minimum degree $n - t$ for any constant $t \geq 4$

Our paper is organized as follows. Notations and formal definitions are given in Section 2. The study of (dense) bipartite graphs is established in Section 3. Section 4 presents the results on cubic graphs. In Section 5 we study dense graphs. Some conclusions are given at the end of the paper.

2 Preliminaries

In this paper we assume that all graphs are undirected, without loops or multiple edges, and not necessary connected. We use $G = (V, E)$ to denote an undirected graph with a set V of vertices and a set E of edges. We use $|V|$ to denote the number of vertices in G , i.e., the order of G , and we use $|E|$ to denote the number of edges in G , i.e., the size of G . We denote by $\deg_G(v)$ the degree of $v \in V$ in G that is the number of edges incident to v and by $D_G(i)$ the set of vertices of degree i in G . The maximum degree of G , denoted by $\Delta(G)$, is the degree of the vertex with the greatest number of edges incident to it. The minimum degree of G , denoted by $\delta(G)$, is the degree of the vertex with the least number of edges incident to it. For any vertex $v \in V$, $N_G(v)$ is the set of neighbors of v in G and $N_G[v] = N_G(v) \cup \{v\}$. Moreover, $N_G(S) = \bigcup_{v \in S} N_G(v)$. For a graph $G = (V, E)$ and a subset $S \subseteq V$ we denote by $E(S)$ the set of the edges of G with both endpoints in S . For a given partition $\{A, B\}$ of V , we denote by $E(A, B) = \{uv \in E : u \in A, v \in B\}$. Further, $G[S]$ denotes the graph induced by S , defined as $G[S] = (S, E(S))$.

A triangle graph is the cycle graph C_3 or the complete graph K_3 . A diamond graph has 4 vertices and 5 edges, it consists of a complete graph K_4 minus one edge. A graph is called cubic if all its vertices are of degree three. A graph is bipartite if its vertices can be partitioned into two sets A and B such that every edge connects a vertex in A to one in B . A complete bipartite graph is a special kind of bipartite graph where every vertex of A is connected to every vertex in B . A graph on n vertices is δ -dense if its minimum degree is at least δn . A set of instances is called dense if there is a constant $\delta > 0$ such that all instances in this set are δ -dense (this notion was introduced in [1] and called everywhere-dense).

The density $d(G)$ of a graph $G = (V, E)$ is the ratio between the number of edges and the number of vertices in G , that is, $d(G) = \frac{|E|}{|V|}$. Moreover, for $S \subseteq V$, $d(S) = d(G[S]) = \frac{|E(S)|}{|S|}$. We use \mathcal{P} to denote a partition of the set V of vertices of G , that is, $\mathcal{P} = \{V_1, \dots, V_k\}$, where $\bigcup_{i=1}^k V_i = V$, and $V_i \cap V_j = \emptyset$ for each $i, j \in \{1, \dots, k\}$. Then the density of a partition \mathcal{P} of G is defined as $d(\mathcal{P}) = \sum_{i=1}^k d(G[V_i])$, where $G[V_i]$ is the subgraph of G induced by the subset V_i of vertices, that is, $G[V_i] = (V_i, E_i)$, $E_i = \{\{u, v\} : \{u, v\} \in E \wedge u, v \in V_i\}$.

13:4 Dense Graph Partitioning on Sparse and Dense Graphs

We study the problem of finding a partition $\mathcal{P} = \{V_1, \dots, V_k\}$ of a given graph G , such that $k \geq 1$ and that, among all such partitions, $d(\mathcal{P})$ is maximized. We refer to this problem as MAX DENSE GRAPH PARTITION and we define its decision version as follows.

DENSE GRAPH PARTITION

Input: An undirected graph $G = (V, E)$, a positive rational number r .

Question: Is there a partition \mathcal{P} such that $d(\mathcal{P}) \geq r$?

Given an optimization problem in NPO and an instance I of this problem, we denote by $|I|$ the size of I , by $opt(I)$ the optimum value of I , and by $val(I, S)$ the value of a feasible solution S of instance I . The performance ratio of S (or approximation factor) is $r(I, S) = \max\{\frac{val(I, S)}{opt(I)}, \frac{opt(I)}{val(I, S)}\} \geq 1$. For a function f , an algorithm is an $f(|I|)$ -approximation, if for every instance I of the problem, it returns a solution S such that $r(I, S) \leq f(|I|)$. Moreover if the algorithm runs in polynomial time in $|I|$, then this algorithm gives a polynomial-time $f(|I|)$ -approximation. We consider in this paper only polynomial time algorithms. When f is a constant α , the problem is polynomial-time α -approximable. When $f = 1 + \varepsilon$, for any $\varepsilon > 0$, the problem admits a polynomial-time approximation scheme. When the running time of an approximation scheme is of the form $O(g(1/\varepsilon)poly(|I|))$ the problem has an efficient polynomial-time approximation scheme (eptas).

Before we start studying specific graph classes, we observe the following helpful structural properties that hold for DENSE GRAPH PARTITION on general graphs.

► **Remark 1.** We can assume that for any optimal partition \mathcal{P} and for any part $P \in \mathcal{P}$, $G[P]$ is connected, since otherwise turning each connected component into its own part does not decrease the density.

When discussing the density of a (sub)graph, it is often useful to think about how close this subgraph is to being a clique. We therefore call a pair of non-adjacent vertices in a (sub)graph a *missing edge*, and use the number of such missing edges to estimate the density of the (sub)graph. With such estimations, it is easy to show that the following intuition about favouring complete graphs as communities.

► **Lemma 2.** *Among all partitions of G into $t \geq 2$ parts, those where the parts correspond to complete graphs, if there exists such, have the largest density.*

Proof. Consider a partition of G into t parts $\{V_1, \dots, V_t\}$ of size n_1, \dots, n_t . If $G[V_i]$ has o_i missing edges for any $1 \leq i \leq t$, then the density of this partition is $\frac{n-t}{2} - \frac{o_1}{n_1} - \dots - \frac{o_t}{n_t}$.

Consider a partition of G into t parts of size n'_1, \dots, n'_t such that each part induces a complete graph for any $1 \leq i \leq t$. Then the density of this partition is $\frac{n-t}{2}$ and thus it is larger than the density of any partition in t parts where at least one edge is missing inside $G[V_i]$ for some $1 \leq i \leq t$. ◀

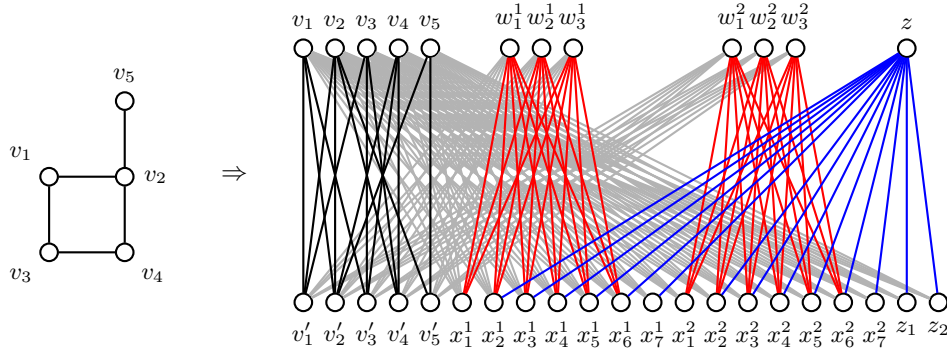
A direct consequence of this is the following.

► **Lemma 3.** *Let $G = (V, E)$ be a graph and \mathcal{P} be any partition of V . Then $d(\mathcal{P}) \leq \frac{|V|}{2} - \frac{|\mathcal{P}|}{2}$.*

3 Dense Bipartite Graphs

In this section we show that MAX DENSE GRAPH PARTITION has a trivial solution on complete bipartite graphs. Moreover, using this result we show that the problem is NP-hard on dense bipartite graphs.

Considering all possibilities to partition a complete bipartite graph $G_{n,m}$ with the two subsets that are independent sets of size n and m one can derive the following.



■ **Figure 1** A graph G , instance of DOMINATING SET and the bipartite graph G' obtained from G , for $k = 2$ and $n = 5$.

► **Lemma 4.** *The density $d(G_{n,m})$ of a complete bipartite graph $G_{n,m}$ is greater than or equal to the density $d(\mathcal{P})$ of any partition \mathcal{P} of $G_{n,m}$.*

It follows that an optimal solution of any complete bipartite graph is the whole graph. The calculations used to prove Lemma 4 inductively also give the following.

► **Corollary 5.** *For any complete bipartite graph $G = (A, B, E)$ with $|A| = n$ and $|B| = m$, a partition $\mathcal{P} = \{V_1, \dots, V_k\}$ of $A \cup B$ satisfies $d(\mathcal{P}) = \frac{nm}{n+m}$ if and only if $G[V_i] = G_{n_i, m_i}$ with $n_i \neq 0$ and $m_i \neq 0$ and $\frac{n_i}{m_i} = \frac{n}{m}$ for all $i \in \{1, \dots, k\}$.*

Consequently, for any complete bipartite graph $G_{n,m}$, if n and m are relatively prime the only optimal solution of $G_{n,m}$ is the whole graph. Otherwise, several optimal solutions exist and are characterized exactly by Corollary 5.

► **Theorem 6.** *DENSE GRAPH PARTITION is NP-hard on dense bipartite graphs.*

Proof. We give a reduction from DOMINATING SET. Let $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$ and an integer $k \geq 1$ be an instance of DOMINATING SET. Assume without loss of generality that G is connected. We first construct a bipartite graph $G' = (V_1, V_2, E')$, that is not dense, and show how solving DENSE GRAPH PARTITION on it solves DOMINATING SET on G . In a second step, we show how to make G' dense maintaining the reduction.

We construct $G' = (V_1, V_2, E')$ as follows:

- $V_1 = V \cup \{w_i^j : 1 \leq i \leq n - k, 1 \leq j \leq k\} \cup \{z\}$
- $V_2 = V' \cup \{x_r^j : 1 \leq r \leq N, 1 \leq j \leq k\} \cup \{z_i : 1 \leq i \leq N - n\}$ where $V' = \{v'_1, \dots, v'_n\}$ and $N \in \mathbb{N}$ is chosen as follows. Let $c \in \mathbb{N}$ be the smallest integer such that $c(n - k + 1) - 1 > n$ (note that $1 \leq c \leq n$) and define $N = c(n - k + 1) - 1$. For this choice of N it follows that the greatest common divisor of N and $n - k + 1$ is 1, and $n < N \leq 2n$.
- $E' = E_d \cup E_{wx} \cup E_c \cup E_z$ with
 - $E_d = \{\{v_i, v'_j\} : \{v_i, v_j\} \in E\} \cup \{\{v_i, v'_i\} : 1 \leq i \leq n\}$,
 - $E_{wx} = \{\{w_i^j, x_r^j\} : 1 \leq i \leq n - k, 1 \leq r \leq N - 1, 1 \leq j \leq k\}$,
 - $E_c = \{\{w_i^j, v'_s\} : 1 \leq i \leq n - k, 1 \leq j \leq k, 1 \leq s \leq n\} \cup \{\{v_s, x_r^j\} : 1 \leq s \leq n, 1 \leq r \leq N, 1 \leq j \leq k\}$ and
 - $E_z = \{\{z, z_j\} : 1 \leq j \leq N - n\} \cup \{\{z, x_r^j\} : 2 \leq r \leq N, 1 \leq j \leq k\} \cup \{\{v_i, z_j\} : 1 \leq i \leq n, 1 \leq j \leq N - n\}$

Notice that G' is a bipartite graph with $|V_1| = n + 1 + k(n - k)$ and $|V_2| = (k + 1)N$.

We show that there exists a dominating set of cardinality at most k in G if and only if there exists a partition \mathcal{P} of G' with $d(\mathcal{P}) = (k + 1)d(G_{n-k+1, N})$.

Suppose there exists a dominating set D in G with $|D| = k$. Let $D = \{v_{i_1}, \dots, v_{i_k}\}$ and $N'(v_{i_j}) = N_G[v_{i_j}] \setminus (D \cup N_G(\{v_{i_1}, \dots, v_{i_{j-1}}\}))$. Define the partition $\mathcal{P} = \{P_1, \dots, P_{k+1}\}$ by: $P_j = \{v_{i_j}\} \cup \{v_r : v_r \in N'(v_{i_j})\} \cup \{w_r^j : 1 \leq r \leq n - k\} \cup \{x_r^j : 1 \leq r \leq N - |N'(v_{i_j})|\}$ for $1 \leq j \leq k$ and $P_{k+1} = V_1 \cup V_2 \setminus (\cup_{j=1}^k P_j)$. With this definition, \mathcal{P} is clearly a partition of $V_1 \cup V_2$, and each part P_j contains $n - k + 1$ vertices from V_1 and N vertices from V_2 for each $1 \leq j \leq k + 1$. Further, each P_j induces a complete bipartite graph $G_{n-k+1, N}$: All vertices w_r^j and x_r^j are connected to each other, and to all vertices in V_2 and V_1 , respectively, by construction. Further, v_{i_j} is connected in G' to all vertices in $N'(v_{i_j})$; note here that in G' we connected v_i to its “copy” v_i' for all $1 \leq i \leq n$, which models the case that v_{i_j} dominates itself. For P_{k+1} , note that z is adjacent to all x_i^j -vertices, and each z_i is adjacent to all vertices in V . Since D is a dominating set, each vertex from V' is contained in some $N'(v_{i_j})$, thus $V_2 \setminus (\cup_{j=1}^k P_j)$ only contains x_i^j -vertices. Also, the P_j contain all w_i^j vertices and hence $V_1 \setminus (\cup_{j=1}^k P_j)$ only contains vertices from V .

Conversely, let \mathcal{P} be a partition of G' of density $(k + 1)d(G_{n-k+1, N})$. Thus, Corollary 5 implies that the vertices for each set $P \in \mathcal{P}$ induce a complete bipartite graph $G_{r, s}$ such that $\frac{r}{s} = \frac{|V_1|}{|V_2|} = \frac{k(n-k)+n+1}{(k+1)N} = \frac{n-k+1}{N}$. Since the greatest common divisor of $n - k + 1$ and N is one, this yields $r \geq n - k + 1$ and $s \geq N$ and especially \mathcal{P} can contain at most $k + 1$ sets.

For all w_i^j and w_ℓ^t , if $j \neq t$, w_i^j and w_ℓ^t have n common neighbors, and since $n < N$ there is no part $P \in \mathcal{P}$ such that $w_i^j, w_\ell^t \in P$. Moreover, for all i, j , w_i^j and z have $N - 1$ common neighbors so they also cannot be in the same $P \in \mathcal{P}$. Hence, there are exactly $k + 1$ parts in \mathcal{P} that are complete bipartite graphs $G_{n-k+1, N}$.

For all $1 \leq j \leq k$, denote by P_j the set containing the vertices w_i^j for all $1 \leq i \leq n - k$ and P_z the set containing z . To reach cardinality exactly $n - k + 1$, $P_j \cap V_1$ has to contain exactly one vertex from V for each $1 \leq j \leq k$. Further, since for any i , v_i' is not adjacent to z , $V' \subseteq \cup_{j=1}^k P_j$. As each $P \in \mathcal{P}$ induces a complete bipartite graph in G' , $D = V \cap \cup_{j=1}^k P_j$ is a set of size k , such that each vertex in V' is adjacent to at least one vertex in D , so we deduce that D is a dominating set of size k in G .

We extend the construction of the proof to create from G' a dense bipartite graph $G'' = (V'', E'')$ by adding four sets of vertices $V_1^u, V_1^d, V_2^u, V_2^d$ with $|V_1^u| = |V_1^d| = kn|V_1| = kn(k(n-k)+n+1)$ and $|V_2^u| = |V_2^d| = kn|V_2| = knN(k+1)$. Further, we add edges to turn the pairs (V_1^u, V_2^u) , (V_1^d, V_2^d) , (V_1^u, V_1) , and (V_2^d, V_2) each into complete bipartite graphs. Observe that with this construction G'' has $|V''| = (2kn + 1)(k(n - k) + n + 1) + (2kn + 1)N(k + 1) < 10k^2n^2$ vertices and that all vertices have degree at least $kn|V_1| \geq \frac{1}{2}k^2n^2 \in \Theta(|V''|)$. (Note that if $k \geq \frac{n}{2}$, G is a trivial yes-instance for DOMINATING SET.)

We claim that there exists a partition \mathcal{P}' of G'' with $d(\mathcal{P}') = (k + 1)d(G_{n-k+1, N}) + 2kn(k + 1)d(G_{n-k+1, N})$ if and only if there exists a dominating set of size k for G . Corollary 5 again implies that this density for G'' can only be achieved by a partition into complete bipartite graphs $G_{r, s}$ with $\frac{r}{s} = \frac{(2kn+1)(k(n-k)+n+1)}{(2kn+1)N(k+1)} = \frac{n-k+1}{N}$. The vertices in V_1^d are only adjacent to vertices in V_2^d , and the vertices in V_2^u are only adjacent to vertices in V_1^u . Clustering these in a ratio $\frac{r}{s}$ results in clusters containing exactly all newly added vertices, and this can be done with just two sets in total. What remains is to cluster the graph G' into complete bipartite graphs $G_{r, s}$ such that $\frac{r}{s} = \frac{|V_1|}{|V_2|} = \frac{k(n-k)+n+1}{(k+1)N} = \frac{n-k+1}{N}$ as before. ◀

4 Cubic Graphs

In this section, we study DENSE GRAPH PARTITION on cubic graphs, show that it remains NP-complete on this restricted graph class, but also give a polynomial time $\frac{4}{3}$ -approximation for its optimization variant MAX DENSE GRAPH PARTITION. We start with some general observations on the structure of communities in cubic graphs.

► **Definition 7.** For $P \subseteq V$, the utility of a vertex $v \in P$ is defined by $u_P(v) = \frac{d(P)}{|P|}$, and the utility of P is defined by $u(P) = u_P(v)$ for any $v \in P$. For a partition $\mathcal{P} = \{V_1, \dots, V_k\}$, the utility of a vertex v in \mathcal{P} is defined by $u_{\mathcal{P}}(v) = u_{V_i}(v)$ with i such that $v \in V_i$.

Considering these definitions, we can remark that:

- For any subset $P \subseteq V$, and $v, w \in P$, $u_P(v) = u_P(w)$.
- If $P = \{v\}$ then $u_P(v) = 0$.
- For any partition \mathcal{P} of G , $\sum_{V_i \in \mathcal{P}} d(V_i) = \sum_{v \in V} u_{\mathcal{P}}(v)$.

► **Lemma 8.** Let $G = (V, E)$ be a cubic graph without connected components that induce a K_4 . For any partition \mathcal{P} of G the following holds:

- $u_{\mathcal{P}}(v) \leq \frac{1}{3}$ for all vertices $v \in V$
- if $P \in \mathcal{P}$ is not a triangle, diamond or Case 1 in Figure 2 then $u(P) \leq \frac{1}{4}$

Proof. Let \mathcal{P} be a partition of G , $P \in \mathcal{P}$ and $v \in P$. Since G is cubic, $d(P) \leq \frac{3|P|}{2|P|} = \frac{3}{2}$. Then $u_{\mathcal{P}}(v) \leq \frac{3}{2|P|}$. If $|P| \geq 6$, $u_{\mathcal{P}}(v) \leq \frac{3}{2 \cdot 6} = \frac{1}{4}$. For $|P| = 5$ it follows that $u_{\mathcal{P}}(v) \leq \frac{7}{25} < \frac{1}{3}$, since a cubic graph on 5 vertices cannot have more than 7 edges. Also, since there exists no K_4 in G , the only graph on 5 vertices with 7 edges is Case 1 in Figure 2, and all other graphs on 5 vertices have 6 or less edges which yields a utility of at most $\frac{6}{25} < \frac{1}{4}$.

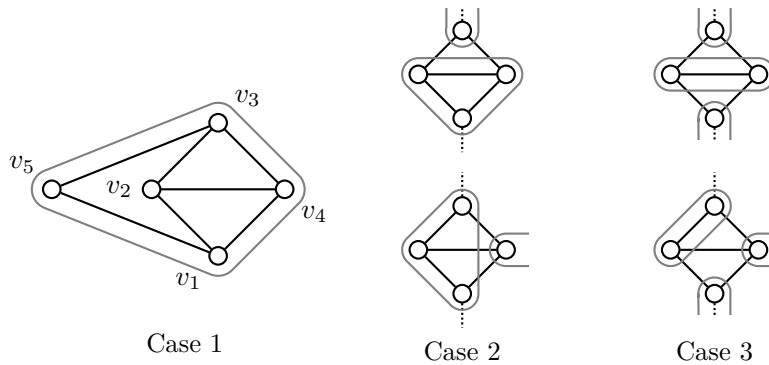
Case analysis on the graphs of size 4 or less yields that the largest utility is achieved for P being a triangle, which gives $u_{\mathcal{P}}(v) = \frac{1}{3}$. Further, if P is not a triangle or a diamond, case analysis on the graphs of size 4 or less shows that $u_{\mathcal{P}}(v)$ is maximized when P is an induced matching and its value is $\frac{1}{4}$. ◀

Considering all cases to distribute vertices of a diamond, as illustrated in Figure 2 can be used to show the following.

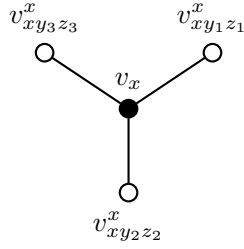
► **Lemma 9.** Let G be a cubic graph without connected components that induce a K_4 , and let v_1, v_2, v_3, v_4 be vertices in G that induce a diamond. Then $u_{\mathcal{P}}(v_1) + u_{\mathcal{P}}(v_2) + u_{\mathcal{P}}(v_3) + u_{\mathcal{P}}(v_4) \leq \frac{5}{4}$ for any partition \mathcal{P} for G .

► **Lemma 10.** Let G be a cubic graph on n vertices without connected components that induce a K_4 , and let D be the set of diamonds in G and T the set of triangles in G that do not belong to a diamond. For any partition \mathcal{P} , $d(\mathcal{P}) \leq \frac{5}{4}|D| + |T| + \frac{1}{4}(n - 3|T| - 4|D|)$.

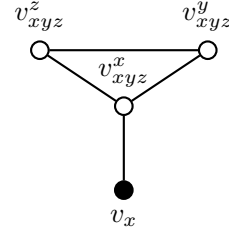
Proof. By Lemma 8, the only vertices with utility more than $\frac{1}{4}$ are those that are in triangles, diamonds, or the unique neighbors of diamonds (in the sense of vertex v_5 in Case 1 of Figure 2), and we know that the sum of the utilities of the vertices constituting a triangle



■ **Figure 2** Different cases of Lemma 9.



■ **Figure 3** Subgraph containing one vertex of type 1, v_x , and its neighbors in G .



■ **Figure 4** Subgraph containing one vertex of type 1, v_x , and three of type 2.

is at most $3 \cdot \frac{1}{3} = 1$. By Lemma 9, we further know that the sum of the utilities of the vertices constituting a diamond is at most $\frac{5}{4}$. The unique neighbors of diamonds have a utility of more than $\frac{1}{4}$ if and only if they are in a part isomorphic to Case 1 of Figure 2, which has a density of $\frac{7}{5} < \frac{5}{4} + \frac{1}{4}$. Thus, if S is the set of unique neighbors of diamonds, then the sum of the utilities of the vertices in the diamonds in D and the vertices in S is at most $\frac{5}{4}|D| + \frac{1}{4}|S|$. All remaining vertices have a utility of at most $\frac{1}{4}$ by Lemma 8. We deduce that $d(G) \leq \frac{5}{4}|D| + \frac{1}{4}|S| + |T| + \frac{1}{4}(n - 3|T| - 4|D| - |S|) = \frac{5}{4}|D| + |T| + \frac{1}{4}(n - 3|T| - 4|D|)$. ◀

We will reduce from the following problem, known to be NP-hard by [13]:

RESTRICTED EXACT COVER BY 3-SETS (RX3C)

Input: A set X of elements with $|X| = 3q$ and a collection C of 3-element subsets of X where each element appears in exactly 3 sets.

Question: Does C contain an exact cover for X , i.e. a subcollection $C' \subseteq C$ such that every element occurs in exactly one member of C' ?

The following definition gives the construction to reduce RX3C to DENSE GRAPH PARTITION.

► **Definition 11.** Let $I = (X, C)$ be an instance of RX3C. We define the construction σ transforming I into the graph $G = (V, E) := \sigma(I)$ as follows (see Figures 3 and 4):

- for each element $x \in X$, add the vertex v_x to V (called vertices of type 1 or black vertices).
- for each subset of the collection $\{x, y, z\} \in C$, add the vertices $v_{xyz}^x, v_{xyz}^y, v_{xyz}^z$ to V (called vertices of type 2 or white vertices).
- add the edges $\{v_{xyz}^x, v_{xyz}^y\}, \{v_{xyz}^x, v_{xyz}^z\}$ and $\{v_{xyz}^y, v_{xyz}^z\}$ to E
- add the edges $\{v_{xyz}^x, v_x\}, \{v_{xyz}^y, v_y\}$ and $\{v_{xyz}^z, v_z\}$ to E

Notice that G is a cubic graph on $|X|$ vertices of type 1 and $3|X|$ vertices of type 2.

Case distinction on the subgraphs in $\sigma(I)$ can be used to show:

► **Lemma 12.** For $G = (V, E) = \sigma(I)$ and any $P \subseteq V$, it holds that $u(P) \geq \frac{1}{4}$ if and only if $G[P]$ is isomorphic to one of the following three graphs:

- a triangle where all the vertices are of type 2 and then $u(P) = \frac{1}{3}$.
- an edge with at least one vertex of type 2, and then $u(P) = \frac{1}{4}$.
- the subgraph described in Figure 4 and then $u(P) = \frac{1}{4}$.

► **Remark 13.** The case-analysis in the proof of Lemma 12 also shows that for any subset $P \subseteq V$ of the vertices of the graph $\sigma(I)$, if v is of type 2 then $u_S(v) \leq \frac{1}{3}$, otherwise $u_S(v) \leq \frac{1}{4}$.

► **Theorem 14.** DENSE GRAPH PARTITION is NP-complete on cubic graphs.

Proof. Let $I = (X, C)$ be an RX3C instance. We claim that I is a yes-instance of RX3C if and only if $I' = (G, \frac{7|X|}{6})$ with $G = \sigma(I)$ is a yes-instance of DENSE GRAPH PARTITION.

Let $C' \subseteq C$ be an exact cover for X of size $\frac{|X|}{3}$. Consider the partition \mathcal{P} built from the following $\frac{5|X|}{3}$ parts: for any $c \in C'$, $c = \{x, y, z\}$ add $\{v_x, v_{xyz}^x\}$, $\{v_y, v_{xyz}^y\}$, $\{v_z, v_{xyz}^z\}$ and for any $c \notin C'$, $c = \{x, y, z\}$ add $\{v_{xyz}^x, v_{xyz}^y, v_{xyz}^z\}$. Since C' is an exact cover, \mathcal{P} is a partition for G and its density is $\frac{3}{2} \cdot \frac{|X|}{3} + \frac{2}{3}|X| = \frac{7}{6}|X|$.

Let \mathcal{P}' be a partition of G of density $d(\mathcal{P}') = \frac{7}{6}|X|$. Firstly, we show that \mathcal{P}' has necessarily the following shape: $\frac{2|X|}{3}$ parts of size 3 containing only vertices of type 2 forming a triangle and $|X|$ parts of size 2 containing one vertex of type 1 and one of type 2 adjacent in G . From Remark 1, we can assume that all parts induce connected subgraphs.

We first show that $d(\mathcal{P}') = \frac{7|X|}{6}$ implies that there are at least $\frac{2|X|}{3}$ parts in \mathcal{P}' which are triangles. Assume by contradiction that \mathcal{P}' has $\frac{2|X|}{3} - \ell$ triangles, with $\ell > 0$. Since G has $4|X|$ vertices, there are $2|X| + 3\ell$ vertices that do not belong to a part in \mathcal{P}' that corresponds to a triangle. By Lemma 12 the utility of these vertices is at most $\frac{1}{4}$ which yields:

$$d(\mathcal{P}') \leq \frac{2|X|}{3} - \ell + (2|X| + 3\ell) \cdot \frac{1}{4} = \frac{7|X|}{6} - \frac{\ell}{4} < \frac{7|X|}{6}.$$

This contradicts the choice of \mathcal{P}' such that $d(\mathcal{P}') = \frac{7|X|}{6}$, hence \mathcal{P}' has at least $\frac{2|X|}{3}$ triangles.

Assume by contradiction that \mathcal{P}' has $\frac{2|X|}{3} + \ell$ triangles, with $\ell > 0$. Since there are $3|X|$ vertices of type 2 and among these vertices $3 \cdot (\frac{2|X|}{3} + \ell)$ belong to a triangle, $|X| - 3\ell$ vertices of type 2 do not belong to a triangle. Each neighbor of a vertex v_x of type 1 is of type 2, so if the utility of v_x is positive, there exists a vertex of type 2, v_{xyz}^x , neighbor of v_x , that is in the same part as v_x and v_{xyz}^x does not belong to a triangle. Moreover, as all type 1 vertices have no common neighbors, for each type 1 vertex with positive utility, there is a type 2 vertex that is not in a triangle. Since there are at most $|X| - 3\ell$ type 2 vertices that do not belong to a triangle, there are at most $|X| - 3\ell$ type 1 vertices with positive utility, thus:

$$d(\mathcal{P}') \leq \frac{2|X|}{3} + \ell + \frac{|X| - 3\ell}{4} + \frac{|X| - 3\ell}{4} \leq \frac{7|X|}{6} - \frac{\ell}{2} < \frac{7|X|}{6}.$$

This again contradicts the choice of \mathcal{P}' with $d(\mathcal{P}') = \frac{7|X|}{6}$, thus \mathcal{P}' has exactly $\frac{2|X|}{3}$ triangles.

We claim that $d(\mathcal{P}') = \frac{7|X|}{6}$ implies that all type 1 vertices are in a part that is a matching with a type 2 vertex. There are $|X|$ type 1 vertices and $|X|$ type 2 vertices that are not in some triangle in \mathcal{P}' . Since there are exactly $\frac{2|X|}{3}$ parts in \mathcal{P}' forming a triangle and the utility of each other vertex is smaller than or equal to $\frac{1}{4}$, to reach a density of $\frac{7|X|}{6}$ it is necessary that each of the $2|X|$ vertices outside the parts that are triangles has a utility of exactly $\frac{1}{4}$. To reach this utility, by Lemma 12 there are two possibilities, the graph described in Figure 4 and an edge. Since there are exactly $|X|$ vertices of type 1 and $|X|$ vertices of type 2 outside the triangles in \mathcal{P}' , and vertices of type 1 only have neighbors of type 2, the only possibility is if each type 1 vertex is matched with one type 2 vertex.

Consider now the following subcollection $C'' \subseteq C$: for each triple $v_{xyz}^x, v_{xyz}^y, v_{xyz}^z$ that does not belong to a triangle, we add the set $\{x, y, z\}$ to C'' . The subcollection C'' is a cover since each type 1 vertex is a neighbor of one of these vertices and it is an exact cover since there are exactly $\frac{|X|}{3}$ 3-element subsets that do not belong to a triangle. ◀

Our observations about the maximum utility of certain vertices can also be used to show the following positive result.

► **Theorem 15.** *MAX DENSE GRAPH PARTITION is polynomial-time $\frac{4}{3}$ -approximable on cubic graphs.*

13:10 Dense Graph Partitioning on Sparse and Dense Graphs

Proof. Let $I = G$ be a cubic graph, instance of MAX DENSE GRAPH PARTITION. If G contains connected components isomorphic to K_4 , create a part for each such component, as this is the optimum way to partition these sets. Then, let D be the set of all diamonds in G , and T the set of all triangles that do not belong to a diamond. Diamonds (resp. triangles) can be found in polynomial time simply by enumerating all 4-tuples (resp. 3-tuples) of vertices and checking if they induce a diamond (resp. triangle) as subgraph. Let G' be the graph obtained from G after removing the vertices of D and T . Compute a maximum matching M of G' , and let G'' be the graph obtained from G' after removing the vertices of M . Since M is a maximal matching, the vertices in G'' form an independent set.

We show in the following that $|V(G'')| \leq \frac{|V(G)|}{4}$. For each $v \in V$ we associate a function $t(v)$ and initialize it with $t(v) = 1$. When removing the diamonds and triangles from G in order to get G' we update the function t as follows:

- For every diamond $\{u_1, u_2, u_3, u_4\} \subseteq V$ that is deleted from V , let u_1 and u_3 be the vertices with neighbors outside of the diamond (if these vertices still exist) and let v_1 and v_3 be these neighbors (with the possibility that $v_1 = v_3$). We update the function $t : t(v_1) := t(v_1) + t(u_1) + t(u_2)$ and $t(v_3) := t(v_3) + t(u_3) + t(u_4)$ (thus $t(v_1) := t(v_1) + t(u_1) + t(u_2) + t(u_3) + t(u_4)$ if $v_1 = v_3$). If v_1 or v_3 were already deleted, we delete their associated t function.
- For every triangle $\{u_1, u_2, u_3\} \subseteq V$ that is deleted from V , let v_1 (resp. v_2 and v_3) be the neighbor of u_1 (resp. u_2 and u_3) outside of the triangle (if these vertices exist). We update the function $t : t(v_1) := t(v_1) + t(u_1)$, $t(v_2) := t(v_2) + t(u_2)$ and $t(v_3) := t(v_3) + t(u_3)$. If v_1, v_2 or v_3 do not exist, we delete their associated t function.

Observe that after updating t for any $v \in V(G')$, if $v \in D_{G'}(3)$ then $t(v) \geq 1$, if $v \in D_{G'}(2)$ then $t(v) \geq 2$, if $v \in D_{G'}(1)$ then $t(v) \geq 3$ and if $v \in D_{G'}(0)$ then $t(v) \geq 4$. In order to justify this, observe that the t function associated to vertices in $V(G')$ cannot decrease. If a vertex v is of degree $3 - i$ in G' , $1 \leq i \leq 3$, then there are at least i adjacent edges to distinct vertices in triangles or diamonds that were removed from G and increase $t(v)$. Each time when a neighbor of a vertex v from a diamond or a triangle is removed then $t(v)$ increases by at least one. Then, in G' , each vertex v of degree $3 - i$ has $t(v) \geq i + 1$.

Let n'_i be the number of vertices of degree i in G' . By the previous remark, we have

$$\sum_{v \in V(G')} t(v_i) \geq 4n'_0 + 3n'_1 + 2n'_2 + n'_3 \quad (1)$$

Since G' is a subcubic triangle-free graph and M a maximum matching in G' , using a result of Munaro [16], we get

$$|V(M)| \geq \frac{9}{10}n'_3 + \frac{3}{5}n'_2 + \frac{3}{10}n'_1 \quad (2)$$

We show now that $4|V(G'')| \leq \sum_{v \in V(G')} t(v_i)$. In fact, combining $|V(G')| = n'_0 + n'_1 + n'_2 + n'_3$ with inequality (2) gives $|V(G'')| \leq n'_0 + \frac{7}{10}n'_1 + \frac{2}{5}n'_2 + \frac{1}{10}n'_3$. Thus, $4|V(G'')| \leq 4n'_0 + \frac{28}{10}n'_1 + \frac{8}{5}n'_2 + \frac{4}{10}n'_3 \leq 4n'_0 + 3n'_1 + 2n'_2 + n'_3 \leq \sum_{v \in V(G')} t(v_i)$ using inequality (1). Then

$$4|V(G'')| \leq \sum_{v \in V(G')} t(v_i) \text{ and since } |V(G)| \geq \sum_{v \in V(G')} t(v_i) \text{ we get } |V(G'')| \leq \frac{1}{4}|V(G)|.$$

Consider the partition $\mathcal{P} = D \cup T \cup M \cup V(G'')$ in the sense that \mathcal{P} contains a set for each diamond in D , one set for each triangle in T , one set for each edge in the matching M and one set for each vertex in $V(G'')$. Then $d(\mathcal{P}) = \frac{5}{4}|D| + |T| + \frac{1}{2}|M| \geq \frac{5}{4}|D| + |T| + \frac{1}{4}(n - 3|T| - 4|D| - \frac{n}{4})$ since $|V(G'')| \leq \frac{1}{4}|V(G)|$. By Lemma 10 it follows that: $\frac{opt(J)}{d(\mathcal{P})} \leq \frac{\frac{5}{4}|D| + |T| + \frac{1}{4}(n - 3|T| - 4|D| - \frac{n}{4})}{\frac{5}{4}|D| + |T| + \frac{1}{4}(n - 3|T| - 4|D| - \frac{n}{4})} = \frac{\frac{1}{4}|D| + \frac{1}{4}|T| + \frac{n}{4}}{\frac{1}{4}|D| + \frac{1}{4}|T| + \frac{3n}{16}} = 1 + \frac{n}{4|D| + 4|T| + 3n} \leq 1 + \frac{1}{3}$. ◀

5 Dense Graphs

In this section we consider graphs $G = (V, E)$ on n vertices such that G can be viewed as $G = \overline{H}$ where H is a graph of small maximum degree. Note that the edges of H are exactly the *missing edges* of G . We first consider graphs $G = (V, E)$ on n vertices such that $\delta(G) \geq n - 3$, that is $G = \overline{H}$ where H has $\Delta(H) = 2$ and has $q \leq n$ edges and show that MAX DENSE GRAPH PARTITION is solvable in polynomial time on these graphs.

► **Lemma 16.** *For any graph G on n vertices such that $\delta(G) \geq n - 3$, its density $d(G)$ is greater than or equal to the density of any partition \mathcal{P} of G into $t \geq 3$ parts.*

Proof. The density of G is given by $d(G) = \frac{n(n-1)-q}{n} = \frac{n-1}{2} - \frac{q}{n}$. From Lemma 2, among all partitions of G into $t \geq 3$ parts, those where the parts correspond to complete graphs have the largest density. The density of such a partition into t parts of size n_1, \dots, n_t is $\frac{n-t}{2}$. Thus, the density of G is at least as large as the density of this last partition since $t \geq 3$ and $q \leq n$ (note here that a graph with minimum degree $n - 3$ has at most n missing edges). ◀

Observe that in the proof of the previous lemma when $q = n$ and $t = 3$, the density of a partition in 3 parts corresponding to complete subgraphs and the density of the entire graph are the same. This previous lemma implies that for any graph G such that $\delta(G) \geq n - 3$, there exists a partition into one or two parts of maximum density.

► **Lemma 17.** *For any graph G on n vertices such that $\delta(G) \geq n - 3$, in any partition for G into two parts, the sum of missing edges in the two parts is at least o , where o is the number of odd cycles in \overline{G} .*

Proof. Let C be an odd cycle in \overline{G} . Since C is not bipartite, there is no partition $\{V_1, V_2\}$ of V such that all the edges of C have one endpoint in V_1 and one endpoint in V_2 . Hence, for any partition $\{V_1, V_2\}$ at least one of the missing edges from C is inside $G[V_1] \cup G[V_2]$. ◀

► **Lemma 18.** *Among all partitions into 2 parts of fixed size containing x missing edges, the one containing all missing edges in the largest part has the best density.*

Proof. Consider two partitions $\{V_1, V_2\}$ and $\{V'_1, V'_2\}$ such that $|V_1| = |V'_1| = n_1$ and $|V_2| = |V'_2| = n_2$ with $n_1 \leq n_2$ and $G[V_1]$ (resp. $G[V'_1]$) containing x_1 (resp. x_2) missing edges and $G[V_2]$ (resp. $G[V'_2]$) containing 0 (resp. $x = x_1 + x_2$) missing edges. Then:

$$d(\{V_1, V_2\}) = \frac{n-2}{2} - \frac{x_1}{n_1} - \frac{x_2}{n_2}, \text{ and}$$

$$d(\{V'_1, V'_2\}) = \frac{n-2}{2} - \frac{x}{n_2}.$$

Since $x = x_1 + x_2$ and $n_1 \leq n_2$, it follows that $d(\{V_1, V_2\}) \leq d(\{V'_1, V'_2\})$. ◀

► **Lemma 19.** *Among all partitions into 2 parts containing 0 (resp. x) missing edges in the smaller (resp. larger) part, the one with a maximum number of vertices in the largest part has the best density.*

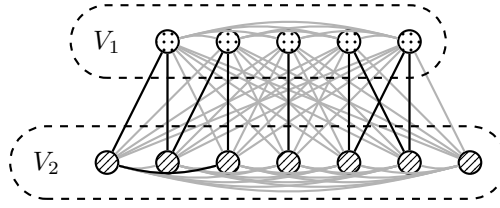
Proof. Consider two partitions $\{V_1, V_2\}$ and $\{V'_1, V'_2\}$ such that $|V_1| = n_1$, $|V_2| = n_2$ with $n_1 \leq n_2$ and $|V'_1| = n'_1$, $|V'_2| = n'_2$ with $n'_1 \leq n'_2$ and $G[V_1]$ (resp. $G[V'_1]$) containing 0 (resp. x) missing edges and $G[V_2]$ (resp. $G[V'_2]$) containing 0 (resp. x) missing edges. Moreover suppose $n_2 \leq n'_2$. The densities for these partitions are:

$$d(\{V_1, V_2\}) = \frac{n-2}{2} - \frac{x}{n_2}, \text{ and}$$

$$d(\{V'_1, V'_2\}) = \frac{n-2}{2} - \frac{x}{n'_2}.$$

Since $n_2 \leq n'_2$, it follows that $d(\{V_1, V_2\}) \leq d(\{V'_1, V'_2\})$. ◀

13:12 Dense Graph Partitioning on Sparse and Dense Graphs



■ **Figure 5** Construction of V_1 and V_2 in Theorem 20.

► **Theorem 20.** *MAX DENSE GRAPH PARTITION is solvable in polynomial time on graphs G with n vertices with $\delta(G) \geq n - 3$.*

Proof. Let G be a graph of minimum degree $n - 3$. We first define a partition $\{V_1, V_2\}$ of the vertices of G by giving vertices color 1 or 2, in the sense that V_1 (resp. V_2) contains vertices of color 1 (resp. 2). An example is given in Figure 5. We assign color 2 to each vertex of degree $n - 1$. Since the minimum degree in G is $n - 3$, the graph H of missing edges is a collection of paths and cycles. We color the vertices on paths or cycles with an even number of vertices alternating by 1 and 2. For vertices on paths or cycles with an odd number of vertices we also color them alternating by 1 and 2, always starting with color 2. Thus cycles of odd size have two adjacent vertices of color 2.

Let o be the number of odd cycles in H . The partition $\{V_1, V_2\}$ defined by our 2-coloring contains o missing edges in V_2 and $|V_2|$ is maximized among all such partitions. Its density is equal to $\frac{n-2}{2} - \frac{o}{n_2}$, where $n_2 = |V_2|$. Denote by d_{n-1} the number of vertices of G of degree $n - 1$ and by p_o the number of paths with an odd number of vertices (even length) among the missing edges. The sets V_1 and V_2 contain the same number of vertices of degree $n - 2$ that are extremities of a path with an even number of vertices in H . The set V_2 contains p_o more vertices of degree $n - 2$, that are extremities of a path with an odd number of vertices, than V_1 . The set V_2 contains o more vertices of degree $n - 3$ than V_1 . Thus $n_1 = \frac{1}{2}(n - d_{n-1} - p_o - o)$ and $n_2 = \frac{1}{2}(n + d_{n-1} + p_o + o)$. We claim that there is no partition into two parts that has a higher density.

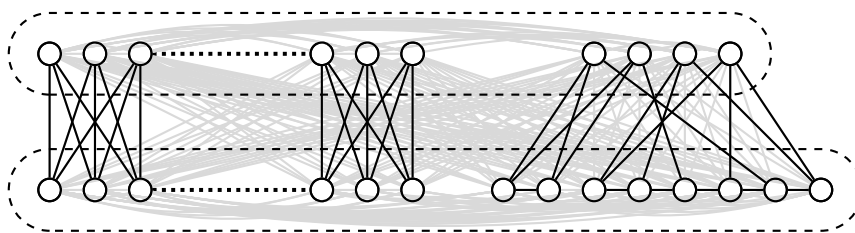
By Lemma 17, any partition into two sets contains at least o missing edges inside the two parts. By construction we have maximized the number of vertices in the part with the missing edges among all partitions with the minimum number o of missing edges, i.e., there is no partition into two parts $\{V'_1, V'_2\}$ with o missing edges all contained in V'_2 and $|V'_2| > |V_2|$. Hence, by Lemmas 18 and 19, it remains to show that any partition $\{V'_1, V'_2\}$ with $o + x > o$ missing edges for some $x > 0$ has a smaller density than $\{V_1, V_2\}$.

Let $\{V'_1, V'_2\}$ be a partition with $o + x > o$ missing edges for some $x > 0$ and assume w.l.o.g. that $|V'_1| \leq |V'_2|$. By definition of the partition $\{V_1, V_2\}$, it follows that $|E(H)| = 2n_1 - p_o + o$ (note that all non-edges have to either be among the o missing edges in the partition or in the cut between V_1 and V_2). In the partition $\{V'_1, V'_2\}$, it follows that $|E(H)| \leq 2|V'_1| - r_1 + (o + x)$, where r_1 is the number of vertices in V'_1 adjacent to only one edge in H . In the cut between V'_1 and V'_2 , each vertex in V'_1 is adjacent to at most two such edges. Combining these two bounds on $|E(H)|$ yields

$$2n_1 - p_o \leq 2|V'_1| - r_1 + x. \quad (3)$$

We claim that $r_1 \geq p_o - x$. To see this, observe that every path of odd length either results in a vertex in V'_1 adjacent to only one edge in $E(H)$ (r_1) or in a missing edge. Also, every cycle of odd length creates at least one missing edge. Thus the number of missing edges $o + x$ for $\{V'_1, V'_2\}$ is at least $p_o - r_1 + o$. Reordering this yields the claimed

$$r_1 \geq p_o - x. \quad (4)$$



■ **Figure 6** The construction of G' in Definition 21.

Inequalities (3) and (4) yield $2n_1 - p_o \leq 2|V'_1| - p_o + 2x$ and thus $n_1 - |V'_1| \leq x$. Since $\{V'_1, V'_2\}$ is a partition it follows that $|V'_2| = n - |V'_1| \leq n - n_1 + x = n_2 + x$.

By Lemmas 18 and 19, the best case of missing edges for $\{V'_1, V'_2\}$ is that they all are in the larger part V'_2 , hence the density of $\{V'_1, V'_2\}$ is at most $\frac{n-2}{2} - \frac{o+x}{|V'_2|}$. With $|V'_2| \leq n_2 + x$, we can bound $d(V'_1, V'_2) \leq \frac{n-2}{2} - \frac{o+x}{n_2+x}$. Since H is of degree at most 2, we know that there cannot be more missing edges than vertices in a part, thus in particular $o \leq n_2$. This last observation allows to bound $d(V'_1, V'_2) \leq \frac{n-2}{2} - \frac{o+x}{n_2+x} \leq \frac{n-2}{2} - \frac{o}{n_2} = d(V_1, V_2)$, thus the density of $\{V'_1, V'_2\}$ is not larger than the density of $\{V_1, V_2\}$. ◀

We now consider graphs $G = (V, E)$ on n vertices that are $(n - 4)$ -regular, that is $G = \overline{H}$ where H is a cubic graph. We show that DENSE GRAPH PARTITION is NP-hard on $(n - 4)$ -regular graphs, by showing a reduction from MIN UNCUT on cubic graphs, that is the complement of MAX CUT. This last problem on cubic graphs was proved NP-hard and even not polynomial-time 1.003-approximable, unless $P=NP$ [3].

MIN UNCUT

Input: A graph $G = (V, E)$, an integer k .

Question: Does G contain a partition of V into two parts A, B such that the number of edges with both endpoints in the same part is at most k ?

► **Definition 21.** Let $I = (G, k)$ be an instance of MIN UNCUT where $G = (V, E)$ is a cubic graph. We define the construction σ transforming the graph G into the graph $G' := (V', E') = \sigma(G)$ (see Figure 6) as follows:

- let $G_0 = (V_0, E_0)$ be the union of $\frac{n^2-n}{6}$ copies of $K_{3,3}$ (see remark below). Thus G_0 is a cubic bipartite graph with $n^2 - n$ vertices and V_0 is the union of two independent sets L, R such that $|L| = |R|$.
- let $G_1 = (V \cup V_0, E \cup E_0)$.
- let $G' = \overline{G_1}$.

► **Remark 22.** Note that we can assume that the number of vertices of a cubic graph G is a multiple of 6. Since G is cubic, n is a multiple of 2. If n is not a multiple of 3, we consider the instance I_{triple} defined as follows: G_{triple} is the union of 3 copies of G and $k_{triple} = 3k$, and thus in the new instance I_{triple} the graph has $3n$ vertices. Note that the number of edges with both endpoints in the same part is $3k$ in G_{triple} if and only if it is k in G .

Let $n = |V|$, $m = |E|$, $n' = |V'|$ and $m' = |E'|$. Observe that $n' = n^2$, and G' is a $(n' - 4)$ -regular graph. With this construction one can show the following.

► **Theorem 23.** DENSE GRAPH PARTITION is NP-complete on $(n - 4)$ -regular graphs with n vertices.

13:14 Dense Graph Partitioning on Sparse and Dense Graphs

At the end of this section we show that a partition into a bounded number of cliques provides a good approximation for graphs of large minimum degree.

► **Theorem 24.** *DENSE GRAPH PARTITION is polynomial-time $\frac{n-1}{\delta(G)+1}$ -approximable on graphs G with n vertices.*

Proof. Let G be a graph on n vertices with minimum degree $\delta = \delta(G)$, instance of MAX DENSE GRAPH PARTITION. If $\delta \geq n - 3$, we can give an optimum solution in polynomial time by Theorem 20. So assume $\delta \leq n - 4$. By Lemma 3, any partition \mathcal{P} for the vertices of G satisfies $d(\mathcal{P}) \leq \frac{n-1}{2}$. Using Brooks' theorem [5], \overline{G} is $(n - \delta - 1)$ -colorable, and further, such a coloring can be computed in polynomial time. (Note that $\delta \leq n - 4$ implies that \overline{G} is not a complete graph or a circle, the two exceptions in Brooks' theorem where one more color is needed.) Using such a coloring, G can be partitioned into $n - \delta - 1$ cliques. Then the density of this partition is $\frac{n-(n-\delta-1)}{2} = \frac{\delta+1}{2}$. Comparing this value with the upper bound of $\frac{n-1}{2}$ on the optimum shows that this partition into $n - \delta - 1$ cliques gives a polynomial-time $\frac{n-1}{\delta+1}$ -approximation for DENSE GRAPH PARTITION. ◀

Notice that if $\delta(G) > \frac{n-3}{2}$, the ratio given in Theorem 24 improves upon the current best ratio of 2 for DENSE GRAPH PARTITION on general graphs. This approximation can further be used to show the following.

► **Theorem 25.** *There is an efficient polynomial-time approximation scheme for MAX DENSE GRAPH PARTITION on graphs G with n vertices and $\delta(G) = n - t$ for a constant $t \geq 4$.*

Proof. Let $I = G$ be a graph on n vertices and $\delta(G) = n - t$, instance of MAX DENSE GRAPH PARTITION. We establish in the following an ϵ -approximation. Given $\epsilon > 0$, consider two cases.

If $n \geq t - 1 + \frac{t-2}{\epsilon}$, then let \mathcal{P} be a partition that corresponds to a $(t - 1)$ -coloring of \overline{G} such that each part is a clique in G as in the proof of Theorem 24. Then $d(\mathcal{P}) = \frac{n-t+1}{2} \geq \frac{n+1 - \frac{n\epsilon + \epsilon + 2}{1+\epsilon}}{2} \geq \frac{n-1}{2(1+\epsilon)} \geq \frac{\text{opt}(I)}{1+\epsilon}$, where the last inequality $\text{opt}(I) \leq \frac{n-1}{2}$ comes from Lemma 3.

Otherwise, that is $n < t - 1 + \frac{t-2}{\epsilon}$, enumerate all the partitions of G and consider the best one. Since the number of partitions of G is the Bell number of order $|V| = n$, B_n , and $B_n \leq n^n$, we get an optimal solution in time $(1/\epsilon)^{O(1/\epsilon)}$. ◀

6 Conclusion

In order to have a better understanding of the complexity of MAX DENSE GRAPH PARTITION it would be nice to study it on other graph classes. It was proved to be polynomial-time solvable on trees, but the complexity on graphs of bounded treewidth remains open. Moreover no result exists on split graphs. Concerning approximation, no lower bound was established, it would be nice to improve the 2-approximation algorithm or to show that no polynomial-time approximation scheme exist on general instances.

References

- 1 Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. *Journal of Computer and System Sciences*, 58(1):193–210, 1999.
- 2 Haris Aziz, Serge Gaspers, Joachim Gudmundsson, Julián Mestre, and Hanjo Taubig. Welfare maximization in fractional hedonic games. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 468–474. AAAI Press, 2015.

- 3 Piotr Berman and Marek Karpinski. On some tighter inapproximability results (extended abstract). In Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Proceedings of the 26th International Colloquium on Automata, Languages and Programming, ICALP 1999*, volume 1644 of *LNCS*, pages 200–209. Springer, 1999.
- 4 Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k -subgraph. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 201–210. ACM, 2010.
- 5 Rowland Leonard Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37:2, pages 194–197. Cambridge University Press, 1941.
- 6 Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer, 2016.
- 7 Derek G. Corneil and Yehoshua Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1):27–39, 1984.
- 8 Julien Darlay, Nadia Brauner, and Julien Moncel. Dense and sparse graph partition. *Discrete Applied Mathematics*, 160(16-17):2389–2396, 2012.
- 9 Uriel Feige, Guy Kortsarz, and David Peleg. The dense k -subgraph problem. *Algorithmica*, 29(3):410–421, 2001.
- 10 Santo Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.
- 11 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.
- 12 Andrew V. Goldberg. *Finding a maximum density subgraph*. University of California Berkeley, 1984.
- 13 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 14 Subhash Khot. Ruling out PTAS for graph min-bisection, dense k -subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, 2006.
- 15 Samir Khuller and Barna Saha. On finding dense subgraphs. In *Proceedings of 36th International Colloquium on Automata, Languages and Programming, ICALP 2009, Part I*, volume 5555 of *LNCS*, pages 597–608, 2009.
- 16 Andrea Munaro. On line graphs of subcubic triangle-free graphs. *Discrete Mathematics*, 340(6):1210–1226, 2017.
- 17 Mark E. J. Newman. Detecting community structure in networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 38(2):321–330, 2004.
- 18 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

The Diameter of Caterpillar Associahedra

Benjamin Aram Berendsohn  

Institut für Informatik, Freie Universität Berlin, Germany

Abstract

The caterpillar associahedron $\mathcal{A}(G)$ is a polytope arising from the rotation graph of search trees on a caterpillar tree G , generalizing the rotation graph of binary search trees (BSTs) and thus the conventional associahedron. We show that the diameter of $\mathcal{A}(G)$ is $\Theta(n + m \cdot (H + 1))$, where n is the number of vertices, m is the number of leaves, and H is the entropy of the leaf distribution of G .

Our proofs reveal a strong connection between caterpillar associahedra and searching in BSTs. We prove the lower bound using Wilber's first lower bound for dynamic BSTs, and the upper bound by reducing the problem to searching in static BSTs.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Theory of computation → Data structures design and analysis

Keywords and phrases Graph Associahedra, Binary Search Trees, Elimination Trees

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.14

Funding Work supported by DFG grant KO 6140/1-1.

Acknowledgements I would like to thank László Kozma for helpful discussions and suggestions.

1 Introduction

Associahedra are a family of polytopes with interesting combinatorial properties. In particular, the skeleton of the $(n - 1)$ -dimensional associahedron \mathcal{A}_n represents the *rotation graph* of binary search trees (BSTs) on n keys. More precisely, each vertex of \mathcal{A}_n represents a BST, and each edge represents a rotation (definitions of BSTs and rotations can be found in standard textbooks). The diameter of \mathcal{A}_n is known to be precisely $2n - 6$ when $n > 10$ [21, 19]; that is, every BST can be transformed into any other BST with at most $2n - 6$ rotations, and this is tight.

In this paper, we study a generalization of associahedra called *graph associahedra*. Graph associahedra were originally defined using *tubings* [6]. We use an equivalent definition based on *search trees on graphs* (STGs). While the keyspace of a BST is a linearly ordered set, the key space of an STG is a graph. Formally, given a connected graph $G = (V, E)$, a *search tree on G* is a rooted tree T that can be constructed as follows. Choose a vertex $r \in V$ as the root. Then, recursively create search trees on the connected components of $G \setminus v$, and add them to T as children of r . Rotations on STGs can be defined similarly as for BSTs (more details in Section 2). Search trees on graphs have been used in various contexts under different names (see, e.g., [3, Section 2.2]).

Given a connected graph G on n vertices, Carr and Devadoss [6] defined the graph associahedron $\mathcal{A}(G)$ as an $(n - 1)$ -dimensional polytope such that the skeleton of $\mathcal{A}(G)$ is isomorphic to the rotation graph of the search trees on G . Since search trees on the path with n vertices correspond to BSTs on n nodes, we obtain the conventional associahedron when G is a path.

For search trees T_1 and T_2 on a graph G , let the *rotation distance* $d(T_1, T_2)$ be the minimum number of rotations required to transform T_1 into T_2 . The diameter $\delta(\mathcal{A}(G))$ of $\mathcal{A}(G)$ is the maximum rotation distance between two search trees on G . Manneville and Pilaud [15] showed that the diameter of graph associahedra is monotone under the addition of edges, and $\max\{2n - 18, m\} \leq \delta(\mathcal{A}(G)) \leq \binom{n}{2}$ holds for each connected graph G on n



© Benjamin Aram Berendsohn;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 14; pp. 14:1–14:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

vertices and m edges. Both the upper and the lower bound are asymptotically tight: For example, conventional associahedra (G is a path) and cyclohedra (G is a cycle) have linear diameter, and permutohedra (G is a complete graph) have diameter $\binom{n}{2}$.

In this paper, we consider the case where G is a *caterpillar tree*. A caterpillar tree (or simply *caterpillar*) is a tree consisting of a path and some number of leaves that are adjacent to the path. The choice of that path is not unique for non-trivial caterpillars (it is always possible to either add or remove a leaf at the end of the path), but we assume that any considered caterpillar consists of a distinguished path called the *spine* and any number of leaves, called *legs*. Our asymptotic results are not affected by the choice of the spine.

We determine the diameter of every caterpillar associahedron up to a constant factor. This involves the Shannon entropy of the “leg distribution”, which we now properly define. Let G be a caterpillar tree with n spine vertices s_1, s_2, \dots, s_n , let s_i be adjacent to m_i leg vertices, and let $m = m_1 + m_2 + \dots + m_n$ be the total number of leg vertices. Then

$$H(G) = H(m_1, m_2, \dots, m_n) = \sum_{i \in [n], m_i > 0} \frac{m_i}{m} \log \left(\frac{m}{m_i} \right).$$

For simplicity of presentation, we write $H'(\cdot) = H(\cdot) + 1$. We are now ready to state our main result.

► **Theorem 1.1.** *Let G be a caterpillar tree with n spine vertices and m leg vertices. Then $\delta(\mathcal{A}(G)) \in \Theta(n + m \cdot H'(G))$.*

Notably, if $m = n$ and each spine node is adjacent to one leaf node, then $\delta(\mathcal{A}(G)) \in \Omega(n \log n)$.

Our proofs make use of techniques from the design of *optimal BSTs*. A connection between rotations in BSTs and rotations in search trees on caterpillars is not surprising – caterpillars are similar to paths, after all. However, we show a connection to *queries* to BSTs. Essentially, the leg nodes in search trees on caterpillars can be seen as queries to the BST on the spine nodes. For our upper bound (Section 4), we use the fact that an optimal *static* BST for an input distribution X has amortized query cost $H'(X)$ [16]. For our lower bound (Section 5), we use *Wilber’s first lower bound* [22], which bounds the performance of *dynamic* BSTs on a certain input sequence. We show that it also bounds the rotation distance between certain search trees on a caterpillar. Finally, we show that Wilber’s first lower bound is asymptotically equal to $H'(X)$ if the input distribution X is fixed, but the order of queries is worst possible. Note that this implies that dynamic BSTs cannot beat optimal static BST on any distribution if the ordering is worst possible. Kujala and Elomaa [12] previously showed that this is true even if the ordering is random, but they did not use Wilber’s bound.

1.1 Related work

Improved bounds on $\delta(\mathcal{A}(G))$ are known if G belongs to certain graph classes. Pournin [18] showed that $\delta(\mathcal{A}(G)) \approx 2.5n$ if G is the cycle on n vertices. Manneville and Pilaud [15] noted that $\delta(\mathcal{A}(G)) \in \mathcal{O}(n \log n)$ if G is a tree on n vertices, and Cardinal, Langerman and Pérez-Lantero [3] showed this bound is tight if G has the form of a balanced binary tree.

Recently, Cardinal, Pournin, and Valencia-Pabon [4, 5] showed that $\delta(\mathcal{A}(G)) \in \mathcal{O}(\text{td}(G) \cdot n)$, where $\text{td}(G)$ is the *treedepth*¹ of G , and that this bound is attained by *trivially perfect graphs*. Using the relationship between treedepth and *treewidth*, this extends the $\mathcal{O}(n \log n)$

¹ The treedepth $\text{td}(G)$ can be defined as the minimum height of a search tree on G .

upper bound to graphs with bounded treewidth. They also showed that this bound is tight for graphs of *pathwidth two* (which have treewidth at most two, but are not necessarily trees). For the definitions of treewidth and pathwidth, we refer to [4]. Our Theorem 1.1 shows that the $\mathcal{O}(n \log n)$ bound is tight already for caterpillars, which are both trees and have pathwidth *one* (in fact, caterpillars are precisely the graphs of pathwidth one).

We do not consider *queries* to STGs in this paper. Some results from BSTs have been shown to hold in the more general case where G is a tree. Bose, Cardinal, Iacono, Koumoutsos, and Langerman [2] presented an $\mathcal{O}(\log \log n)$ -competitive search tree algorithm based on *tango trees* for BSTs [10]. Berendsohn and Kozma [1] described a variant of Splay trees [20], and a polynomial-time approximation scheme for the optimal static search tree on a given tree for a given input distribution. Notably, it is still unknown whether an optimal static search tree on a tree can be found in polynomial time.

Berendsohn and Kozma [1] also showed that if we only consider a subset of search trees on a tree G called *k-cut trees*, then the maximum rotation distance between two STGs is linear. A special case of *k-cut trees* are *Steiner-closed trees*, which play a central role in the results of [2] and [1].

2 Preliminaries

In this paper, we consider (simple and undirected) graphs on the one hand, and (rooted) search trees on the other. We call the vertices of search trees *nodes*. In both cases, we denote by $V(\cdot)$ the set of vertices or nodes and by $E(\cdot)$ the set of edges.

Let G be a graph. We denote the subgraph of G induced by $U \subseteq V(G)$ by $G[U]$. For $v \in V(G)$, we write $G \setminus v = G[V(G) \setminus \{v\}]$.

Let T be a rooted tree and $x \in V(T)$. For a node x , T_x denotes the subtree of T consisting of x and all its descendants. The *depth* of x is the number of nodes in the path from the root of T to x , and is denoted by $\text{depth}_T(x)$.

2.1 Queries in binary search trees

In the *dynamic BST model*, we are given a starting BST S on $[n]$ and a sequence σ of access queries. Each access query specifies a node $i \in [n]$. We start each query with a pointer at the root, and are required to move the pointer to the node i to satisfy the query. To this end, we are allowed to move the pointer to the parent or a child of the node it is currently pointing at, or execute a rotation involving that node. Let $\text{OPT}(S, \sigma)$ denote the minimum number of pointer moves and rotations needed to serve σ . We charge a pointer move at the start of each query, when the pointer is moved to the root, so each query has cost at least one.

Since the rotation distance between two BSTs is $\mathcal{O}(n)$, we can always replace the starting BST S by a different one at the cost of $\mathcal{O}(n)$. If the access sequence is long enough, this cost is insignificant; therefore, let us define $\text{OPT}(\sigma) = \min_S \text{OPT}(S, \sigma)$. For each BST S , we have $\text{OPT}(S, \sigma) \leq \text{OPT}(\sigma) + \mathcal{O}(n)$.

It is not known how to compute or approximate $\text{OPT}(S, \sigma)$ or the associated sequence of operations efficiently. However, a number of algorithms have been conjectured to be *instance-optimal*, i.e., to serve every access sequence σ with a cost of $\mathcal{O}(\text{OPT}(\sigma) + n)$, most notably *Splay* [20] and *Greedy* [14, 17]. We emphasize that Splay is an *online* algorithm, i.e., it serves each query independently from future queries, and that Greedy can be made online [9] with only a constant-factor overhead. It is currently unknown whether any online algorithm can approximate the offline optimum $\text{OPT}(\sigma)$ by a constant factor; this is the subject of the *dynamic optimality conjecture*.

There are several lower bounds known for $\text{OPT}(\sigma)$. In this paper, we use *Wilber's first lower bound* [22], which we define and discuss in Section 5.1.

If we do not allow rotations, then the best strategy is to simply move the pointer down until we hit the queried node. Thus, the minimum cost of serving $\sigma = (x_1, x_2, \dots, x_m)$ in S is $\sum_{i=1}^m \text{depth}_S(x_i)$. A BST S minimizing this quantity is called an *optimal static BST*. Note that only the frequencies of the elements in σ affect the static cost, not the order. For a BST S on $[n]$ and element frequencies $m_1, m_2, \dots, m_n \in \mathbb{N}_0$, define

$$\text{cost}(S, m_1, m_2, \dots, m_n) = \sum_{i=1}^n m_i \cdot \text{depth}_S(i).$$

Let $\text{OPT-ST}(m_1, m_2, \dots, m_n)$ be the minimum $\text{cost}(S, m_1, m_2, \dots, m_n)$ over all possible BSTs S .

It is possible to compute an optimal static BST in $\mathcal{O}(n^2)$ time [11]. Mehlhorn showed that $\frac{1}{m} \text{OPT-ST}(m_1, m_2, \dots, m_n)$ is close to the entropy of the input distribution.

► **Lemma 2.1** (Mehlhorn [16]). *Let $X = (m_1, m_2, \dots, m_n)$ be a sequence of nonnegative integers, and let $m = m_1 + m_2 + \dots + m_n$. Then*

$$\frac{1}{2} H(X) \cdot m \leq \text{OPT-ST}(X) \leq (2H(X) + 2) \cdot m = 2H'(X) \cdot m.$$

2.2 Search trees on graphs

Let G be a connected graph, and T be a rooted tree, such that $V(T) = V(G)$. Let r be root of T and let c_1, c_2, \dots, c_k be the children of r in T . Then T is a *search tree on G* if

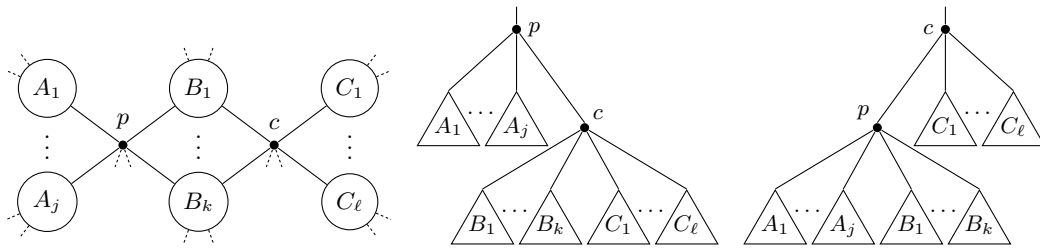
- (a) $G \setminus r$ consists of precisely k connected components C_1, C_2, \dots, C_k such that $V(T_{c_i}) = V(C_i)$ for each $i \in [k]$; and
- (b) T_{c_i} is a search tree on C_i for each $i \in [k]$.

Let T be a search tree on G , let $p \in V(G)$, let c be a child of p in T , and let g be the parent of p in T , if p is not the root. A *rotation* of the edge (p, c) makes c the parent of p and child of g (or root), and accordingly redistributes children so that the result is still a search tree on G . More precisely, it (1) makes c a child of g , if p is not the root, and otherwise, makes c the root; (2) makes p a child of c ; and (3) makes each child x of c a child of p where $V(T_c)$ contains both a vertex adjacent to p and a vertex adjacent to c . See Figure 1 for an illustration. It can be checked that the rooted tree resulting from a rotation is indeed a search tree on G . It is also easy to see that each search tree on G can be rotated into every other search tree on G , e.g., by rotating the correct element to the root and then recursing on the subtrees.

2.3 Projections of STGs

Both of our proofs make use of a concept defined by Cardinal, Langerman, and Pérez-Lantero [3]. Let T be a search tree on a graph G , and let v be a leaf vertex in G . Note that v has at most one child in T . We define $T \setminus v$ to be the following search tree on $G \setminus v$: If v has no children, simply remove v . If v has a parent p and a child c , remove v and make c a child of p . If v is the root of T and has a child c , then remove v and make c the root. We call this operation *pruning v* .

If G is a tree, then we can obtain every subgraph of G by progressively removing leaves. Accordingly, if T is a search tree on a tree G , and $U \subseteq V(G)$ is a set of vertices such that $G[U]$ is connected, then we can define the *projection* of T onto U , written $T[U]$, as the search



■ **Figure 1** A rotation in a search tree on G . (left) A subgraph G' of G with two vertices p and c that split G' into $j + k + \ell$ components. Each solid line represents one or more edges. Dashed lines indicate possible edges to the rest of G . (center) A subtree T_p in a search tree T on G , with $V(T_p) = V(G')$. (right) The result of the rotation (p, c) in T .

tree on $G[U]$ obtained by progressively pruning the vertices in $V(G) \setminus U$. It is easy to see that the order of pruning does not matter.

The main utility of projections lies in the following lemma, which essentially states that projections onto U are only affected by rotations between nodes in U .

► **Lemma 2.2** (Cardinal et al. [3]). *Let T be a search tree on a tree G , let $U \subseteq V(G)$ such that $G[U]$ is connected, let (x, y) be an edge of T , and let T' be the tree obtained by rotating (x, y) . If $x, y \in U$, then $T'[U]$ is the STG obtained when rotating (x, y) in $T[U]$. Otherwise, $T'[U] = T[U]$.*

3 Search trees on caterpillars

Let $n \in \mathbb{N}_+$ and $m_1, m_2, \dots, m_n \in \mathbb{N}_0$, and write $m = \sum_{i=1}^n m_i$. We define the caterpillar $C(m_1, m_2, \dots, m_n)$ with n spine vertices and m leg vertices as follows. The spine of G consists of the vertices s_1, s_2, \dots, s_n , in that order. Additionally, for each $i \in [n]$ and $j \in [m_i]$, there is a leg vertex $\ell_{i,j}$ that is adjacent to s_i . Clearly, every caterpillar can be constructed this way.

Let T be a search tree on $G = C(m_1, m_2, \dots, m_n)$, and let $x \in V(T)$. We call x a *leg node* if it corresponds to a leg vertex, and a *spine node* if it corresponds to a spine vertex. We denote nodes in T in the same way we denote vertices in G , i.e., we write $\ell_{i,j}$ for leg nodes and s_i for spine nodes.

We call a leg node *bound* if it has no children, and *free* otherwise.

► **Observation 3.1.** *Let T be a search tree on $C(m_1, m_2, \dots, m_n)$. Consider a leg node $\ell_{i,j}$. If $\ell_{i,j}$ has no children, then s_i is its parent. Otherwise, $\ell_{i,j}$ has exactly one child, and s_i is a descendant of $\ell_{i,j}$.*

Define $\text{bst}(T)$, the *spine BST*, as the projection of T onto the spine vertices of G (see Figure 2 for an example). Since the spine vertices form a path, $\text{bst}(T)$ indeed corresponds to a binary search tree. By Lemma 2.2, each rotation between two spine nodes in T corresponds to a BST rotation in $\text{bst}(T)$. However, the converse is not true in general, since two nodes u, v that are parent and child in $\text{bst}(T)$ might have leg nodes between them in T , in which case a rotation of u, v in $\text{bst}(T)$ cannot be applied to T . Call an edge (u, v) of $\text{bst}(T)$ *light* if (u, v) is also an edge in T . Essentially, as long as we restrict ourselves to light edges, we can apply BST restructuring algorithms to T . This will be useful to prove our upper bound. We further observe that rotations between spine nodes preserve the parents of leg nodes.

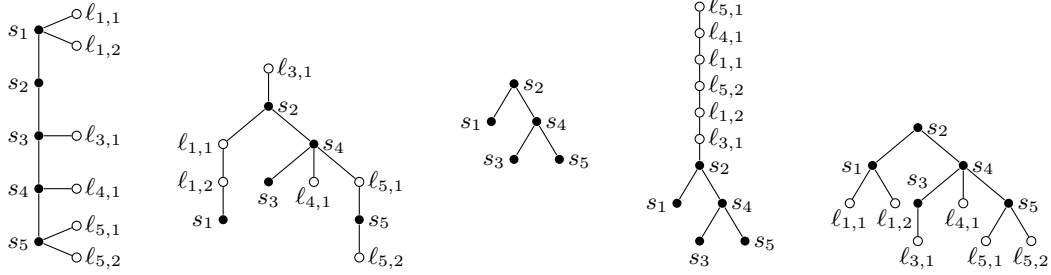


Figure 2 (left) The caterpillar $G = C(2, 0, 1, 1, 2)$. (center left) A search tree T on G . (center) The spine BST $S = \text{bst}(T)$. (center right) The STG $A(S, \pi)$ with $\pi = (\ell_{3,1}, \ell_{1,2}, \ell_{5,2}, \ell_{1,1}, \ell_{4,1}, \ell_{5,1})$. (right) The STG $B(S)$.

► **Observation 3.2.** Let T be a search tree on a caterpillar, let T' arise from a rotation between spine nodes in T , and let ℓ be a leg node. If ℓ is the root of T , then ℓ is also the root of T' . Otherwise, the parent of ℓ in T' is equal to the parent of ℓ in T .

3.1 Special STGs

Before proceeding with the proofs, we define two useful kinds of STGs where the spine BST has only light edges. Let $G = C(m_1, m_2, \dots, m_n)$ be a caterpillar, let S be a BST on the spine nodes of G , and let π be an ordering of the leg nodes of G . Define $A(S, \pi)$ to be the unique search tree on G such that $\text{bst}(A(S, \pi)) = S$, each leg node is above each spine node (i.e., the leg nodes form a path at the top of the tree), and the order of the leg nodes from bottom to top is π . Define $B(S)$ to be the unique search tree on G such that all leg nodes are bound and $\text{bst}(B(S)) = S$. Clearly, every search tree T on G without free leg nodes is equal to $B(\text{bst}(T))$. Figure 2 shows examples.

4 Upper bound

Fix a caterpillar $G = C(m_1, m_2, \dots, m_n)$. We first consider only STGs without free leg nodes.

► **Lemma 4.1.** Let T_1, T_2 be search trees on G without free leg nodes. Then, T_1 can be transformed into T_2 with $\mathcal{O}(n)$ rotations.

Proof. Let $S_1 = \text{bst}(T_1)$ and $S_2 = \text{bst}(T_2)$. As stated in the introduction, there is a sequence of at most $\mathcal{O}(n)$ rotations that transforms S_1 into S_2 . We simply apply these rotations to T_1 . For this to be well-defined, we need to show that we never (attempt to) rotate a heavy edge. Since T_1 has no free leg nodes, all edges in S_1 are light, so the first rotation goes through. Furthermore, a rotation between two spine nodes can never change a leg node from bound to free (or vice versa). Thus, by induction, after each rotation, all leg nodes are still bound, so we can apply the next rotation. ◀

The above lemma provides us with a “core” of the caterpillar associahedron with linear diameter. In the following, we show that the rotation distance from any search tree to *some* STG without free leg nodes is $\mathcal{O}(n + H'(G) \cdot m)$. By the triangle inequality, this means that the rotation distance between any two STGs is at most $2 \cdot \mathcal{O}(n + H'(G) \cdot m) + \mathcal{O}(n) = \mathcal{O}(n + H'(G) \cdot m)$, and thus we have the upper bound of Theorem 1.1.

We first show how to reduce the problem to the case that $T = A(S, \pi)$ for some BST S and some leg node ordering π . For this, the following lemma is useful.

- **Lemma 4.2.** *Let T be a BST. There exists a sequence of $\mathcal{O}(n)$ rotations on T such that*
- (i) *every rotation involves only nodes at depth at most 3; and*
 - (ii) *every node becomes the root of $\text{bst}(T)$ at some point.*

Lemma 4.2 can be easily derived from a result by Cleary [8], which uses algebraic techniques. We provide an elementary proof for completeness.

Proof of Lemma 4.2. We proceed in two phases. In the first phase, we repeatedly rotate the root of T with its left child, until the root has no left child. This requires $\mathcal{O}(n)$ rotations.

In the second phase, we repeat the following step as long as the root has a right child u . If u has a left child v , then rotate u with v . Otherwise, rotate the root with u .

We now bound the number of rotations in the second phase. Let $L(T)$ be the *left path* of a BST T , i.e., the maximal sequence v_1, v_2, \dots where v_1 is the root and v_{i+1} is the left child of v_i . Let the *right path* $R(T)$ be defined similarly. Observe that each step in the second phase increases the quantity $2|L(T)| + |R(T)|$ by one. Since $2|L(T)| + |R(T)| \leq 2n + 1$ for every BST T , the total number of rotations is linear.

The rotations only involve the root, its children, and its grandchildren, so (i) holds. To show (ii), suppose a node v never becomes the root. At the start of the second phase, v is in the right subtree of the root, and at the end, v is in the left subtree of the root. Thus, v must pass from the right subtree to the left subtree of the root at some point. The only way this can happen without v becoming the root is that v is in the left subtree of the right child of the root, and we rotate at the root. But in the second phase we only rotate at the root when the right child of the root has no left child, a contradiction. Thus, (ii) holds. ◀

- **Lemma 4.3.** *Let T be an arbitrary search tree on G . Then T can be transformed into some $A(S, \pi)$ using $2m + \mathcal{O}(n)$ rotations.*

Proof. An STG has the form $A(S, \pi)$ if and only if each leg node has no spine ancestor. The basic idea of the proof is to apply rotations between spine nodes to eventually bring each spine node to the root (of the spine BST). At any point, if the current root of the spine BST has a leg node as a child, rotate it with the leg node. We refer to all such rotations between two spine node rotations as the `cleanup` step. By Observation 3.2, every leg node is transported to the top this way.

The problem with this approach is that we can only rotate light edges in the spine BST, and the only edges that we know must be light are the edges between the BST root and its children. However, if we extend our `cleanup` step to consider leg nodes that are somewhat deeper in the tree (that is, nodes with two spine ancestors instead of just one), we guarantee that all BST edges *near* the BST root are light. This allows us to apply Lemma 4.2 to bring each spine node to the root. We now describe the sequence of rotations more formally, starting with the `cleanup` step.

Let T' be the current STG. Let `cleanup`(T') be the following sequence of rotations: As long as there is a leg node ℓ with a spine parent p and at most two spine ancestors, rotate (p, ℓ) . Arbitrarily resolve conflicts. Let T'' be the STG after applying `cleanup`. Clearly, no spine node s with $\text{depth}_{\text{bst}(T'')}(s) \leq 2$ has a leg node child in T'' , so all edges in $\text{bst}(T'')$ involving the root or its children are light. Moreover, each leg node that is touched by `cleanup` is rotated at most twice, and afterwards has no spine node ancestors.

Let \mathcal{X} be the sequence of rotations obtained by applying Lemma 4.2 to $\text{bst}(T)$. We first apply `cleanup` to T , then apply the spine rotations in \mathcal{X} , with a `cleanup` step after each spine rotation. Since each rotation in \mathcal{X} involves either the root of the spine BST or one of its children, the rotation is applied to a light edge. Thus, the whole sequence can indeed be applied to T .

The number of rotations is at most $2m + \mathcal{O}(n)$. Indeed, since no rotation between spine nodes can change the parent of a leg node (see Observation 3.2), each leg node is only touched in a single `cleanup` step (where it is rotated above all spine nodes), and only twice in that `cleanup` step. The length of \mathcal{X} is $\mathcal{O}(n)$ by Lemma 4.2.

Finally, we show that the final tree is indeed of the form $A(S, \pi)$. For this, it suffices to show that each leg node that has at least one spine ancestor in T is touched in some `cleanup` step. Suppose that such a leg node ℓ is not involved in a `cleanup` step. Without loss of generality, let the parent p of ℓ be a spine node. By Observation 3.2, since ℓ is not touched in a `cleanup` step and we never rotate between leg nodes, p stays the parent of ℓ throughout the sequence of rotations. However, then p will be the root of $\text{bst}(T)$ at some point, so ℓ is rotated upwards by the next `cleanup` step, a contradiction. ◀

It remains to show how to transform $A(S, \pi)$ into an STG without free leg nodes.

► **Lemma 4.4.** *Let $T = A(S, \pi)$. Then there is a sequence of $\mathcal{O}(n + H'(G) \cdot m)$ rotations that transform T into a search tree without free leg nodes.*

Proof. Since every edge in $\text{bst}(T)$ is light, we can first transform $\text{bst}(T)$ into an arbitrary BST S' using $\mathcal{O}(n)$ rotations. We will later specify S' . Let $T' = A(S', \pi)$ be the resulting STG. Now pick the lowest leg node $\ell_{i,j}$ in T' , and rotate it down until it is bound (i.e., a child of s_i). Clearly, this requires $\text{depth}_{\text{bst}(T')}(\ell_{i,j}) = \text{depth}_{S'}(s_i)$ rotations. Repeat this until all leg nodes are bound.

The total number of rotations is

$$\mathcal{O}(n) + \sum_{i=0}^n m_i \cdot \text{depth}_{S'}(s_i).$$

This is precisely $\text{cost}(S', m_1, m_2, \dots, m_n)$, the cost of *accessing* each $i \in [n]$ with frequency m_i in the static BST S' . As such, if we choose S' to be the optimal static BST for these frequencies, we need $\mathcal{O}(n) + \text{OPT-ST}(m_1, m_2, \dots, m_n) \leq \mathcal{O}(n) + 2H'(G) \cdot m$ rotations, by Lemma 2.1. ◀

Lemmas 4.1, 4.3, and 4.4 together imply the upper bound in Theorem 1.1.

In the proof of Lemma 4.4, we essentially treat the leg nodes as queries to our optimal static BST, where a leg node $\ell_{i,j}$ queries the spine node s_i . Rotating the leg nodes down is akin to moving down the pointer in the static BST model. Here, the pointer always points at the parent of the one leg node that has a spine node parent.

Observe that we can similarly implement the dynamic BST model as rotations transforming $A(S, \pi)$ into a search tree without bound leg nodes, simply by allowing spine node rotations (BST rotations) in between leg node rotations (pointer moves). If the dynamic BST algorithm wants to rotate the single heavy edge in the spine BST of our STG, we have to move the leg node out of the way (and back afterwards), but this only adds a constant-factor overhead. Thus we obtain a generalization of the dynamic BST model, where we can start processing queries before finishing previous ones (although the way “pointers” work in this model is not very intuitive).

Let $\sigma = \sigma(\pi)$ be the sequence of spine nodes obtained by replacing every leg node in π by its adjacent spine node. Our observations imply that transforming $A(S, \pi)$ into an STG without free leg nodes requires no more than $\mathcal{O}(\text{OPT}(S, \sigma))$ rotations, and Lemma 4.4 essentially uses the fact that $\text{OPT}(S, \sigma) \leq \text{OPT-ST}(m_1, m_2, \dots, m_n)$. In the next section, we show that *Wilber's first lower bound* [22] for $\text{OPT}(S, \sigma)$ also holds for our generalized model.

5 Lower bound

We start by defining a variant of Wilber's first lower bound and proving that it is equal to the Shannon entropy of the query distribution in the worst case (up to a constant factor). Then, we show that it also bounds the rotation distance between $A(S, \sigma)$ and $B(S)$ if σ is the worst-case ordering and S is a suitable search tree.

5.1 Wilber's first lower bound for binary search trees

Let S be a binary search tree on n nodes, let $\sigma = (x_1, x_2, \dots, x_m)$ be a sequence of queries, and let u be a node of S . Then we define $\lambda(S, u, \sigma)$ as follows. If u has at most one child, then $\lambda(S, u, \sigma) = 0$. Otherwise, let v, w be the children of u and write $A = V(S_u) = \{u\} \cup V(S_v) \cup V(S_w)$. Let $\sigma|_A$ be the sequence obtained from σ by removing all elements not in A . Now $\lambda(S, u, \sigma)$ is defined as the number of times the sequence $\sigma|_A$ switches between an element of $V(S_v)$, an element of $V(S_w)$, and u . More formally, $\lambda(S, u, \sigma)$ is the number of pairs of adjacent values x, y in σ such that neither $x, y \in V(S_v)$, nor $x, y \in V(S_w)$, nor $x = y = u$. Let $\Lambda(S, \sigma) = \sum_{u \in V(S)} \lambda(S, u, \sigma)$.

For convenience, define $\lambda'(S, u, \sigma)$ as $\lambda(S, u, \sigma)$ plus the number of occurrences of u in σ , and let $\Lambda'(S, \sigma) = \sum_{u \in V(S)} \lambda'(S, u, \sigma) = \Lambda(S, \sigma) + m$. It is known that $\text{OPT}(S, \sigma) \in \Omega(\Lambda'(S, \sigma))$. This is not tight in general [7, 13]. Still, Wilber [22] showed that if σ is the *bit reversal permutation*, then $\Lambda'(S, \sigma) \in \Theta(n \log n)$ for all S . This bound is already matched by a balanced static tree, so, on that sequence, Wilber's bound is tight and dynamic BSTs do not perform better than balanced trees. We now generalize this result to arbitrary distributions.

► **Lemma 5.1.** *Let $n \in \mathbb{N}_+$, let $m_1, m_2, \dots, m_n \in \mathbb{N}_0$, and let $m = \sum_{i=1}^n m_i$. Then there is a BST S on $[n]$ and a sequence σ of length m where each $i \in [n]$ occurs precisely m_i times, such that $\Lambda'(S, \sigma) \geq \frac{1}{2} H(m_1, m_2, \dots, m_n) \cdot m$.*

Proof. We recursively construct a BST $S_{p,q}$ on each interval $[p, q]$ with $1 \leq p \leq q \leq n$, and in the end set $S = S_{1,n}$. The construction is essentially the approximately optimal static BST construction due to Mehlhorn [16].

Fix p and q . Let $k = q - p + 1$ be the number of nodes in $S_{p,q}$, and, for each i with $p \leq i \leq q$, let $a_i = \sum_{j=p}^{i-1} m_j$ and $b_i = \sum_{j=i+1}^q m_j$. We claim that there exists an $i \in [p, q]$ such that $m_i + \min(a_i, b_i) \geq \frac{k}{2}$. Suppose not. Then, for each i , we have either (1) $m_i + a_i < \frac{k}{2} < b_i$ or (2) $m_i + b_i < \frac{k}{2} < a_i$. Observe that for $i = p$, (2) cannot hold, and likewise for $i = q$, (1) cannot hold. Let i' be the maximum index where (1) holds. Then, $i < q$ and $m_i + a_i < b_i = m_{i+1} + b_{i+1} < a_{i+1} = m_i + a_i$, a contradiction.

Choose $i \in [p, q]$ such that $m_i + \min(a_i, b_i) \geq \frac{k}{2}$. Make i the root of $S_{p,q}$, and attach the recursively constructed subtrees $S_{p,i-1}$ and $S_{i+1,q}$ as the left and right child to it (for $p' > q'$, we let $S_{p',q'}$ be the empty BST).

Let $c(p, q) = \sum_{j=p}^q m_j \cdot \text{depth}_{S_{p,q}}(j)$. Intuitively, $c(p, q)$ is the cost of accessing the relevant nodes within $S_{p,q}$. We now recursively construct a sequence $\sigma_{p,q}$ such that $c(p, q) \leq 2\Lambda'(S_{p,q}, \sigma_{p,q})$.

First, if $p = q$, then let $\sigma_{p,q}$ simply consist of m_p times the element p . Clearly, $c(p, q) = m_p = \Lambda'(S_{p,q}, \sigma_{p,q})$. Otherwise, let i be the root of $S_{p,q}$. We construct $\sigma_{p,q}$ by combining $\sigma_{p,i-1}$, $\sigma_{i+1,q}$, and the m_i occurrences of the element i as follows. Start with the m_i occurrences of i , then alternate between $\sigma_{p,i-1}$ and $\sigma_{i+1,q}$ for as long as possible, and finally append the remaining elements from either $\sigma_{p,i-1}$ or $\sigma_{i+1,q}$. Since $\sigma_{p,i-1}$ has length a_i , and $\sigma_{i+1,q}$ has length b_i , we have $\lambda'(S_{p,q}, i, \sigma_{p,q}) \geq m_i + \min(a_i, b_i) \geq \frac{k}{2}$.

For each $j \in [p, i-1]$, we have $\text{depth}_{S_{p,q}}(j) = 1 + \text{depth}_{S_{p,i-1}}(j)$, and similarly for each $j \in [i+1, q]$, we have $\text{depth}_{S_{p,q}}(j) = 1 + \text{depth}_{S_{i+1,q}}(j)$. Thus, by induction,

$$\begin{aligned} c(p, q) &= m_i + a_i + c(p, i-1) + b_i + c(i+1, q) \\ &\leq k + 2\Lambda'(S_{p,i-1}, \sigma_{p,i-1}) + 2\Lambda'(S_{i+1,q}, \sigma_{i+1,q}) \\ &\leq 2\lambda'(S_{p,q}, i, \sigma_{p,q}) + 2\Lambda'(S_{p,i-1}, \sigma_{p,i-1}) + 2\Lambda'(S_{i+1,q}, \sigma_{i+1,q}) \leq 2\Lambda'(S_{p,q}, \sigma_{p,q}). \end{aligned}$$

Now let $S = S_{1,n}$ and $\sigma = \sigma_{1,n}$. We have $c(1, n) \leq 2\Lambda'(S, \sigma)$, and by Lemma 2.1, we know that $c(1, n) \geq \text{cost}(m_1, m_2, \dots, m_n) \geq H(m_1, m_2, \dots, m_n) \cdot m$. This concludes the proof. ◀

5.2 Wilber's lower bound for rotation distance

We now show that the rotation distance between $A(S, \pi)$ and $B(S)$ is at least $\frac{1}{2}\Lambda'(S, \sigma(\pi))$, where $\sigma(\pi)$ is defined as in Section 4, by replacing the leaf nodes in π with their adjacent spine nodes. Our proof is based on the lower bound on the diameter of tree associahedra by Cardinal, Langerman, and Pérez-Lantero [3, Lemma 8].

► **Lemma 5.2.** *Let $G = C(m_1, m_2, \dots, m_n)$ be a caterpillar, let S be a BST on the spine nodes of G , and let π be an ordering of the leg nodes of G . Then, transforming $A(S, \pi)$ into $B(S)$ requires at least $\frac{1}{2}\Lambda'(S, \sigma(\pi))$ rotations.*

Proof. Write $T = A(S, \pi)$, $T' = B(S)$, $\sigma = \sigma(\pi)$, and let r be the root of S . Let the set D consist of r and its adjacent legs, i.e., if $r = s_i$, then $D = \{s_i\} \cup \{\ell_{i,j} \mid j \in [m_i]\}$. Suppose r has two children u and v . Then $G \setminus D$ has two connected components, one consisting of the spine nodes $V(S_u)$ and all adjacent legs, and the other consisting of $V(S_v)$ and all adjacent legs. Call the former E and latter F . If r has only one child u , let E consist of $V(S_u)$ and all adjacent legs, and let $F = \emptyset$. If r has no children, let $E = F = \emptyset$. Note that D, E, F form a partition of $V(G)$.

We first consider the rotations within each of the three sets. By Lemma 2.2, we can simply sum up the number of rotations required to transform $T[D]$, $T[E]$, and $T[F]$ into $T'[D]$, $T'[E]$, $T'[F]$, respectively.

$T[D]$ consists of the spine node r and m_r free leg nodes, and $T'[D]$ consists of r and m_r bound leg nodes. Thus, we need m_r rotations to make all leg nodes bound.

If $E \neq \emptyset$, observe that $T[E] = A(S_u, \pi|_E)$ and $T'[E] = B(S_u)$, so we need $\frac{1}{2}\Lambda'(S_u, \sigma|_E) = \frac{1}{2}\Lambda'(S_u, \sigma)$ rotations by induction. If $F \neq \emptyset$, we similarly get a lower bound of $\frac{1}{2}\Lambda'(S_v, \sigma)$ to transform $T[F]$ into $T'[F]$.

We now show that there are at least $\frac{1}{2}\lambda(S, r, \sigma)$ rotations between *different* parts of the partition D, E, F . If $\lambda(S, r, \sigma) = 0$, this is trivially true, so suppose $\lambda(S, r, \sigma) > 0$ and thus $E \neq \emptyset$.

Define the *alternation number* of a path P in a search tree on G as the number of edges (x, y) in P such that x and y are in different parts of the partition D, E, F . Define the alternation number $\text{alt}(T^*)$ of a search tree T^* on G as the maximum alternation number among all paths starting at the root in T^* . Observe that $\text{alt}(T') = 1$, and that $\text{alt}(T) \geq \lambda(S, r, \sigma) + 1$, since the leg nodes in T have $\lambda(S, r, \sigma)$ alternations by definition, and there is one more alternation from r to $E \neq \emptyset$.

We now show how rotations affect the alternation number. Consider a rotation between the nodes x and y , and a node z . The path from the root to z before and after the rotation may only differ if it contains x or y (or both), and only in one of the following ways:

- x is inserted before y , or y is inserted before x .
- x is deleted before y , or y is deleted before x .
- x and y are swapped (and are neighbors).

It is easy to see that if $x, y \in D$, or $x, y \in E$, or $x, y \in F$, then the rotation (x, y) does not affect the alternation number, and otherwise, it can only differ by at most two. This means that we need at least $\frac{1}{2}|\text{alt}(T) - \text{alt}(T')| \geq \frac{1}{2}\lambda(S, r, \sigma)$ rotations not within one of the sets D , E , or F . The total number of rotations is thus at least (setting $\Lambda'(S', \sigma) = 0$ if S' is empty):

$$m_r + \frac{1}{2}\Lambda'(S_u, \sigma) + \frac{1}{2}\Lambda'(S_v, \sigma) + \frac{1}{2}\lambda(S, r, \sigma) \geq \frac{1}{2}\Lambda'(S, \sigma). \quad \blacktriangleleft$$

Lemmas 5.1 and 5.2 together imply that $\delta(\mathcal{A}(G)) \geq \frac{1}{4}H(G) \cdot m$. As mentioned in the introduction, Manneville and Pilaud [15] proved that $\delta(\mathcal{A}(G)) \in \Omega(m + n)$. This concludes the proof of the lower bound.

6 Conclusion

In this paper, we determined the diameter of each caterpillar associahedron up to a constant, revealing a surprising connection to searching in static and dynamic binary search trees. In particular, transforming $A(S, \pi)$ into $B(S)$ via rotations can be seen as a generalization of serving the access sequence $\sigma(\pi)$ in a dynamic BST. The number of rotations required is between Wilber's first lower bound $\Lambda(S, \sigma(\pi))$ and $\text{OPT}(S, \sigma(\pi))$, begging the question whether other lower bounds for OPT hold in our generalized model, or whether it perhaps matches Λ or OPT. Results in this direction could give new insight into the dynamic BST model.

References

- 1 Benjamin Aram Berendsohn and László Kozma. Splay trees on trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1875–1900, 2022. doi:10.1137/1.9781611977073.75.
- 2 Prosenjit Bose, Jean Cardinal, John Iacono, Grigorios Koumoutsos, and Stefan Langerman. Competitive online search trees on trees. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1878–1891, 2020. doi:10.1137/1.9781611975994.115.
- 3 Jean Cardinal, Stefan Langerman, and Pablo Pérez-Lantero. On the diameter of tree associahedra. *The Electronic Journal of Combinatorics*, 2018. doi:10.37236/7762.
- 4 Jean Cardinal, Lionel Pournin, and Mario Valencia-Pabon. Bounds on the diameter of graph associahedra. *Procedia Computer Science*, 195:239–247, 2021. Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium. doi:10.1016/j.procs.2021.11.030.
- 5 Jean Cardinal, Lionel Pournin, and Mario Valencia-Pabon. Diameter estimates for graph associahedra. *arXiv e-prints*, 2021. arXiv:2106.16130.
- 6 Michael Carr and Satyan L. Devadoss. Coxeter complexes and graph-associahedra. *Topology and its Applications*, 153:2155–2168, 2004. doi:10.1016/j.topol.2005.08.010.
- 7 Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the Strong Wilber 1 Bound for Binary Search Trees. In Jarosław Byrka and Raghv Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.33.

- 8 S. Cleary. Restricted rotation distance between binary trees. *Inf. Process. Lett.*, 84:333–338, 2002. doi:10.1016/S0020-0190(02)00315-0.
- 9 Erik D. Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2009. doi:10.1137/1.9781611973068.55.
- 10 Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality – almost. *SIAM Journal on Computing*, 37(1):240–251, 2007. doi:10.1137/S0097539705447347.
- 11 D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, March 1971. doi:10.1007/BF00264289.
- 12 Jussi Kujala and Tapio Elomaa. The cost of offline binary search tree algorithms and the complexity of the request sequence. *Theoretical Computer Science*, 393(1):231–239, 2008. doi:10.1016/j.tcs.2007.12.015.
- 13 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber’s bounds for dynamic optimality. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.68.
- 14 J.M. Lucas. Canonical forms for competitive binary search tree algorithms. Technical report DCS-TR-250, Department of Computer Science, Hill Center for the Mathematical Sciences, Busch Campus, Rutgers University, New Brunswick, New Jersey 08903, 1988.
- 15 Thibault Manneville and Vincent Pilaud. Graph properties of graph associahedra. *Séminaire Lotharingien de Combinatoire*, 73, 2015. URL: <https://www.mat.univie.ac.at/~slc/wpapers/s73mannpil.html>.
- 16 Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, December 1975. doi:10.1007/BF00264563.
- 17 J. Ian Munro. On the competitiveness of linear search. In Mike S. Paterson, editor, *Algorithms – ESA 2000*, pages 338–345, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-45253-2_31.
- 18 L. Pournin. The asymptotic diameter of cyclohedra. *Israel Journal of Mathematics*, 219:609–635, 2014. doi:10.1007/s11856-017-1492-0.
- 19 Lionel Pournin. The diameter of associahedra. *Advances in Mathematics*, 259:13–42, 2014. doi:10.1016/j.aim.2014.02.035.
- 20 D. Sleator and R. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32:652–686, 1985. doi:10.1145/3828.3835.
- 21 Daniel D Sleator, Robert E Tarjan, and William P Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1(3):647–681, 1988. doi:10.1090/S0894-0347-1988-0928904-4.
- 22 Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM journal on Computing*, 18(1):56–67, 1989. doi:10.1137/0218004.

Stable Approximation Algorithms for the Dynamic Broadcast Range-Assignment Problem

Mark de Berg 

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Arpan Sadhukhan 

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Frits Spieksma 

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Abstract

Let P be a set of points in \mathbb{R}^d (or some other metric space), where each point $p \in P$ has an associated transmission range, denoted $\rho(p)$. The range assignment ρ induces a directed communication graph $\mathcal{G}_\rho(P)$ on P , which contains an edge (p, q) iff $|pq| \leq \rho(p)$. In the broadcast range-assignment problem, the goal is to assign the ranges such that $\mathcal{G}_\rho(P)$ contains an arborescence rooted at a designated root node and the cost $\sum_{p \in P} \rho(p)^2$ of the assignment is minimized.

We study the dynamic version of this problem. In particular, we study trade-offs between the stability of the solution – the number of ranges that are modified when a point is inserted into or deleted from P – and its approximation ratio. To this end we introduce the concept of *k-stable algorithms*, which are algorithms that modify the range of at most k points when they update the solution. We also introduce the concept of a *stable approximation scheme*, or *SAS* for short. A SAS is an update algorithm ALG that, for any given fixed parameter $\varepsilon > 0$, is $k(\varepsilon)$ -stable and that maintains a solution with approximation ratio $1 + \varepsilon$, where the stability parameter $k(\varepsilon)$ only depends on ε and not on the size of P . We study such trade-offs in three settings.

- For the problem in \mathbb{R}^1 , we present a SAS with $k(\varepsilon) = O(1/\varepsilon)$. Furthermore, we prove that this is tight in the worst case: any SAS for the problem must have $k(\varepsilon) = \Omega(1/\varepsilon)$. We also present algorithms with very small stability parameters: a 1-stable $(6 + 2\sqrt{5})$ -approximation algorithm – this algorithm can only handle insertions – a (trivial) 2-stable 2-approximation algorithm, and a 3-stable 1.97-approximation algorithm.
- For the problem in \mathbb{S}^1 (that is, when the underlying space is a circle) we prove that no SAS exists. This is in spite of the fact that, for the static problem in \mathbb{S}^1 , we prove that an optimal solution can always be obtained by cutting the circle at an appropriate point and solving the resulting problem in \mathbb{R}^1 .
- For the problem in \mathbb{R}^2 , we also prove that no SAS exists, and we present a $O(1)$ -stable $O(1)$ -approximation algorithm.

Most results generalize to when the range-assignment cost is $\sum_{p \in P} \rho(p)^\alpha$, for some constant $\alpha > 1$. All omitted theorems and proofs are available in the full version of the paper [14].

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Computational geometry, online algorithms, broadcast range assignment, stable approximation schemes

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.15

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2112.05426>

Funding MdB, AS, and FS are supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.



© Mark de Berg, Arpan Sadhukhan, and Frits Spieksma;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 15; pp. 15:1–15:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank the reviewers of an earlier version of the paper for pointing us to some important references and for other helpful comments.

1 Introduction

The broadcast range-assignment problem. Let P be a set of points in \mathbb{R}^d , representing transmission devices in a wireless network. By assigning each point $p \in P$ a transmission range $\rho(p)$, we obtain a *communication graph* $\mathcal{G}_\rho(P)$. The nodes in $\mathcal{G}_\rho(P)$ are the points from P and there is a directed edge (p, q) iff $|pq| \leq \rho(p)$, where $|pq|$ denotes the Euclidean distance between p and q . The energy consumption of a device depends on its transmission range: the larger the range, the more energy it needs. More precisely, the energy needed to obtain a transmission range $\rho(p)$ is given by $\rho(p)^\alpha$, for some real constant $\alpha > 1$ called the *distance-power gradient*. In practice, α depends on the environment and ranges from 1 to 6 [18]. Thus the overall cost of a range assignment is $\text{cost}_\alpha(\rho(P)) := \sum_{p \in P} \rho(p)^\alpha$, where we use $\rho(P)$ to denote the set of ranges given to the points in P by the assignment ρ . The goal of the range-assignment problem is to assign the ranges such that $\mathcal{G}_\rho(P)$ has certain connectivity properties while minimizing the total cost [6]. Desirable connectivity properties are that $\mathcal{G}_\rho(P)$ is (h -hop) strongly connected [8, 9, 10, 16] or that $\mathcal{G}_\rho(P)$ contains a *broadcast tree*, that is, an arborescence rooted at a given source $s \in P$. The latter property leads to the *broadcast range-assignment problem*, which is the topic of our paper.

The broadcast range-assignment problem has been studied extensively, sometimes with the extra condition that any point in P is reachable in at most h hops from the source s . For $\alpha = 1$ the problem is trivial in any dimension: setting the range of the source s to $\max\{|sp| : p \in P\}$ and all other ranges to zero is optimal; however, for any $\alpha > 1$ the problem is NP-hard in \mathbb{R}^d for $d \geq 2$ [5, 15]. Approximation algorithms and results on hardness of approximation are known as well [4, 7, 15]. Many of our results will be on the 1-dimensional (or: linear) broadcast range-assignment problem. Linear networks are important for modeling road traffic information systems [3, 17] and as such they have received ample attention. In \mathbb{R}^1 , the broadcast range-assignment problem is no longer NP-hard, and several polynomial-time algorithms have been proposed, for the standard version, the h -hop version, as well as the weighted version [2, 4, 7, 11, 12]. The currently fastest algorithms for the (standard and h -hop) broadcast range-assignment problem run in $O(n^2)$ time [11].

All results mentioned so far are for the static version of the problem. Our interest lies in the dynamic version, where points can be inserted into and deleted from P (except the source, which should always be present). This corresponds to new sensors being deployed and existing sensors being removed, or, in a traffic scenario, cars entering and exiting the highway. Recomputing the range assignment from scratch when P is updated may result in all ranges being changed. The question we want to answer is therefore: is it possible to maintain a close-to-optimal range assignment that is relatively stable, that is, an assignment for which only few ranges are modified when a point is inserted into or deleted from P ? And which trade-offs can be achieved between the quality of the solution and its stability?

To the best of our knowledge, the dynamic problem has not been studied so far. The online problem, where the points from P arrive one by one (there are no deletions) and it is not allowed to decrease ranges, is studied by De Berg et al. [13]. This restriction is arguably unnatural, and it has the consequence that a bounded approximation ratio cannot be achieved. Indeed, let the source s be at $x = 0$, and suppose that first the point $x = 1$ arrives, forcing us to set $\rho(s) := 1$, and then the points $x = i/n$ arrive for $1 \leq i < n$. In the optimal static solution at the end of this scenario all points, except the rightmost one, have

range $1/n$; for $\alpha = 2$ this induces a total cost of $n \cdot (1/n)^2 = 1/n$. But if we are not allowed to decrease the range of s after setting $\rho(s) = 1$, the total cost will be (at least) 1, leading to an unbounded approximation ratio. Therefore, [13] analyze the competitive ratio: they compare the cost of their algorithm to the cost of an optimal offline algorithm (which knows the future arrivals, but must still maintain a valid solution at all times without decreasing any range). As we will see, by allowing to also decrease a few ranges, we are able to maintain solutions whose cost is close even to the static optimum.

Our contribution. Before we state our results, we first define the framework we use to analyze our algorithms. Let P be a dynamic set of points in \mathbb{R}^d , which includes a fixed source point s that cannot be deleted.

An update algorithm ALG for the dynamic broadcast range-assignment problem is an algorithm that, given the current solution (the current ranges of the points in the current set P) and the location of the new point to be inserted into P , or the point to be deleted from P , modifies the range assignment so that the updated solution is a valid broadcast range assignment for the updated set P . We call such an update algorithm k -stable if it modifies at most k ranges when a point is inserted into or deleted from P . Here we define the range of a point currently not in P to be zero. Thus, if a newly inserted point receives a positive range it will be counted as receiving a modified range; similarly, if a point with positive range is deleted then it will be counted as receiving a modified range. To get a more detailed view of the stability, we sometimes distinguish between the number of increased ranges and the number of decreased ranges, in the worst case. When these numbers are k^+ and k^- , respectively, we say that ALG is (k^+, k^-) -stable. This is especially useful when we separately report on the stability of insertions and deletions; often, when insertions are (k_1, k_2) -stable then deletions will be (k_2, k_1) -stable.

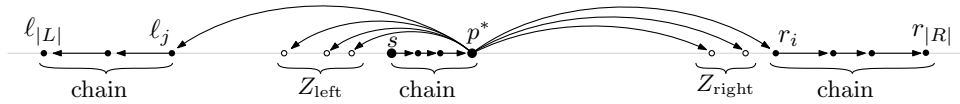
We are not only interested in the stability of our update algorithms, but also in the quality of the solutions they provide. We measure this in the usual way, by considering the approximation ratio of the solution. As mentioned, we are interested in trade-offs between the stability of an algorithm and its approximation ratio. Of particular interest are so-called stable approximation schemes, defined as follows.

► **Definition 1.** A stable approximation scheme, or SAS for short, is an update algorithm ALG that, for any given yet fixed parameter $\varepsilon > 0$, is $k(\varepsilon)$ -stable and that maintains a solution with approximation ratio $1 + \varepsilon$, where the stability parameter $k(\varepsilon)$ only depends on ε and not on the size of P .

Notice that in the definition of a SAS we do not take the computational complexity of the update algorithm into account. We point out that, in the context of dynamic scheduling problems (where jobs arrive and disappear in an online fashion, and it is allowed to re-assign jobs), a related concept has been introduced under the name *robust PTAS*: a polynomial-time algorithm that, for any given parameter $\varepsilon > 0$, computes a $(1 + \varepsilon)$ -approximation with re-assignment costs only depending on ε , see e.g. [19] and [20].

We now present our results. Recall that $\text{cost}_\alpha(\rho(P)) := \sum_{p \in P} \rho(p)^\alpha$, is the cost of a range assignment ρ , where $\alpha > 1$ is a constant. To make the results easier to interpret, we state the results for $\alpha = 2$; the dependencies of the bounds on the parameter α can be found in the theorems presented in later sections.

- In Section 3 we present a SAS for the broadcast range-assignment problem in \mathbb{R}^1 , with $k(\varepsilon) = O(1/\varepsilon)$. We prove that this is tight in the worst case, by showing that any SAS for the problem must have $k(\varepsilon) = \Omega(1/\varepsilon)$.



■ **Figure 1** The structure of an optimal solution. The non-filled points are zero-range points, the solid black points all have a standard range (for $\ell_{|L|}$ and $r_{|R|}$ the standard range is zero), except for the root-crossing point which (in this example) has a long range.

- Our SAS (as well as some other algorithms) needs to know an optimal solution after each update. The fastest existing algorithms to compute an optimal solution in \mathbb{R}^1 run in $O(n^2)$ time. In Section 2 we show how to recompute an optimal solution in $O(n \log n)$ time after each update, which we believe to be of independent interest. As a result, our SAS also runs in $O(n \log n)$ time per update.
- There is a very simple 2-stable 2-approximation algorithm. We show that a 1-stable algorithm with bounded approximation ratio does not exist when both insertions and deletions must be handled. For the insertion-only case, however, we give a 1-stable $(6 + 2\sqrt{5})$ -approximation algorithm. We have not been able to improve upon the approximation ratio 2 with a 2-stable algorithm, but we show that with a 3-stable we can get a 1.97-approximation. Due to lack of space, these results are mostly delegated to the appendix.
- Next we study the problem in \mathbb{S}^1 , that is, when the underlying 1-dimensional space is circular. This version has, as far as we know, not been studied so far. We first prove that in \mathbb{S}^1 an optimal solution for the static problem can always be obtained by cutting the circle at an appropriate point and solving the resulting problem in \mathbb{R}^1 . This leads to an algorithm to solve the static problem optimally in $O(n^2 \log n)$ time. We also prove that, in spite of this, a SAS does not exist in \mathbb{S}^1 .
- Finally, we consider the problem in \mathbb{R}^2 . Based on the no-SAS proof in \mathbb{S}^1 , we show that the 2-dimensional problem does not admit a SAS either. In addition, we present an 17-stable 12-approximation algorithm for the 2-dimensional version of the problem.

All omitted results and proofs are there in the full version of the paper [14].

2 Maintaining an optimal solution in \mathbb{R}^1

Before we can present our stable algorithms for the broadcast range-assignment problem in \mathbb{R}^1 , we first introduce some terminology and we discuss the structure of optimal solutions. We also present an efficient subroutine to maintain an optimal solution.

2.1 The structure of an optimal solution

Several papers have characterized the structure of optimal broadcast range assignments in \mathbb{R}^1 , in a more or less explicit manner. We use the characterization by Caragiannis et al. [4], which is illustrated in Figure 1 and described next.

Let $P := L \cup \{s\} \cup R$ be a point set in \mathbb{R}^1 . Here s is the designated source node, $L := \{\ell_1, \dots, \ell_{|L|}\}$ contains all points from P to the left of s , and $R := \{r_1, \dots, r_{|R|}\}$ contains all points to the right of s . The points in L are numbered in order of increasing distance from s , and the same is true for the points in R . The points $\ell_{|L|}$ and $r_{|R|}$ are called *extreme points*. In the following, and with a slight abuse of notation, we sometimes use p or q to refer a generic point from P – that is, a point that could be s , or a point from R , or a point from L . Furthermore, we will not distinguish between points in P and the corresponding nodes in the communication graph $\mathcal{G}_\rho(P)$.

For a non-extreme point $r_i \in R$, we define r_{i+1} to be its *successor*; similarly, ℓ_{i+1} is the successor of ℓ_i . The source s has (at most) two successors, namely r_1 and ℓ_1 . The successor of a point p is denoted by $\text{succ}(p)$; for an extreme point p we define $\text{succ}(p) = \text{NIL}$. If $\text{succ}(p) = q \neq \text{NIL}$, then we call p the *predecessor* of q and we write $\text{pred}(q) = p$. A *chain* is a path in the communication graph $\mathcal{G}_\rho(P)$ that only consists of edges connecting a point to its successor. Thus a chain either visits consecutive points from $\{s\} \cup R$ from left to right, or it visits consecutive points from $\{s\} \cup L$ from right to left. It will be convenient to consider the empty path from s to itself to be a chain as well.

Consider a range assignment ρ . We say that a point $q \in P$ is *within reach* of a point $p \in P$ if $|pq| \leq \rho(p)$. Let \mathcal{B} a broadcast tree in $\mathcal{G}_\rho(P)$ – that is, \mathcal{B} is an arborescence rooted at s . A point in $R \cup L$ in \mathcal{B} is called *root-crossing* in \mathcal{B} if it has a child on the other side of s ; the source s is root-crossing if it has a child in L and a child in R . The following theorem, which holds for any distance-power gradient $\alpha > 1$, is proven in [4].

► **Theorem 2** ([4]). *Let P be a point set in \mathbb{R}^1 . If all points in $P \setminus \{s\}$ lie to the same side of the source s , then the optimal solution induces a chain from s to the extreme point in P . Otherwise, there is an optimal range assignment ρ such that $\mathcal{G}_\rho(P)$ contains a broadcast tree \mathcal{B} with the following structure:*

- \mathcal{B} has a single root-crossing point, p^* .
- \mathcal{B} contains a chain from s to p^* .
- All points within reach of p^* , except those on the chain from s to p^* , are children of p^* .
- Let r_i and ℓ_j be the rightmost and leftmost point within reach of p^* , respectively. Then \mathcal{B} contains a chain from r_i to $r_{|R|}$, and a chain from ℓ_j to $\ell_{|L|}$.

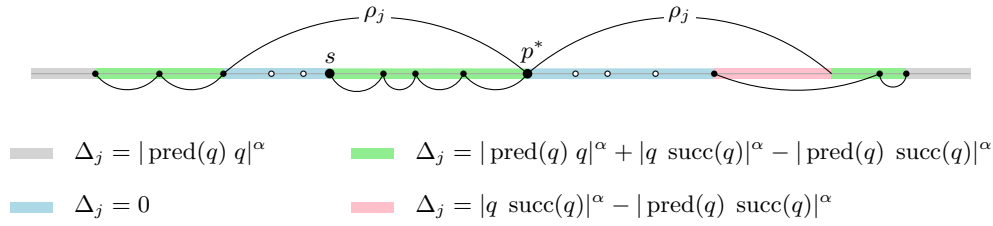
From now on, whenever we talk about optimal range assignments and their induced broadcast trees, we implicitly assume that the broadcast tree has the structure described in Theorem 2. Note that the communication graph $\mathcal{G}_\rho(P)$ induced by an optimal range assignment ρ can contain more edges than the ones belonging to the broadcast tree \mathcal{B} . Obviously, for ρ to be optimal it must be a minimum-cost assignment inducing \mathcal{B} .

Define the *standard range* of a non-extreme point $r_i \in R$ to be $|r_i r_{i+1}|$; the standard range of the extreme point $r_{|R|}$ is defined to be zero. The standard ranges of the points in L are defined similarly. The source s has two standard ranges, $|s\ell_1|$ and $|sr_1|$. A range assignment in which every point has a standard range is called a *standard solution*; a standard solution may or may not be optimal. Note that, in the static problem, it is never useful to give a point a non-zero range that is smaller than its standard range. Hence, we only need to consider three types of points: *standard-range points*, *zero-range points*, and *long-range points*. Here zero-range points are non-extreme points with a zero range, and a point is said to have a *long range* if its range is greater than its standard range. Theorem 2 implies that an optimal range assignment has the following properties; see also Figure 1.

- There is at most one long-range point.
- The set $Z \subset P$ of zero-range points (which may be empty) can be partitioned into two subsets, Z_{left} and Z_{right} , such that Z_{left} consists of consecutive points that lie to the left of the source s , and Z_{right} consists of consecutive points that lie to the right of s .

2.2 An efficient update algorithm

Using Theorem 2 an optimal solution for the broadcast range-assignment problem can be computed in $O(n^2)$ time [11]. Below we show that maintaining an optimal solution under insertions and deletions can be done more efficiently than by re-computing it from scratch: using a suitable data structure, we can update the solution in $O(n \log n)$ time. This will also be useful in later sections, when we give algorithms that maintain a stable solution.



■ **Figure 2** Various cases that can arise when a new point q is inserted into P . Open disks indicate zero-range points. The arcs indicate the ranges of the points before the insertion of q , where the range of the root-crossing point is drawn both to its right and to its left. The colored intervals relate the possible locations of q to the corresponding values Δ_j , where Δ_j refers to the (signed) difference of the cost of the range assignment before and after the insertion of q .

Recall that an optimal solution for a given point set P has a single root-crossing point, p^* . Once the range $\rho(p^*)$ is fixed, the solution is completely determined. Since $\rho(p^*) = |p^*p|$ for some point $p \neq p^*$, there are $n - 1$ candidate ranges for a given choice of the root-crossing point p^* . The idea of our solution is to implicitly store the cost of the range assignment for each candidate range of p^* such that, upon the insertion or deletion of a point in P , we can find the best range for p^* in $O(\log n)$ time. By maintaining n such data structures \mathcal{T}_{p^*} , one for each choice of the root-crossing point p^* , we can then find the overall best solution.

The data structure for a given root-crossing point. Next we explain our data structure for a given candidate root-crossing point p^* . We assume without loss of generality that p^* lies to the right of the source point s ; it is straightforward to adapt the structure to the (symmetric) case where p^* lies to the left of s , and to the case where $p^* = s$.

Let \mathcal{R}_{p^*} be the set of all ranges we need to consider for p^* , for the current set P . The range of a root-crossing point must extend beyond the source point. Hence,

$$\mathcal{R}_{p^*} := \{|p^*p| : p \in P \text{ and } |p^*p| > |p^*s|\}.$$

Let $\lambda_1, \dots, \lambda_m$ denote the sequence of ranges in \mathcal{R}_{p^*} , ordered from small to large. (If $\mathcal{R}_{p^*} = \emptyset$, there is nothing to do and our data structure is empty.) As mentioned, once we fix a range λ_j for the given root-crossing point p^* , the solution is fully determined by Theorem 2: there is a chain from s to p^* , a chain from the rightmost point within range of p^* to the right-extreme point, and a chain from the leftmost point within range of p^* to the left-extreme point. We denote the resulting range assignment¹ for P by $\Gamma(P, p^*, \lambda_j)$.

Our data structure, which implicitly stores the costs of the range assignments $\Gamma(P, p^*, \lambda_j)$ for all $\lambda_j \in \mathcal{R}_{p^*}$, is an augmented balanced binary search tree \mathcal{T}_{p^*} . The key to the efficient maintenance of \mathcal{T}_{p^*} is that, upon the insertion of a new point p (or the deletion of an existing point), many of the solutions change in the same way. To formalize this, let Δ_j be the signed difference of the cost of the range assignment $\Gamma(P, p^*, \lambda_j)$ before and after the insertion of q , where Δ_j is positive if the cost increases. Figure 2 shows various possible values for Δ_j , depending on the location of the new point q with respect to the range λ_j . It follows from the figure that there are only four possible values for Δ_j . This allows us to design our data structure \mathcal{T}_{p^*} such that it can be updated using $O(1)$ *bulk updates* of the following form:

¹ When P lies completely to one side of s , then the range assignment is formally not root-crossing. We permit ourselves this slight abuse of terminology because by considering s as root-crossing point, setting $\rho(s) := |s \text{succ}(s)|$ and adding a chain from $\text{succ}(s)$ to the extreme point, we get an optimal solution.

Given an interval I of range values and an update value Δ , add Δ to the cost of $\Gamma(P, p^*, \lambda_j)$ for all $\lambda_j \in I$.

In the full version of the paper [14], we define the information stored in \mathcal{T}_{p^*} and we show how bulk updates can be done in $O(\log n)$ time. We eventually obtain the following theorem.

► **Theorem 3.** *An optimal solution to the broadcast range-assignment problem for a point set P in \mathbb{R}^1 can be maintained in $O(n \log n)$ per insertion and deletion, where n is the number of points in the current set P .*

3 A stable approximation scheme in \mathbb{R}^1

In this section we use the structure of an optimal solution provided by Theorem 2 to obtain a SAS for the 1-dimensional broadcast range-assignment problem. Our SAS has stability parameter $k(\varepsilon) = O((1/\varepsilon)^{1/(\alpha-1)})$, which we will show to be asymptotically optimal.

The optimal range assignment can be very unstable. Indeed, suppose the current point set is $P := \{s, r_1, \dots, r_n\}$ with $s = 0$ and $r_i = i$ ($1 \leq i \leq n$), and take any $\alpha > 1$. Then the (unique) optimal assignment ρ_{opt} has $\rho_{\text{opt}}(s) = \rho_{\text{opt}}(r_1) = \dots = \rho_{\text{opt}}(r_{n-1}) = 1$ and $\rho_{\text{opt}}(r_n) = 0$. If now the point $\ell_1 = -n$ is inserted, then the optimal assignment becomes $\rho_{\text{opt}}(s) = n$ and $\rho_{\text{opt}}(r_1) = \dots = \rho_{\text{opt}}(r_n) = \rho_{\text{opt}}(\ell_1) = 0$, causing n ranges to be modified.

Next, we will define a feasible solution, referred to as a *canonical range assignment* ρ_k that is more stable than an optimal assignment, while still having a cost close to the cost of an optimal solution. Here k is a parameter that allows a trade-off between stability and quality of the solution. The assignment ρ_k for a given point set P will be uniquely determined by the set P —it does not depend on the order in which the points have been inserted or deleted. This means that the update algorithm simply works as follows. Let $\rho_k(P)$ be the canonical range assignment for a point set P , and suppose we update P by inserting a point q . Then the update algorithm computes $\rho_k(P \cup \{q\})$ and it modifies the range of each point $p \in P \cup \{q\}$ whose canonical range in $\rho_k(P \cup \{q\})$ is different from its canonical range in $\rho_k(P)$. The goal is now to specify ρ_k such that (i) many ranges in $\rho_k(P \cup \{q\})$ are the same as in $\rho_k(P)$, (ii) the cost of $\rho_k(P)$ is close to the cost of $\rho_{\text{opt}}(P)$.

The instance in the example above shows that there can be many points whose range changes from being standard to being zero (or vice versa) when preserving optimality of the consecutive instances. Our idea is therefore to construct solutions where the number of points with zero range is limited, and instead give many points their standard range; if we do this for points whose standard range is relatively small, then the cost of this solution remains bounded compared to the cost of an optimum solution. We now make this idea precise.

Consider a point set P and let ρ_{opt} be an optimal range assignment satisfying the structure described in Theorem 2. Assuming there are points in P on both sides of the source, ρ_{opt} induces a broadcast tree \mathcal{B} with the structure depicted in Figure 1. Let $\rho_{\text{st}}(p)$ be the standard range of a point p . The canonical range assignment ρ_k is now defined as follows.

- If all points from P lie to the same side of s , then $\rho_k(p) := \rho_{\text{opt}}(p)$ for all $p \in P$. Note that in this case $\rho_k(p) = \rho_{\text{st}}(p)$ for all $p \in P$.
- Otherwise, let Z be the set of zero-range points in $\rho_{\text{opt}}(P)$. If $|Z| \leq k$ then let $Z_k := Z$; otherwise let $Z_k \subseteq Z$ be the k points from Z with the largest standard ranges, with ties broken arbitrarily. We define ρ_k as follows.
 - $\rho_k(p) := \rho_{\text{opt}}(p)$ for all $p \in P \setminus Z$. Observe that this means that $\rho_k(p) = \rho_{\text{st}}(p)$ for all $p \in P \setminus Z$ except (possibly) for the root-crossing point.
 - $\rho_k(p) := 0$ for all $p \in Z_k$.
 - $\rho_k(p) := \rho_{\text{st}}(p)$ for all $p \in Z \setminus Z_k$.

15:8 Stable Approximation Algorithms for Range Assignment

Notice that ρ_k is a feasible solution since $\rho_k(p) \geq \rho_{\text{opt}}(p)$ for each $p \in P$. The next lemma analyzes the stability of the canonical range assignment ρ_k . Recall that for any range assignment ρ – hence, also for ρ_k – and any point q not in the current set P , we have $\rho(q) = 0$ by definition.

► **Lemma 4.** *Consider a point set P and a point $q \notin P$. Let $\rho_{\text{old}}(p)$ be the range of a point p in $\rho_k(P)$ and let $\rho_{\text{new}}(p)$ be the range of p in $\rho_k(P \cup \{q\})$. Then*

$$|\{p \in P \cup \{q\} : \rho_{\text{new}}(p) > \rho_{\text{old}}(p)\}| \leq k+3 \text{ and } |\{p \in P \cup \{q\} : \rho_{\text{new}}(p) < \rho_{\text{old}}(p)\}| \leq k+3.$$

Proof. The range of a point $p \in P \cup \{q\}$ can increase due to the insertion of q only if

- (i) $p = q$ and $\rho_{\text{new}}(q) > 0$, or
- (ii) p is a zero-range point in $\rho_k(P)$, or
- (iii) p is the root-crossing point in $\rho_k(P \cup \{q\})$, or
- (iv) the standard range of p increases due to the insertion of q , or
- (v) $p = s$ and, out of the two standard ranges it has, s gets assigned a larger one in $\rho_k(P \cup \{q\})$ than in $\rho_k(P)$.

Recall that we defined ρ_k such that the number of zero-range points is at most k . Furthermore, at most one standard range can increase due to the insertion of q , namely, the standard range of a point that is extreme in P but not in $P \cup \{q\}$. When this happens, however, q is extreme in $P \cup \{q\}$ and so $\rho_{\text{new}}(q) = 0$; this implies that cases (i) and (iv) cannot both happen. Hence, $|\{p \in P \cup \{q\} : \rho_{\text{new}}(p) > \rho_{\text{old}}(p)\}| \leq k+3$.

The range of a point p can decrease only if

- (i) p is a zero-range point in $\rho_k(P \cup \{q\})$, or
- (ii) p is the root-crossing point in $\rho_k(P)$, or
- (iii) the standard range of p decreases due to the insertion of q , or
- (iv) $p = s$ and, out of the two standard ranges it has, p gets assigned a smaller one in $\rho_k(P \cup \{q\})$ than in $\rho_k(P)$.

Since the only point whose standard range decreases is the predecessor of q in P , we conclude that $|\{p \in P \cup \{q\} : \rho_{\text{new}}(p) < \rho_{\text{old}}(p)\}| \leq k+3$. ◀

Next we bound the approximation ratio of ρ_k .

► **Lemma 5.** *For any set P and any $\alpha > 1$, we have $\text{cost}_\alpha(\rho_k(P)) \leq (1 + \frac{2^\alpha}{k^{\alpha-1}}) \cdot \text{cost}_\alpha(\rho_{\text{opt}}(P))$.*

Proof. If all points in P lie to the same side of s then $\rho_k(P) = \rho_{\text{opt}}(P)$, and we are done. Otherwise, let p^* be the root-crossing point. The only points receiving a different range in $\rho_k(P)$ when compared to $\rho_{\text{opt}}(P)$ are the points in $Z \setminus Z_k$; these points have $\rho_k(p) = \rho_{\text{st}}(p)$ while $\rho_{\text{opt}}(p) = 0$. This means we are done when $Z \setminus Z_k = \emptyset$. Thus we can assume that $|Z| > k$, so $Z \setminus Z_k \neq \emptyset$. Assume without loss of generality that $\rho_{\text{opt}}(p^*) = 1$. As each $p \in Z$ is within reach of p^* , we have $\sum_{p \in Z} \rho_{\text{st}}(p) \leq 2$. Since Z_k contains the k points with the largest standard ranges among the points in Z , we have $\max\{\rho_{\text{st}}(p) : p \in Z \setminus Z_k\} \leq 2/k$. Hence,

$$\sum_{p \in Z \setminus Z_k} \rho_k(p)^\alpha = \sum_{p \in Z \setminus Z_k} \rho_{\text{st}}(p)^\alpha = \sum_{p \in Z \setminus Z_k} \rho_{\text{st}}(p)^{\alpha-1} \cdot \rho_{\text{st}}(p) \leq \left(\frac{2}{k}\right)^{\alpha-1} \sum_{p \in Z \setminus Z_k} \rho_{\text{st}}(p) \leq \frac{2^\alpha}{k^{\alpha-1}}.$$

(The analysis can be made tighter by using that $\sum_{p \in Z \setminus Z_k} \rho_{\text{st}}(p) \leq 2 - k \max_{p \in Z \setminus Z_k} \rho_{\text{st}}(p)$, but this will not change the approximation ratio asymptotically.) We conclude that

$$\frac{\text{cost}_\alpha(\rho_k(P))}{\text{cost}_\alpha(\rho_{\text{opt}}(P))} \leq \frac{\sum_{p \in P \setminus (Z \setminus Z_k)} \rho_k(p)^\alpha + \sum_{p \in Z \setminus Z_k} \rho_k(p)^\alpha}{\sum_{p \in P \setminus (Z \setminus Z_k)} \rho_{\text{opt}}(p)^\alpha} \leq 1 + \frac{2^\alpha}{k^{\alpha-1}},$$

where the last inequality follows because we have $\rho_k(p) = \rho_{\text{opt}}(p)$ for all $p \in P \setminus (Z \setminus Z_k)$ and $\sum_{p \in P \setminus (Z \setminus Z_k)} \rho_{\text{opt}}(p)^\alpha \geq 1$. ◀

By maintaining the canonical range assignment ρ_k for $k = (2^\alpha/\varepsilon)^{1/(\alpha-1)} = O((1/\varepsilon)^{1/(\alpha-1)})$ we obtain the following theorem.

► **Theorem 6.** *There is a SAS for the dynamic broadcast range-assignment problem in \mathbb{R}^1 with stability parameter $k(\varepsilon) = O((1/\varepsilon)^{1/(\alpha-1)})$, where $\alpha > 1$ is the distance-power gradient. The time needed by the SAS to compute the new range assignment upon the insertion or deletion of a point is $O(n \log n)$, where n is the number of points in the current set.*

Proof. Our SAS maintains the canonical range assignment ρ_k for $k = (2^\alpha/\varepsilon)^{1/(\alpha-1)} = O((1/\varepsilon)^{1/(\alpha-1)})$. We then have $\text{cost}_\alpha(\rho_k(P)) \leq (1 + \varepsilon) \cdot \rho_{\text{opt}}(P)$ by Lemma 5. Furthermore, the number of modified ranges when P is updated is $2k + 6$ by Lemma 4. To determine the assignment ρ_k , we need to know an optimal assignment ρ_{opt} with the structure from Theorem 2. Such an optimal assignment can be maintained in $O(n \log n)$ time per update, by Theorem 3. Once we have the new optimal assignment, the new optimal assignment can be determined in $O(n)$ time. ◀

Next we show that the stability parameter $k(\varepsilon)$ in our SAS is asymptotically optimal.

► **Theorem 7.** *Any SAS for the dynamic broadcast range-assignment problem in \mathbb{R}^1 must have stability parameter $k(\varepsilon) = \Omega((1/\varepsilon)^{1/(\alpha-1)})$, where $\alpha > 1$ is the distance-power gradient.*

Proof. Let ALG be a k -stable algorithm, where $k \geq 4$ and $k^{\alpha-1} \geq \frac{1}{2^{\alpha+1}(2^{\alpha-1}-1)}$ and k is even, and let ρ_{alg} be the range assignment it maintains. Note that the condition on k is satisfied for k large enough. We will show that the approximation ratio of ALG is at least $1 + \frac{1}{2^{\alpha+2}k^{\alpha-1}}$. Since a SAS has approximation ratio $1 + \varepsilon$, this implies that the stability parameter $k(\varepsilon)$ of ALG must satisfy $k(\varepsilon) = \Omega((1/\varepsilon)^{1/(\alpha-1)})$.

Consider the point set $P := \{s, r_1, r_2, \dots, r_{2k}\}$, where $s = 0$ and $r_i = i/(2k)$ for $i = 1, 2, \dots, 2k$. We consider two cases.

Case I: The number of zero-range points in $\rho_{\text{alg}}(P)$ is at least $k/2$, where we assume without loss of generality that all points with range less than $1/(2k)$ actually have range zero. The cheapest possible solution in this case is to have exactly $k/2$ zero-range points, k points with range $1/(2k)$, and $k/2$ points with range $1/k$, for a total cost of

$$\text{cost}_\alpha(\rho_{\text{alg}}(P)) \geq k \cdot \left(\frac{1}{2k}\right)^\alpha + \frac{k}{2} \cdot \left(\frac{1}{k}\right)^\alpha = \left(1 + \frac{2^{\alpha-1} - 1}{2}\right) \cdot 2k \left(\frac{1}{2k}\right)^\alpha.$$

An optimal solution has cost $2k \cdot (1/(2k))^\alpha$, and so the approximation ratio of ALG in Case I is at least $1 + \frac{2^{\alpha-1}-1}{2}$, which is at least $1 + \frac{1}{2^{\alpha+2}k^{\alpha-1}}$ since $k^{\alpha-1} \geq \frac{1}{2^{\alpha+1}(2^{\alpha-1}-1)}$.

Case II: The number of zero-range points $\rho_{\text{alg}}(P)$ is less than $k/2$. Now suppose the point $\ell_1 = -1$ arrives. Since $\rho_{\text{alg}}(P)$ had less than $k/2$ zero-range points and ALG can modify at most k ranges, $\rho_{\text{alg}}(P \cup \{\ell_1\})$ has less than $3k/2$ zero-range points. Hence, at least $k/2$ points in $P \cup \{\ell_1\}$ have a range that is at least $1/(2k)$, one of which must have a range at least 1 . This implies that $\text{cost}_\alpha(\rho_{\text{alg}}(P \cup \{\ell_1\})) \geq 1 + (k/2 - 1) \cdot \left(\frac{1}{2k}\right)^\alpha \geq 1 + \frac{1}{2^{\alpha+2}k^{\alpha-1}}$, where the last inequality holds since $k/2 - 1 \geq k/4$ (because $k \geq 4$). An optimal range assignment on $P \cup \{\ell_1\}$ has $\rho_{\text{opt}}(s) = 1$ and all other ranges equal to zero, for a total cost of 1, and so the approximation ratio of ALG in Case II is at least $1 + \frac{1}{2^{\alpha+2}k^{\alpha-1}}$ as well. ◀

■ **Table 1** An overview of the approximation ratio of 1-stable, 2-stable and 3-stable algorithms.

| ℓ -stable algorithm | Approximation Ratio | Remarks |
|--------------------------|-------------------------------|--------------------------------|
| $\ell = 1$ | $6 + 2\sqrt{5} \approx 10.47$ | $\alpha = 2$, insertions only |
| $\ell = 2$ | 2 | for any $\alpha > 1$ |
| $\ell = 3$ | 1.97 | $\alpha = 2$ |

4 1-Stable, 2-Stable, and 3-Stable Algorithms in \mathbb{R}^1

In Section 3 we presented a $(2k + 6)$ -stable algorithm with approximation ratio $1 + 2^\alpha/k^{\alpha-1}$, which provided us with a SAS. For small k the algorithm is not very good: the most stable algorithm we can get is 6-stable, by setting $k = 0$. A careful analysis shows that the approximation ratio of this 6-stable algorithm is 3, for $\alpha = 2$. Below we briefly discuss the results we obtained for more stable algorithms; details are available in the full version [14].

We give a 1-stable $O(1)$ -approximation algorithm; obviously, this is the best we can do in terms of stability. This algorithm can only handle insertions. We also show that this is necessarily the case: a 1-stable algorithm that can handle insertions as well as deletions cannot have bounded approximation ratio. We then present a straightforward 2-stable 2-approximation algorithm, which simply gives every point its standard range. Finally, we study 3-stable algorithms: we show that using a 3-stable algorithm it is possible to get an approximation ratio strictly below 2. See Table 1 for an overview of results. We now briefly sketch our 3-stable algorithm.

A 3-stable algorithm with approximation ratio less than 2. Given the simplicity of our 2-stable 2-approximation algorithm, it is surprisingly difficult to obtain an approximation ratio strictly smaller than 2. In fact, we have not been able to do this with a 2-stable algorithm. Below we show this is possible with a 3-stable algorithm, at least for the case $\alpha = 2$, which we assume from now on.

Recall that for any set P with points on both sides of the source point s , there is an optimal range assignment inducing a broadcast tree with a single root-crossing point; see Figure 1. Unfortunately the root-crossing point may change when P is updated. This may cause many changes if we maintain a solution with a good approximation ratio and the same root-crossing point as the optimal solution. We therefore restrict ourselves to *source-based range assignments*, where s is the root-crossing point. The main question is then how large the range of s should be, and which points within range of s should be zero-range points.

We now define our source-based range assignment, which we denote by ρ_{sb} , more precisely. It will be uniquely defined by the set P ; it does not depend on the order in which points have been inserted or deleted. Let δ be a parameter with $1/2 < \delta < 1$; later we will choose δ such that the approximation ratio of our algorithm is optimized. We call a point $p \in P \setminus \{s\}$ *expensive* if $\text{succ}(p) \neq \text{NIL}$ and $|p \text{ succ}(p)| > \delta \cdot |s \text{ succ}(p)|$, and we call it *cheap* otherwise. The source s is defined to be always expensive. (This is consistent in the sense that for $p = s$ the condition $|p \text{ succ}(p)| > \delta \cdot |s \text{ succ}(p)|$ holds for both successors, since $\delta < 1$.) We denote the set of all expensive points in P by P_{exp} and the set of all cheap points by P_{cheap} . Define $d_{\text{max}} := \max\{|s \text{ succ}(p)| : p \in P_{\text{exp}}\}$, that is, d_{max} is the maximum distance from s to the successor of any expensive point. We say that a point $p \in P_{\text{exp}}$ is *crucial* if $|s \text{ succ}(p)| = d_{\text{max}}$. Typically there is a single crucial point, but there can also be two: one on the left and one on the right of s . Our source-based range assignment ρ_{sb} is now defined as follows.

- $\rho_{\text{sb}}(s) := d_{\text{max}}$,
- $\rho_{\text{sb}}(p) := 0$ for all $p \in P_{\text{exp}} \setminus \{s\}$, and
- $\rho_{\text{sb}}(p) := \rho_{\text{st}}(p)$ for all $p \in P_{\text{cheap}}$, where $\rho_{\text{st}}(p)$ denotes the standard range of a point.

It is easily checked that we can maintain this range assignment with a 3-stable algorithm. The challenge is to analyze its approximation ratio. In the full version of the paper [14], we show that, for a suitable choice of δ , the approximation ratio is strictly smaller than 2, leading to the following theorem.

► **Theorem 8.** *There exists a 3-stable 1.97-approximation algorithm for the dynamic broadcast range-assignment problem in \mathbb{R}^1 for $\alpha = 2$.*

5 The problem in \mathbb{S}^1

We now turn to the setting where the underlying space is \mathbb{S}^1 , that is, the points in P lie on a circle and distances are measured along the circle. In Section 5.1, we prove that the structure of an optimal solution in \mathbb{S}^1 is very similar to the structure of an optimal solution in \mathbb{R}^1 as formulated in Theorem 2. In spite of this, and contrary to the problem in \mathbb{R}^1 , we prove in Section 5.2 that no SAS exists for the problem in \mathbb{S}^1 .

Again, we denote the source point by s . The clockwise distance from a point $p \in \mathbb{S}^1$ to a point $q \in \mathbb{S}^1$ is denoted by $d_{\text{cw}}(p, q)$, and the counterclockwise distance by $d_{\text{ccw}}(p, q)$. The actual distance is then $d(p, q) := \min(d_{\text{cw}}(p, q), d_{\text{ccw}}(p, q))$. The closed and open clockwise interval from p to q are denoted by $[p, q]^{\text{cw}}$ and $(p, q)^{\text{cw}}$, respectively.

5.1 The structure of an optimal solution in \mathbb{S}^1

Here we prove that the structure of an optimal solution in \mathbb{S}^1 is very similar to the structure of an optimal solution in \mathbb{R}^1 . The heart of this proof is the following lemma 9. Define the *covered region* of P with respect to a range assignment ρ , denoted by $\text{cov}(\rho, P)$, to be the set of all points $r \in \mathbb{S}^1$ such that there exists a point $p \in P$ with $\rho(p) \geq d(p, r)$.

► **Lemma 9.** *Let P be a point set in \mathbb{S}^1 with $|P| > 2$ and let ρ_{opt} be an optimal range assignment for P . Then there exists a point $r \in \mathbb{S}^1$ such that $r \notin \text{cov}(\rho_{\text{opt}}, P)$.*

Lemma 9 implies that an optimal solution for an instance in \mathbb{S}^1 corresponds to an optimal solution for an instance in \mathbb{R}^1 derived as follows. For a point $r \in \mathbb{S}^1$, define the mapping $\mu_r : P \rightarrow \mathbb{R}^1$ such that $\mu_r(s) := 0$, and $\mu_r(p) := d_{\text{cw}}(s, p)$ for all $p \in [s, r]^{\text{cw}}$, and $\mu_r(p) := -d_{\text{ccw}}(s, p)$ for all $p \in [r, s]^{\text{cw}}$. Let $\mu_r(P)$ denote the resulting point set in \mathbb{R}^1 .

► **Theorem 10.** *Let P be an instance of the broadcast range-assignment problem in \mathbb{S}^1 . There exists a point $r \in \mathbb{S}^1$ such that an optimal range assignment for $\mu_r(P)$ in \mathbb{R}^1 induces an optimal range assignment for P . Moreover, we can compute an optimal range assignment for P in $O(n^2 \log n)$ time, where n is the number of points in P .*

Proof. Let $r \in \mathbb{S}^1$ be a point such that $r \notin \text{cov}(\rho_{\text{opt}}, P)$, which exists by Lemma 9. Consider the mapping μ_r . Any feasible range assignment for $\mu_r(P)$ induces a feasible range assignment for P in \mathbb{S}^1 , since $d(p, q) \leq |\mu_r(p) - \mu_r(q)|$ for any two points $p, q \in P$. Conversely, an optimal range assignment for P induces a feasible range assignment for $\mu_r(P)$, since the point r is not covered in the optimal solution. This proves the first part of the theorem.

Now let $P := \{s, p_1, \dots, p_n\}$, where the points p_i are ordered clockwise from s . For $0 \leq i \leq n$, let r_i be a point in $(p_i, p_{i+1})^{\text{cw}}$, where $p_0 = p_{n+1} = s$. Since $\mu_{r_i} = \mu_r$ for any $r \in (p_i, p_{i+1})^{\text{cw}}$, an optimal solution can be computed by finding the best solution over all

15:12 Stable Approximation Algorithms for Range Assignment

mappings μ_{r_i} . The only difference between μ_{r_i} and $\mu_{r_{i+1}}$ is the location that p_{i+1} is mapped to, so after computing an optimal solution for $\mu_1(P)$ in $O(n^2 \log n)$ time, we can go through the mappings μ_2, \dots, μ_n and update the optimal solution in $O(n \log n)$ time using Theorem 3. Hence, an optimal range assignment for P can be computed in $O(n^2 \log n)$ time. ◀

Next we prove Lemma 9. Without loss of generality we identify \mathbb{S}^1 with a circle of perimeter 1. Let ρ_{opt} be a fixed optimal range-assignment on P . We will need the following lemma.

► **Lemma 11.** *If $|P| > 2$ then $\rho_{\text{opt}}(p) < \frac{1}{2}$ for all $p \in P$.*

Proof. Note that setting $\rho(s) = \frac{1}{2}$ and $\rho(p) = 0$ for all $p \in P \setminus \{s\}$ gives a feasible solution. Since $\rho(s) > 0$ in any feasible solution, this means that $\rho_{\text{opt}}(p) < \frac{1}{2}$ for all $p \neq s$. Hence, it suffices to show that $\rho_{\text{opt}}(s) < \frac{1}{2}$. If there is no point $p \in P$ which is diametrically opposite s then clearly $\rho_{\text{opt}}(s) < \frac{1}{2}$. Now suppose some point $p \in P$ lies diametrically opposite s . Let $q \in P \setminus \{s, p\}$ be a point that maximizes the distance from s among all points in $P \setminus \{s, p\}$. The point q exists since $|P| > 2$. Note that $d(s, q) + d(q, p) = \frac{1}{2}$. Hence, setting $\rho(s) = d(s, q)$ and $\rho(q) = d(q, p)$ (and keeping all other ranges zero) gives a solution of cost $d(s, q)^\alpha + d(q, p)^\alpha$, which is less than $(\frac{1}{2})^\alpha$ since $\alpha > 1$. Thus $\rho_{\text{opt}}(s) < \frac{1}{2}$, which finishes the proof. ◀

Before we proceed, we introduce some more notation.

For two points $p, q \in \mathbb{S}^1$, we let $[p, q]^{\text{cw}} \subset \mathbb{S}^1$ denote the closed clockwise interval from p to q . In other words, $[p, q]^{\text{cw}}$ is the clockwise arc along \mathbb{S}^1 from p to q , including its endpoints. Furthermore, we define $(p, q)^{\text{cw}}$ to be the open clockwise interval from p to q . The intervals $[p, q]^{\text{ccw}}$ and $(p, q)^{\text{ccw}}$ are defined similarly, but for the counterclockwise direction. Now consider a directed edge (p, q) in a communication graph $\mathcal{G}_\rho(P)$. We say that (p, q) is a *clockwise edge* if $\rho(p) \geq d_{\text{cw}}(p, q)$, and we say that it is a *counterclockwise edge* if $\rho(p) \geq d_{\text{ccw}}(p, q)$. Lemma 11 implies that an edge cannot be both clockwise and counterclockwise in an optimal range assignment, assuming $|P| > 2$. Finally, we define the *covered region* of a subset $Q \subseteq P$ with respect to a range assignment ρ to be the set of all points $r \in \mathbb{S}^1$ such that there exists a point $p \in Q$ such that $\rho(p) \geq d(p, r)$. We denote this region by $\text{cov}(\rho, Q)$. Furthermore, the *counterclockwise covered region* of Q , denoted by $\text{cov}_{\text{ccw}}(\rho, Q)$, is the set of all points $r \in \mathbb{S}^1$ such that there exists a point $p \in Q$ such that $\rho(p) \geq d_{\text{ccw}}(p, r)$. The *clockwise covered region* of Q , denoted by $\text{cov}_{\text{cw}}(\rho, Q)$, is defined similarly.

We can now state the main lemma of this section.

► **Lemma 9.** *Let P be a point set in \mathbb{S}^1 with $|P| > 2$ and let ρ_{opt} be an optimal range assignment for P . Then there exists a point $r \in \mathbb{S}^1$ such that $r \notin \text{cov}(\rho_{\text{opt}}, P)$.*

Proof. Let $d_{\text{hop}}(p, q)$ denote the hop distance from p to q in the communication graph $\mathcal{G}_{\rho_{\text{opt}}}(P)$. Let \mathcal{B} broadcast tree rooted at s in $\mathcal{G}_{\rho_{\text{opt}}}(P)$ with the following properties.

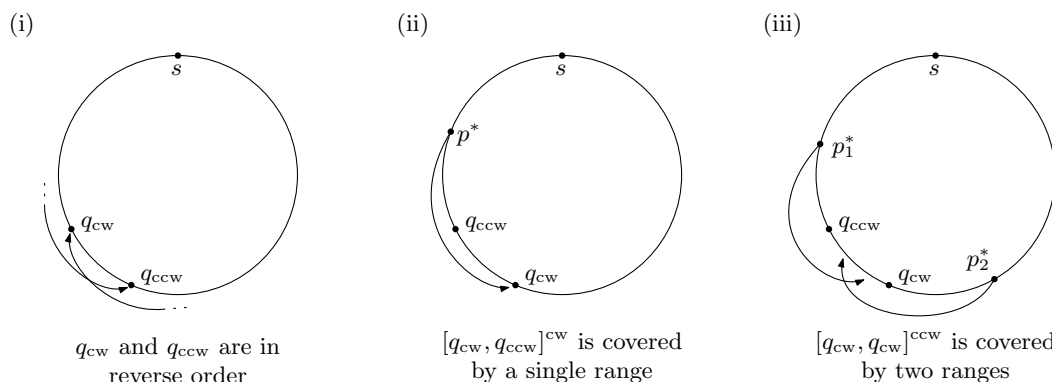
- \mathcal{B} is a shortest-path tree in terms of hop distance, that is, the hop-distance from s to any point p in \mathcal{B} is equal to $d_{\text{hop}}(s, p)$.
- Among all such shortest-path trees, \mathcal{B} maximizes the number of clockwise edges.

For two points $p, q \in P$, let $\pi(p, q)$ denote the path from p to q in \mathcal{B} , and let $|\pi(p, q)|$ be its length, that is, the number of edges on the path. Note that $|\pi(s, p)| = d_{\text{hop}}(s, p)$ for any $p \in P$. Let $\text{pa}(p)$ denote the parent of a point p in \mathcal{B} and define

$$S_{\text{cw}} = \{p \in P \setminus \{s\} : (\text{pa}(p), p) \text{ is a clockwise edge}\}$$

and

$$S_{\text{ccw}} = \{p \in P \setminus \{s\} : (\text{pa}(p), p) \text{ is a counterclockwise edge}\}.$$



■ **Figure 3** Illustration for the proof of Lemma 9. Note that the point p^* in part (ii) of the figure could also lie in $[s, q_{cw}]^{cw}$. Similarly, in part (iii) the points p_1^* and p_2^* could lie on “the other side” of s .

Note that $S_{cw} \cup S_{ccw} = P \setminus \{s\}$. Now define

$$q_{cw} = \text{the point from } S_{cw} \text{ that maximizes } d_{cw}(s, p),$$

where $q_{ccw} = s$ if $S_{ccw} = \emptyset$. Similarly, define

$$q_{ccw} = \text{the point from } S_{ccw} \text{ that maximizes } d_{ccw}(s, p),$$

where $q_{ccw} = s$ if $S_{ccw} = \emptyset$. Let $\text{anc}(p)$ be the set of ancestors in \mathcal{B} of a point $p \in P$, that is, $\text{anc}(p)$ contains the points of $\pi(s, p)$ excluding the point p . The following observation will be used repeatedly in the proof.

▷ **Observation.** If $(\text{pa}(p), p)$ is a clockwise edge, then $[s, p]^{cw} \subset \text{cov}(\rho_{opt}, \text{anc}(p))$. Similarly, if $(\text{pa}(p), p)$ is a counterclockwise edge, then $[s, p]^{ccw} \subset \text{cov}(\rho_{opt}, \text{anc}(p))$.

Proof. Assume $(\text{pa}(p), p)$ is a clockwise edge; the proof for when $(\text{pa}(p), p)$ is a counterclockwise edge is similar. If $s \in [\text{pa}(p), p]^{cw}$ – this includes the case where $\text{pa}(p) = s$ – then the statement obviously holds, so assume $\text{pa}(p) \in [s, p]^{cw}$. Since $(\text{pa}(p), p)$ is a clockwise edge, it then suffices to prove that $[s, \text{pa}(p)]^{cw} \subset \text{cov}(\rho_{opt}, \text{anc}(p))$. Note that $\text{cov}(\rho_{opt}, \text{anc}(p))$ is connected, because the points in $\text{anc}(p)$ form a path, namely $\pi(s, \text{pa}(p))$. Since $\pi(s, p)$ is shortest path, $p \notin \text{cov}(\rho_{opt}, \text{anc}(\text{pa}(p)))$, which implies that $[s, \text{pa}(p)]^{cw} \subset \text{cov}(\rho_{opt}, \text{anc}(\text{pa}(p))) \subset \text{cov}(\rho_{opt}, \text{anc}(p))$. ◁

We now proceed to show that q_{ccw} must lie clockwise from q_{cw} , as seen from s , that is, the situation shown in Fig. 3(i) cannot happen.

▷ **Claim.** $d_{cw}(s, q_{cw}) + d_{ccw}(s, q_{ccw}) < 1$.

Proof. Note that $d_{cw}(s, q_{cw}) + d_{ccw}(s, q_{ccw}) \neq 1$, since otherwise $q_{cw} = q_{ccw}$ which cannot happen since $S_{cw} \cap S_{ccw} = \emptyset$.

Now assume for a contradiction that $d_{cw}(s, q_{cw}) + d_{ccw}(s, q_{ccw}) > 1$, which means that $q_{ccw} \in [s, q_{cw}]^{cw}$. Since q_{cw} is reached from its parent by a clockwise edge, this implies that $q_{ccw} \in \text{cov}(\rho_{opt}, \text{anc}(q_{cw}))$ by the observation above. Hence, $d_{hop}(s, q_{cw}) \geq d_{hop}(s, q_{ccw})$. An analogous argument shows that $d_{hop}(s, q_{ccw}) \geq d_{hop}(s, q_{cw})$. Hence, $d_{hop}(s, q_{ccw}) = d_{hop}(s, q_{cw})$. This implies that the edge $(\text{pa}(q_{cw}), q_{cw})$ passes over q_{ccw} , otherwise some other edge of $\pi(s, q_{cw})$ would pass over q_{ccw} and we would have $d_{hop}(s, q_{ccw}) < d_{hop}(s, q_{cw})$. But then we also have a shortest path from s to q_{ccw} whose last edge is a clockwise edge, contradicting the definition of \mathcal{B} . ◁

So we can assume that $d_{cw}(s, q_{cw}) + d_{ccw}(s, q_{ccw}) < 1$ or, in other words, that q_{ccw} lies clockwise from q_{cw} , as seen from s . Clearly no point from P lies in $(q_{cw}, q_{ccw})^{cw}$. If we have $(q_{cw}, q_{ccw})^{cw} \not\subset \text{cov}(\rho_{opt}, P)$ then we are done, so assume for a contradiction that $(q_{cw}, q_{ccw})^{cw} \subset \text{cov}(\rho_{opt}, P)$. This can happen in three ways, each of which will lead to a contradiction.

Case I: There exists a point $p^* \in \mathcal{B}$ such that $q_{cw} \in \text{cov}_{ccw}(\rho_{opt}, \{p^*\})$.

See Fig. 3(ii) for an illustration of the situation. If $p^* = s$ then $d_{hop}(s, q_{cw}) = 1$. Since $q_{cw} \in S_{cw}$ this means that q_{cw} must also have an incoming clockwise edge from s . But then $\rho_{opt}(s) \geq \frac{1}{2}$, which contradicts Lemma 11. So $p^* \neq s$. Now note that p^* must have an outgoing clockwise edge in \mathcal{B} , else we can reduce the range of p^* to $d_{ccw}(p^*, q_{ccw})$, which is smaller than $d_{ccw}(p^*, q_{cw})$, and still get a feasible solution. Observe that $p^* \notin \pi(s, q_{cw})$; otherwise we must have $p^* = \text{pa}(q_{cw})$ (since q_{cw} lies in the range of p^*) which contradicts that $q_{cw} \in S_{cw}$. So for any point from P in the region $[s, q_{cw}]^{cw}$ there exists a path from s in the communication graph induced by ρ_{opt} that does not use p^* . We now have two subcases.

If $p^* \in [s, q_{ccw}]^{ccw}$ then clearly $p^* \in S_{ccw}$ (otherwise the definition of q_{cw} is contradicted). Hence, each point from P in the region $[s, p^*]^{ccw}$ has a path from s that does not use p^* . This implies that can reduce the range of p^* to $d_{ccw}(p^*, q_{ccw})$ and still get a feasible solution.

If $p^* \in [s, q_{cw}]^{cw}$ then obviously we can also reduce the range of p^* to $d_{ccw}(p^*, q_{ccw})$ and still get a feasible solution.

So both subcases lead to the desired contradiction.

Case II: There exists a point $p^* \in \mathcal{B}$ such that $q_{ccw} \in \text{cov}_{cw}(\rho_{opt}, \{p^*\})$.

In the proof of Case I we never used that \mathcal{B} maximizes the number of clockwise edges. Hence, a symmetric argument shows that Case II also leads to a contradiction.

Case III: There are two points $p_1^*, p_2^* \in P$ such that $[q_{cw}, q_{ccw}]^{cw} \subseteq \text{cov}_{ccw}(\rho_{opt}, \{p_1^*\}) \cup \text{cov}_{cw}(\rho_{opt}, \{p_2^*\})$.

See Fig. 3(iii) for an illustration of the situation. We can assume that $q_{cw} \notin \text{cov}_{ccw}(\rho_{opt}, \{p_1^*\})$ and $q_{ccw} \notin \text{cov}_{cw}(\rho_{opt}, \{p_2^*\})$, otherwise we are in Case I or Case II. Now either $p_2^* \notin \pi(s, p_1^*)$ or $p_1^* \notin \pi(s, p_2^*)$ or both. Without loss of generality, assume $p_2^* \notin \pi(s, p_1^*)$. Then $p_2^* \neq s$ and all points from P in the region $[s, q_{ccw}]^{ccw}$ have a path from s in the communication graph $\mathcal{G}_{\rho_{opt}}(P)$ that does not use p_2^* . The point p_2^* must have an outgoing counterclockwise edge, else we can reduce the range of p_2^* to $d_{cw}(p_2^*, q_{cw})$ and still get a feasible solution. We have two subcases.

If $p_2^* \in [s, q_{ccw}]^{ccw}$ then by reducing the range of p_2^* to $d_{cw}(p_2^*, q_{cw})$ we still get a feasible solution.

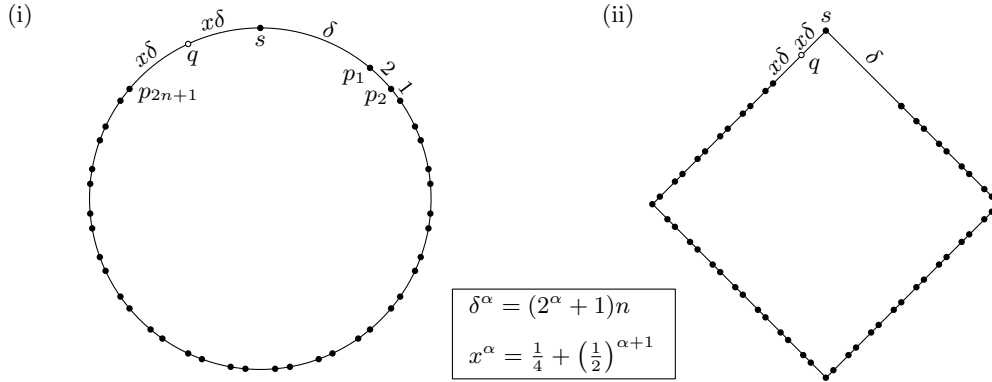
If $p_2^* \in [s, q_{cw}]^{cw}$ then p_2^* must be reached by a clockwise edge from its parent in \mathcal{B} , otherwise the definition of q_{ccw} would be contradicted. Hence, for each point from P in the region $[s, p_2^*]^{cw}$ there is a path from s that does not use p_2^* . So again we can reduce the range of p_2^* to $d_{cw}(p_2^*, q_{cw})$ we still get a feasible solution.

Thus both subcases lead to a contradiction.

This finishes the proof of the lemma. \blacktriangleleft

5.2 Non-existence of a SAS in \mathbb{S}^1

We have seen that an optimal solution for a set P in \mathbb{S}^1 can be obtained from an optimal solution in \mathbb{R}^1 , if we cut \mathbb{S}^1 at an appropriate point r . It is a fact however that the insertion of a new point into P may cause the location of the cutting point r to change drastically. Next we show that this means that the dynamic problem in \mathbb{S}^1 does not admit a SAS.



■ **Figure 4** (i) The instance showing that there is no SAS in \mathbb{S}^1 . (ii) The instance in \mathbb{R}^2 .

► **Theorem 12.** *The dynamic broadcast range-assignment problem in \mathbb{S}^1 with distance power gradient $\alpha > 1$ does not admit a SAS. In particular, there is a constant $c_\alpha > 1$ such that the following holds: for any n large enough, there is a set $P := \{s, p_1, \dots, p_{2n+1}\}$ and a point q in \mathbb{S}^1 such that any update algorithm ALG that maintains a c_α -approximation must modify more than $2n/3 - 1$ ranges upon the insertion of q into P .*

The rest of this section is dedicated to proving Theorem 12. We will prove the theorem for

$$c_\alpha := \min \left(1 + 2^{\alpha-4} - \frac{1}{8}, 1 + \frac{2^{\alpha-1} - 1}{3 \cdot 2^\alpha + 2}, 1 + \frac{\min(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3})}{4(2^\alpha + 1)} \right).$$

Note that each term is a constant strictly greater than 1 for any fixed constant $\alpha > 1$. In particular, for $\alpha = 2$ we have $c_\alpha = 1 + \frac{1}{14}$.

Let $P := \{s, p_1, \dots, p_{2n+1}\}$, where $d_{\text{cw}}(p_i, p_{i+1}) = 2$ for odd i and $d_{\text{cw}}(p_i, p_{i+1}) = 1$ for even i ; see Fig. 4(i). Let $d_{\text{cw}}(s, p_1) = \delta$, where $\delta^\alpha = (2^\alpha + 1)n$. Finally, let $d_{\text{cw}}(p_{2n+1}, q) = d_{\text{cw}}(q, s) = x\delta$, where $x^\alpha = \frac{1}{4} + \left(\frac{1}{2}\right)^{\alpha+1}$. Note that $(1/2)^\alpha < x^\alpha < 1/2$ for any $\alpha > 1$.

Let $\rho(p)$ denote the range given to a point p by ALG. A directed edge (p, p') in the communication graph induced by ρ is called a *clockwise edge* if $\rho(p) \geq d_{\text{cw}}(p, p')$, and it is called a *counterclockwise edge* if $\rho(p) \geq d_{\text{ccw}}(p, p')$. Observe that we may assume that no edge (p, p') is both clockwise and counterclockwise, because otherwise $\rho(p) \geq (\delta + 3n + 2x\delta)/2$, which is much too expensive for an approximation ratio of at most c_α . Define the range $\rho(p)$ of a point in P to be *CW-minimal* if $\rho(p)$ equals the distance from p to its clockwise neighbor in P . Similarly, $\rho(p)$ is *CCW-minimal* if $\rho(p)$ equals the distance from p to its counterclockwise neighbor. The idea of the proof is to show that before the insertion of q , most of the points s, p_1, \dots, p_{2n+1} must have a CW-minimal range, while after the insertion most points must have a CCW-minimal range. This will imply that many ranges must be modified from being CW-minimal to being CCW-minimal.

Before the insertion of q , giving every point a CW-minimal range leads to a feasible assignment of total cost $\delta^\alpha + (2^\alpha + 1)n = 2\delta^\alpha$. After the insertion of q , giving every point a CCW-minimal range leads to a feasible assignment of total cost $2(x\delta)^\alpha + (2^\alpha + 1)n = (2x^\alpha + 1)\delta^\alpha$. Hence, if $\text{OPT}(\cdot)$ denotes the cost of an optimal range assignment, then we have:

► **Observation 13.** $\text{OPT}(P) \leq 2\delta^\alpha$ and $\text{OPT}(P \cup \{q\}) \leq (2x^\alpha + 1)\delta^\alpha < 2\delta^\alpha$.

15:16 Stable Approximation Algorithms for Range Assignment

We first prove a lower bound on the total cost of the points p_1, \dots, p_{2n+1} . Intuitively, only $o(n)$ of those points can be reached from s or q (otherwise the range of s or q would be too expensive) and the cheapest way to reach the remaining points will be to use only CW-minimal or CCW-minimal ranges.

► **Lemma 14.** $\sum_{i=1}^{2n+1} \rho(p_i)^\alpha \geq (2^\alpha + 1)n - o(n)$, both before and after the insertion of q .

Proof. By Observation 13, we have $\rho(p)^\alpha \leq c_\alpha \cdot 2\delta^\alpha$ and, hence, $\rho(p) \leq (2c_\alpha)^{1/\alpha} \cdot \delta < 3\delta$, for any point p . Consider the interval $I = [y_1, y_2]^{\text{cw}}$ where $d_{\text{cw}}(s, y_1) = 3\delta$ and $d_{\text{ccw}}(q, y_2) = 3\delta$. All the points in $I \cap P$ are at distance more than 3δ from s or q and hence $I \cap P \subseteq \text{cov}(\rho_{\text{opt}}, P \setminus \{s, q\})$. Let $p_i \in I \cap P$ be the point whose clockwise distance from s is minimum, and let $p_j \in I \cap P$ be the point whose counterclockwise distance from q is minimum. Then the cost of covering all the points in $I \cap P$ using the points in $P \setminus \{s, q\}$ is at least $\sum_{t=i}^{j-1} d_{\text{cw}}(p_t, p_{t+1})^\alpha - 2^\alpha$, where the term -2^α is because the covered region may leave one interval $[p_t, p_{t+1}]^{\text{cw}}$ uncovered. Recall that the cost of assigning all the points in $P \setminus \{s, q\}$ a CW-minimal range is $(2^\alpha + 1)n$. Note that $i = O(\delta)$ since $d_{\text{cw}}(s, p_i) \leq 3\delta + 2$ and $(2n + 1) - j = O(\delta)$ since $d_{\text{cw}}(p_j, q) \leq 3\delta + 2$. Hence,

$$\sum_{i=1}^{2n+1} \rho(p_i)^\alpha \geq (2^\alpha + 1)n - O(\delta) \cdot 2^\alpha \geq (2^\alpha + 1)n - o(n),$$

since $\delta = ((2^\alpha + 1)n)^{1/\alpha} = o(n)$. ◀

The following lemma gives a key property of the construction.

► **Lemma 15.** *The point p_{2n+1} cannot have an incoming counterclockwise edge before q is inserted, and the point p_1 cannot have an incoming clockwise edge after q has been inserted.*

Proof. Suppose before insertion of q the point p_{2n+1} has an incoming counterclockwise edge. The cheapest incoming counterclockwise edge would be from s and this is already too expensive. Indeed, if $\rho(s) \geq 2x\delta$ then by Lemma 14 the total cost of the range assignment by ALG is at least

$$\begin{aligned} (2x\delta)^\alpha + (2^\alpha + 1)n - o(n) &= \left(2^\alpha \cdot \left(\frac{1}{4} + \left(\frac{1}{2} \right)^{\alpha+1} \right) + 1 \right) \cdot \delta^\alpha - o(n) \\ &= \left(1 + \left(2^{\alpha-3} - \frac{1}{4} \right) \right) \cdot 2\delta^\alpha - o(n) \\ &> \left(1 + \frac{1}{2} \cdot \left(2^{\alpha-3} - \frac{1}{4} \right) \right) \cdot 2\delta^\alpha \quad \text{for } n \text{ sufficiently large} \\ &\geq c_\alpha \cdot \text{OPT}(P) \quad \text{by definition of } c_\alpha \text{ and Observation 13.} \end{aligned}$$

This contradicts the approximation ratio of ALG, proving the first part of the lemma.

Now suppose after the insertion of q the point p_1 has an incoming clockwise edge. The cheapest way to achieve this is with $\rho(s) = \delta$, which is too expensive. Indeed, by Lemma 14 the total cost of the range assignment is then at least

$$\begin{aligned}
\delta^\alpha + (2^\alpha + 1)n - o(n) &= \frac{2\delta^\alpha}{(2x^\alpha + 1)\delta^\alpha} \cdot (2x^\alpha + 1)\delta^\alpha - o(n) \\
&\geq \left(1 + \frac{1}{2} \cdot \left(\frac{2\delta^\alpha}{(2x^\alpha + 1)\delta^\alpha} - 1\right)\right) \cdot \text{OPT}(P \cup \{q\}) \quad \text{for } n \text{ sufficiently large} \\
&= \left(1 + \frac{2 - (2x^\alpha + 1)}{2(2x^\alpha + 1)}\right) \cdot \text{OPT}(P \cup \{q\}) \\
&= \left(1 + \frac{1 - (\frac{1}{2} + \frac{1}{2^\alpha})}{2(\frac{1}{2} + \frac{1}{2^\alpha} + 1)}\right) \cdot \text{OPT}(P \cup \{q\}) \quad \text{since } 2x^\alpha = \frac{1}{2} + \frac{1}{2^\alpha} \\
&= \left(1 + \frac{2^{\alpha-1} - 1}{3 \cdot 2^\alpha + 2}\right) \cdot \text{OPT}(P \cup \{q\}) \\
&\geq c_\alpha \cdot \text{OPT}(P \cup \{q\}) \quad \text{by definition of } c_\alpha \text{ and Observation 13.}
\end{aligned}$$

This contradicts the approximation ratio of ALG, proving the second part of the lemma. ◀

We are now ready to prove that many edges must change from being CW-minimal to being CCW-minimal when q is inserted. Observe that before and after the insertion of a point q , the distance between any two points is either 1, 2 or at least 3. Hence, in the following lemma we may assume that $\rho(p) \in \{0, 1, 2\} \cup [3, \infty)$ for any point $p \in P \cup \{q\}$.

► **Lemma 16.** *Before the insertion of q , at least $4n/3 + 1$ of the points from $\{s, p_1, \dots, p_{2n}\}$ have a CW-minimal range and after the insertion of q at least $4n/3 + 1$ of the points from $\{q, p_1, \dots, p_{2n}\}$ have a CCW-minimal range.*

Proof. We prove the lemma for the situation before q is inserted; the proof for the situation after the insertion of q is similar. It will be convenient to define $p_0 := s$ (although we may still use s if we want to stress that we are talking about the source). Recall that p_{2n+1} does not have an incoming counterclockwise edge in the communication graph $\mathcal{G}_\rho(P)$ before the insertion of q . Let π^* be a minimum-hop path from s to p_{2n+1} in $\mathcal{G}_\rho(P)$. Since p_{2n+1} does not have an incoming counterclockwise edge and π^* is a minimum-hop path, all edges in π^* are clockwise. We assign each point p_j with $1 \leq j \leq 2n + 1$ to the edge (p_i, p_t) in π^* such that $i + 1 \leq j \leq t$, and we define $A(p_i, p_t) := \{p_{i+1}, \dots, p_t\}$ to be the set of all points assigned to (p_i, p_t) . We define the *excess* of a point $p_j \in A(p_i, p_t)$ to be

$$\text{excess}(p_j) := \frac{1}{|A(p_i, p_t)|} \cdot \left(\rho(p_i)^\alpha - \sum_{p_\ell \in A(p_i, p_t)} d(p_{\ell-1}, p_\ell)^\alpha \right).$$

We say that an edge (p_i, p_t) in π^* is CW-minimal if p_i has a CW-minimal range. Note that if a point p_j is assigned to a CW-minimal edge, then this is the edge (p_{j-1}, p_j) and $\text{excess}(p_j) = 0$. Intuitively, $\text{excess}(p_j)$ denotes the additional cost we pay for reaching p_j compared to reaching it by a CW-minimal edge, if we distribute the additional cost of a non-CW-minimal edge over the points assigned to it. Because each of the points p_1, \dots, p_{2n+1} is assigned to exactly one edge on the path π^* , we have

$$\sum_{p_i \in \pi^*} \rho(p_i)^\alpha \geq \sum_{j=1}^{2n+1} d(p_{j-1}, p_j)^\alpha + \sum_{j=1}^{2n+1} \text{excess}(p_j) \geq \text{OPT}(P) + \sum_{j=1}^{2n+1} \text{excess}(p_j) \quad (1)$$

where the second inequality follows from Observation 13 and because $p_0 = s$. The following claim is proved in the full version. (Essentially, the smallest possible excess is obtained when $|A(p_i, p_t)| \in \{1, 2, 3\}$; the three terms in the claim correspond to these cases.)

15:18 Stable Approximation Algorithms for Range Assignment

▷ **Claim.** If p_j is not assigned to a CW-minimal edge then $\text{excess}(p_j) \geq c'_\alpha$, where $c'_\alpha = \min(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3})$.

Now suppose for a contradiction that less than $4n/3 + 1$ points from $\{s, p_1, \dots, p_{2n+1}\}$ have a CW-minimal range. Then at least $2n/3 + 1$ points p_j have $\text{excess}(p_j) \geq c'_\alpha$ by the claim above. By Inequality (1) the total cost incurred by ALG is therefore more than

$$\text{OPT}(P) + c'_\alpha \cdot (2n/3) = \text{OPT}(P) + \frac{c'_\alpha}{3(2^\alpha + 1)} \cdot 2(2^\alpha + 1)n \quad (2)$$

$$> \left(1 + \frac{\min(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3})}{4(2^\alpha + 1)}\right) \cdot \text{OPT}(P) \quad (3)$$

$$\geq c_\alpha \cdot \text{OPT}(P) \quad (4)$$

which contradicts the approximation ratio achieved by ALG. ◀

Lemma 16 implies that at least $4n/3$ of the points p_1, \dots, p_{2n+1} have a CW-minimal range before q is inserted, and at least $4n/3$ of those points have a CCW-minimal range after the insertion. Hence, at least $2n + 1 - 2 \cdot (2n/3 + 1) = 2n/3 - 1$ points must change from being CW-minimal to being CCW-minimal, thus finishing the proof of Theorem 12.

The claim in the proof of Lemma 16

▷ **Claim.** If p_j is not assigned to a CW-minimal edge then $\text{excess}(p_j) \geq c'_\alpha$, where $c'_\alpha = \min(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3})$.

Proof. Consider a non-CW-minimal edge (p_i, p_t) . First suppose only a single point p_j is assigned to (p_i, p_t) . Then $t = i + 1$ and $p_j = p_t$. Hence, $\rho(p_i) \geq d(p_{j-1}, p_j) + 1$ because we assumed $\rho(p_i) \in \{0, 1, 2\} \cup [3, \infty)$. Thus when $|A(p_i, p_t)| = 1$ then

$$\text{excess}(p_j) \geq (d(p_{j-1}, p_j) + 1)^\alpha - d(p_{j-1}, p_j)^\alpha \geq 2^\alpha - 1 \geq c'_\alpha.$$

Now suppose $|A(p_i, p_t)| > 1$. Let z_1 be the number of points $p_j \in A(p_i, p_t)$ with $d(p_{j-1}, p_j) = 1$, and let z_2 be the number of points $p_j \in A(p_i, p_t)$ with $d(p_{j-1}, p_j) = 2$. Since $|A(p_i, p_t)| > 1$ we have $z_1 \geq 1$ and $z_2 \geq 1$ and $|z_1 - z_2| \leq 1$. When $|A(p_i, p_t)| = 2$ then $z_1 = z_2 = 1$, and we are distributing the cost of an edge of length at least 3, minus the costs of edges of length 2 and 1, over two points. Thus in this case we have

$$\text{excess}(p_j) \geq \frac{3^\alpha - 2^\alpha - 1}{2}.$$

Similarly, when $|A(p_i, p_t)| = 3$ then $z_1 = 2$ and $z_2 = 1$ (or vice versa, but that will only lead to a larger excess), and we have

$$\text{excess}(p_j) \geq \frac{4^\alpha - 2^\alpha - 2}{3}.$$

It remains to argue that we do not get a smaller excess when $|A(p_i, p_t)| \geq 4$. To see this, we compare the excess we get when (p_i, p_t) is an edge of π with the excesses we would get when, instead of (p_i, p_t) , the edges (p_i, p_{i+2}) and (p_{i+2}, p_t) would be in π^* . Note that

$$d(p_i, p_t)^\alpha = \left(d(p_i, p_{i+2}) + d(p_{i+2}, p_t)\right)^\alpha > d(p_i, p_{i+2})^\alpha + d(p_{i+2}, p_t)^\alpha$$

since $\alpha > 1$. Hence,

$$\begin{aligned}
& \frac{d(p_i, p_t)^\alpha - \sum_{\ell=i+1}^t d(p_{\ell-1}, p_\ell)^\alpha}{t-i} \\
& > \frac{\left(d(p_i, p_{i+2})^\alpha - \sum_{\ell=i+1}^{i+2} d(p_{\ell-1}, p_\ell)^\alpha\right) + \left(d(p_{i+2}, p_t)^\alpha - \sum_{\ell=i+3}^t d(p_{\ell-1}, p_\ell)^\alpha\right)}{t-i} \\
& \geq \frac{d(p_i, p_{i+2})^\alpha - \sum_{\ell=i+1}^{i+2} d(p_{\ell-1}, p_\ell)^\alpha}{2} + \frac{d(p_{i+2}, p_t)^\alpha - \sum_{\ell=i+3}^t d(p_{\ell-1}, p_\ell)^\alpha}{t-i-2}
\end{aligned}$$

where the last inequality uses that $\frac{a_1+a_2}{b_1+b_2} \geq \min\left(\frac{a_1}{b_1}, \frac{a_2}{b_2}\right)$ for any $a_1, a_2, b_1, b_2 > 0$. Thus the excess we get for (p_i, p_t) is at least the minimum of the excesses we would get for (p_i, p_{i+2}) and (p_{i+2}, p_t) . More generally, when $|A(p_i, p_t)| > 4$ then we can compare the excess for (p_i, p_t) with the excesses we get when we would replace (p_i, p_t) with a path of smaller edges, each being assigned two or three points. The excess for (p_i, p_{i+2}) is at least the minimum of the excesses for these shorter edges. (Reducing to edges that are assigned a single point is not useful, since these may be CW-minimal and have zero excess.) This finishes the proof of the claim. \triangleleft

6 The 2-dimensional problem

The broadcast range-assignment problem is NP-hard in \mathbb{R}^2 , so we cannot expect a characterization of the structure of an optimal solution similar to Theorem 2. Using a similar construction as in \mathbb{S}^1 we can also show that the problem in \mathbb{R}^2 does not admit a SAS.

► **Theorem 17.** *The dynamic broadcast range-assignment problem in \mathbb{R}^2 with distance power gradient $\alpha > 1$ does not admit a SAS. In particular, there is a constant $c_\alpha > 1$ such that the following holds: for any n large enough, there is a set $P := \{s, p_1, \dots, p_{2n+1}\}$ and a point q in \mathbb{R}^2 such that any update algorithm ALG that maintains a c_α -approximation must modify at least $2n/3 - 1$ ranges upon the insertion of q into P .*

Proof. We use the same construction as in \mathbb{S}^1 , where we embed the points on a square and the distances used to define the instance are measured along the square; see Fig. 4(ii). We now discuss the changes needed in the proof to deal with the fact that distances in \mathbb{R}^2 between points from $P \cup \{q\}$ may be smaller than when measured along the square. With a slight abuse of terminology, we will still refer to an edge (p, p') that was clockwise in \mathbb{S}^1 as a clockwise edge, and similarly for counterclockwise edges.

Note that Observation 13 still holds. Now consider Lemma 14. The proof used that the points p_i at distance more than 3δ from s or q must be covered by the ranges of the points p_1, \dots, p_{2n+1} . We now restrict our attention to the points that are also at distance more than 3δ from a corner of the square. Each such point p_i must be covered by the range of some point p_j on the same edge of the square. Hence, the distance in \mathbb{R}^2 of from p_j to p_i is the same as the distance in \mathbb{S}^1 , so we can use the same reasoning as before. Thus the exclusion of the points that are at distance at most 3δ from a corner of the square only influences the constant in the $o(n)$ term in the lemma. Hence, Lemma 14 still holds.

The proof of Lemma 15 still holds, since the cheapest counterclockwise edge to p_{2n+1} before the insertion of q is still from s (and the distance from s to p_{2n+1} did not change), and the cheapest clockwise edge to p_1 after the insertion of q is still from s (and the distance from s to p_1 did not change).

It remains to check Lemma 16. The proof still holds, except that the claim that $\text{excess}(p_j) \geq c'_\alpha$ may not be true for the given value of c'_α when p_j is near a corner of the square, because the distances between points on different edges of the square do not correspond to the distances in \mathbb{S}^1 . To deal with this, we simply ignore the excess of any point within distance 3δ from a corner. This reduces the total excess by $o(n)$. It is easily verified that this does not invalidate the rest of the proof: we have to subtract $o(n)$ from the formulae in Equality (2), but this is still larger than $c_\alpha \cdot \text{OPT}(P)$.

We conclude that all lemmas still hold, which proves Theorem 17. \blacktriangleleft

Although the problem in \mathbb{R}^2 does not admit a SAS, there is a relatively simple $O(1)$ -stable $O(1)$ -approximation algorithm for $\alpha \geq 2$. The algorithm is based on a result by Ambühl [1], who showed that a minimum spanning tree (MST) on P gives a 6-approximation for the static broadcast range-assignment problem: turn the MST into a directed tree rooted at the source s , and assign as a range to each point $p \in P$ the maximum length of any of its outgoing edges. To make this stable, we set the range of each point to the maximum length of any of its incident edges (not just the outgoing ones). Because an MST in \mathbb{R}^2 has maximum degree 6, this leads to 17-stable 12-approximation algorithm; see the full version [14].

7 Concluding remarks

We studied the dynamic broadcast range-assignment problem from a stability perspective, introducing the notions of k -stable algorithms and stable approximation schemes (SASs). Our results provide a fairly complete picture of the problem in \mathbb{R}^1 , in \mathbb{S}^1 , and in \mathbb{R}^2 . In particular, we presented a SAS in \mathbb{R}^1 that has an asymptotically optimal stability parameter, and showed that the problem does not admit a SAS in \mathbb{S}^1 and \mathbb{R}^2 . Future work can focus on improving the (the upper and/or lower bounds for) approximation ratios that we have obtained for algorithms with constant stability parameter.

References

- 1 Christoph Ambühl. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1139–1150, 2005. doi:10.1007/11523468_92.
- 2 Mohammad R. Ataei, Amir H. Banihashemi, and Thomas Kunz. Low-complexity energy-efficient broadcasting in one-dimensional wireless networks. *IEEE Trans. Veh. Technol.*, 61(7):3276–3282, 2012. doi:10.1109/TVT.2012.2204077.
- 3 Mostafa A. Bassiouni and Chun-Chin Fang. Dynamic channel allocation for linear macrocellular topology. In *Proc. 1999 ACM Symposium on Applied Computing*, pages 382–388, 1999. doi:10.1145/298151.298391.
- 4 Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. New results for energy-efficient broadcasting in wireless networks. In *Proc. 13th International Symposium on Algorithms and Computation (ISAAC 2002)*, volume 2518 of *Lecture Notes in Computer Science*, pages 332–343, 2002. doi:10.1007/3-540-36136-7_30.
- 5 Andrea E. F. Clementi, Pierluigi Crescenzi, Paolo Penna, Gianluca Rossi, and Paola Vocca. On the complexity of computing minimum energy consumption broadcast subgraphs. In *Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, volume 2010 of *Lecture Notes in Computer Science*, pages 121–131. Springer, 2001. doi:10.1007/3-540-44693-1_11.
- 6 Andrea E. F. Clementi, Gurvan Huiban, Paolo Penna, Gianluca Rossi, and Yann C. Verhoeven. Some recent theoretical advances and open questions on energy consumption in ad-hoc wireless networks. In *Proc. 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE 2002)*, 2002.

- 7 Andrea E. F. Clementi, Miriam Di Ianni, and Riccardo Silvestri. The minimum broadcast range assignment problem on linear multi-hop wireless networks. *Theor. Comput. Sci.*, 299(1-3):751–761, 2003. doi:10.1016/S0304-3975(02)00538-8.
- 8 Andrea E. F. Clementi, Paolo Penna, Afonso Ferreira, Stephane Perennes, and Riccardo Silvestri. The minimum range assignment problem on linear radio networks. *Algorithmica*, 35(2):95–110, 2003. doi:10.1007/s00453-002-0985-2.
- 9 Andrea E. F. Clementi, Paolo Penna, and Riccardo Silvestri. Hardness results for the power range assignment problem in packet radio networks. In *Proc. 3rd International Workshop on Randomization and Approximation Techniques in Computer Science, and 2nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (RANDOM-APPROX'99)*, volume 1671 of *Lecture Notes in Computer Science*, pages 197–208, 1999. doi:10.1007/978-3-540-48413-4_21.
- 10 Andrea E. F. Clementi, Paolo Penna, and Riccardo Silvestri. The power range assignment problem in radio networks on the plane. In *Proc. 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *Lecture Notes in Computer Science*, pages 651–660, 2000. doi:10.1007/3-540-46541-3_54.
- 11 Gautam K. Das, Sandip Das, and Subhas C. Nandy. Range assignment for energy efficient broadcasting in linear radio networks. *Theor. Comput. Sci.*, 352(1-3):332–341, 2006. doi:10.1016/j.tcs.2005.11.046.
- 12 Gautam K. Das and Subhas C. Nandy. Weighted broadcast in linear radio networks. *Inf. Process. Lett.*, 106(4):136–143, 2008. doi:10.1016/j.ipl.2007.10.016.
- 13 Mark de Berg, Aleksandar Markovic, and Seeun William Umboh. The online broadcast range-assignment problem. In *Proc. 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181 of *LIPICs*, pages 60:1–60:15, 2020. doi:10.4230/LIPICs.ISAAC.2020.60.
- 14 Mark de Berg, Arpan Sadhukhan, and Frits C. R. Spieksma. Stable approximation algorithms for the dynamic broadcast range-assignment problem. *CoRR*, abs/2112.05426, 2021. arXiv:2112.05426.
- 15 Bernhard Fuchs. On the hardness of range assignment problems. *Networks*, 52(4):183–195, 2008. doi:10.1002/net.20227.
- 16 Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. Power consumption in packet radio networks. *Theor. Comput. Sci.*, 243(1-2):289–305, 2000. doi:10.1016/S0304-3975(98)00223-0.
- 17 Rudolf Mathar and Jürgen Matfeldt. Optimal transmission ranges for mobile communication in linear multihop packet radio networks. *Wireless Networks*, 2(4):329–342, 1996. doi:10.1007/BF01262051.
- 18 Kaveh Pahlavan and Allen H. Levesque. *Wireless information networks, Second Edition*. Wiley series in telecommunications and signal processing. Wiley-VCH, 2005.
- 19 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009. doi:10.1287/moor.1090.0381.
- 20 Martin Skutella and José Verschae. A robust PTAS for machine covering and packing. In *Proc. 18th Annual European Symposium (ESA 201)*, volume 6346 of *Lecture Notes in Computer Science*, pages 36–47, 2010. doi:10.1007/978-3-642-15775-2_4.

Well-Separation and Hyperplane Transversals in High Dimensions

Helena Bergold ✉

Institut für Informatik, Freie Universität Berlin, Germany

Daniel Bertschinger ✉

Department of Computer Science, ETH Zürich, Switzerland

Nicolas Grelier ✉

Department of Computer Science, ETH Zürich, Switzerland

Wolfgang Mulzer ✉ 

Institut für Informatik, Freie Universität Berlin, Germany

Patrick Schneider ✉

Department of Mathematical Sciences, University of Copenhagen, Denmark

Abstract

A family of k point sets in d dimensions is *well-separated* if the convex hulls of any two disjoint subfamilies can be separated by a hyperplane. Well-separation is a strong assumption that allows us to conclude that certain kinds of generalized ham-sandwich cuts for the point sets exist. But how hard is it to check if a given family of high-dimensional point sets has this property? Starting from this question, we study several algorithmic aspects of the existence of transversals and separations in high-dimensions.

First, we give an explicit proof that k point sets are well-separated if and only if their convex hulls admit no $(k - 2)$ -transversal, i.e., if there exists no $(k - 2)$ -dimensional flat that intersects the convex hulls of all k sets. It follows that the task of checking well-separation lies in the complexity class coNP. Next, we show that it is NP-hard to decide whether there is a hyperplane-transversal (that is, a $(d - 1)$ -transversal) of a family of $d + 1$ line segments in \mathbb{R}^d , where d is part of the input. As a consequence, it follows that the general problem of testing well-separation is coNP-complete. Furthermore, we show that finding a hyperplane that maximizes the number of intersected sets is NP-hard, but allows for an $\Omega\left(\frac{\log k}{k \log \log k}\right)$ -approximation algorithm that is polynomial in d and k , when each set consists of a single point. When all point sets are finite, we show that checking whether there exists a $(k - 2)$ -transversal is in fact strongly NP-complete.

Finally, we take the viewpoint of parametrized complexity, using the dimension d as a parameter: given k convex sets in \mathbb{R}^d , checking whether there is a $(k - 2)$ -transversal is FPT with respect to d . On the other hand, for $k \geq d + 1$ finite point sets in \mathbb{R}^d , it turns out that checking whether there is a $(d - 1)$ -transversal is $W[1]$ -hard with respect to d .

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases hyperplane transversal, high-dimension, hardness

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.16

Funding *Helena Bergold*: Supported by the German Science Foundation within the research training group “Facets of Complexity” (GRK 2434).

Nicolas Grelier: Supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

Wolfgang Mulzer: Supported in part by ERC StG 757609 and by the German Research Foundation within the collaborative DACH project *Arrangements and Drawings* as DFG Project MU 3501/3-1.



© Helena Bergold, Daniel Bertschinger, Nicolas Grelier, Wolfgang Mulzer, and Patrick Schneider; licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 16; pp. 16:1–16:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the study of high-dimensional ham-sandwich cuts, the following notion has turned out to be fundamental: we call k sets S_1, \dots, S_k in \mathbb{R}^d are *well-separated* if for any *proper* index set $I \subset [k]$ (i.e., I is neither empty nor all of $[k]$), the convex hulls of $[k]$ $S_I = \cup_{i \in I} S_i$ and of $S_{[k] \setminus I} = \cup_{i \in [k] \setminus I} S_i$ can be separated by a hyperplane. Since any two disjoint convex sets can be separated by a hyperplane [16], well-separation is equivalent to the fact that for any proper index set I , the convex hulls of S_I and $S_{[k] \setminus I}$ do not intersect. A hyperplane h is a *transversal* of S_1, \dots, S_k if we have $S_i \cap h \neq \emptyset$, for all $i \in [k]$. More generally, for $m \in \{0, \dots, d-1\}$, an *m -transversal* of S_1, \dots, S_k is an m -flat (i.e., an m -dimensional affine subspace of \mathbb{R}^d) that intersects all the S_i . As we shall see below, it turns out that well-separation is intimately related to transversals: the sets S_1, \dots, S_k are well-separated if and only if there is no $(k-2)$ -transversal of the convex hulls of S_1, \dots, S_k .¹

In the past, transversals have been studied extensively, mostly from a combinatorial, but also from a computational perspective. Arguably the most well-known such theorem is *Helly's theorem* [12], which states that for any finite family of convex sets in \mathbb{R}^d , it holds that if every $d+1$ of them have a point in common, then all of them do. In other words, Helly's theorem gives a sufficient *fingerprint* condition for a family of convex sets to have a 0-transversal. In 1935, Vincensini asked whether such a statement holds for general k -transversals, that is, whether there is some number $m(k, d)$ such that if any $m(k, d)$ sets of a family have a k -transversal, then all of them do. This was disproved by Santaló, who showed that already the number $m(1, 2)$ does not exist (cf. [13] for more details).

One reason why 0-transversals differ significantly from k -transversals with $k > 0$ is that the space of 0-transversals of a family of convex sets is itself a convex set. In contrast, for $k > 0$, the space of k -transversals can be very complicated, even for pairwise disjoint convex sets. Thus, in order to generalize Helly's theorem to k -transversals with $k > 0$, additional assumptions become necessary. For example, Hadwiger's Transversal Theorem [11] states that for any family \mathcal{S} of compact and convex sets in the plane, it holds that if there exists a linear ordering on \mathcal{S} such that any three sets can be transversal by a directed line in accordance with this ordering, then there is a line transversal for \mathcal{S} . This result has been extended to higher dimensions by Pollack and Wenger [18]. Note that to have a well-defined order in which a directed line intersects the sets, the sets should be pairwise disjoint. Now, *well-separation* is a way to extend this idea to transversals of higher dimensions: if no $k+1$ sets in a family \mathcal{S} of convex sets have a $(k-1)$ -transversal, then every k -transversal H intersects the set \mathcal{S} in a well-defined k -ordering, that is, for every way of choosing a k -tuple of points from the intersections of H with \mathcal{S} , one point from each set, the orientation of the resulting simplices is the same (that is, they all have the same *order type*) [18]. Under well-separation, the space of transversals becomes simpler, in particular for hyperplane transversals: it is now a union of contractible sets [21]. Note that in d dimensions, there can be no $d+2$ sets that are well-separated, due to Radon's theorem which states that any set of $d+2$ points in d dimensions can be partitioned into two sets whose convex hulls intersect. For more background on transversals, we refer the interested readers to the relevant surveys, e.g., [2, 10, 13].

¹ Observe that for any $k \leq d$ sets in \mathbb{R}^d , there is always a $(k-1)$ -transversal: choose one point from each set, and consider a $(k-1)$ -flat that goes through these points. The $(k-1)$ -flat is unique if the chosen points are in general position.

Thus, well-separation is a strong assumption on set-families, and it does not come as a surprise that for many problems it leads to stronger results and faster algorithms compared to the general case. One such example is obtained for *Ham-Sandwich cuts*, a well-studied notion that occurs in many places in discrete geometry and topology [16]. Given d point sets P_1, \dots, P_d in \mathbb{R}^d , a Ham-Sandwich cut is a hyperplane that simultaneously bisects all point sets. While a Ham-Sandwich cut exists for any family of d point sets [20], finding such a cut is PPA-complete when the dimension is not fixed [9], meaning that it is unlikely that there is an algorithm that runs in polynomial time in the dimension d . On the other hand, if P_1, \dots, P_d are well-separated, not only do there exist bisecting hyperplanes, but the Ham-Sandwich theorem can be generalized to hyperplanes cutting off arbitrary given fractions from each point set [5, 19]. Further, the problem of finding such a hyperplane lies in the complexity class UEOPL [8], a subclass of PPA that is believed to be much smaller than PPA.

From an algorithmic perspective, the main focus of the previous work have been an efficient algorithms for finding line transversals in two and three dimensions, e.g., see [1, 4, 17]. To the authors' knowledge, in higher dimensions only algorithms for hyperplane transversals have been studied, where the best known algorithm for deciding whether a set of n polyhedra with m edges has a hyperplane transversal runs in time $O(nm^{d-1})$ [3]. In particular, there is an exponential dependence on the dimension d , and there are no non-trivial algorithmic results for the case that the dimension is part of the input. This curse of dimensionality appears in many geometric problems. For several problems, it has been shown that there is probably no hope to get rid of the exponential dependence in the dimension. As a relevant example, Knauer, Tiwary, and Werner [14] showed the following: given d point sets S_1, \dots, S_d in \mathbb{R}^d and a point $p \in \mathbb{R}^d$, where d is part of the input, it is $W[1]$ -hard (and thus NP-hard) to decide whether there is there a Ham-sandwich cut for the sets that passes through p .

Our Results. First, we prove that a family of k sets in \mathbb{R}^d is well-separated if and only if the convex hulls of the sets have no $(k - 2)$ -transversal. This fact seems to be known, but we could only find some references without proofs, and some proofs of only one direction, for similar definitions of well-separation [6, 7]. Therefore, for the sake of completeness, we present a short proof in Section 2. This immediately implies that testing well-separation is in coNP.

In [8], the authors ask for the complexity of determining whether a family of point sets is well-separated when d is not fixed. We present several hardness results for finding $(k - 2)$ -transversals in a family of k sets in \mathbb{R}^d . We consider two cases: a) finite sets, and b) possibly infinite, but convex set.

► **Theorem 1.** *Given a set of $k > d$ point sets in \mathbb{R}^d , each with at most two points, it is NP-hard to check whether there is a $(d - 1)$ -transversal, even in the special case $k = d + 1$.*

Note that this decision problem is trivial for $k \leq d$, as the answer is always yes. The assumption $k = d + 1$ is of special interest to us since the transversals we are considering are hyperplanes in \mathbb{R}^d , as in the Ham-sandwich cuts problem. Moreover, it shows that the problem becomes NP-hard for the first non-trivial value of k . We extend Theorem 1 to show the following:

► **Theorem 2.** *Given a set of $k > d$ line segments in \mathbb{R}^d , it is NP-hard to check whether there is a $(d - 1)$ -transversal, even in the special case $k = d + 1$.*

Theorem 2 implies that testing well-separation is coNP-complete even for $d + 1$ segments in \mathbb{R}^d , answering the question from [8]. Further, we show the following result, with a stronger hardness than Theorem 1; however, we remove the additional constraint that $k = d + 1$.

16:4 Well-Separation and Hyperplane Transversals in High Dimensions

► **Theorem 3.** *Given a set of $k \leq d + 1$ point sets in \mathbb{R}^d , each with most two points, it is strongly NP-hard to check whether there is a $(k - 2)$ -transversal.*

Observe that for the problem of Theorem 3, we consider $(k - 2)$ -transversals instead of $(d - 1)$ -transversals. In this context, the problem becomes trivial for $k \geq d + 2$, because all sets lie in \mathbb{R}^d . On the positive side, we can show the existence of the following approximation algorithm. This can be seen as the special case where each point set consists of a single point.

► **Theorem 4.** *Given a set P of k points in \mathbb{R}^d , it is possible to compute in polynomial time in d and k a hyperplane that contains $\Omega\left(\frac{\text{OPT} \log k}{k \log \log k}\right)$ points of P , where OPT denotes the maximum number of points in P that a hyperplane can contain.*

In Section 4, we study the problem through the lens of parametrized complexity. We show a significant difference between finite sets and convex sets.

► **Theorem 5.** *Checking whether a family of k convex sets in \mathbb{R}^d has a $(k - 2)$ -transversal (or equivalently, whether it is well-separated) is FPT with respect to d .*

► **Theorem 6.** *Checking whether a family of $k \geq d + 1$ finite point sets in \mathbb{R}^d has a $(d - 1)$ -transversal is $W[1]$ -hard with respect to d .*

Observe that for finite point sets (and, more generally, for any non-convex sets), having no $(k - 2)$ -transversal does not a priori imply well-separation. The result of Theorem 6 bears a similarity with the following result, shown in [14]: given a point set P in \mathbb{R}^d , is the origin contained in the affine hull of any d points? Indeed, in our reduction in the proof of Theorem 6, one of the point sets contains only the origin. However, our proof uses a radically different technique, as we have several point sets instead of one, and more importantly the number of points one can choose from is $k \leq d + 1$, whereas in the proof in [14] the set P contains fairly more than d points.

2 Well-separation and transversals

Let us recall some definitions. Let $S_1, \dots, S_k \subset \mathbb{R}^d$ be k sets in d dimensions. An m -transversal of S_1, \dots, S_k is an m -flat $h \subset \mathbb{R}^d$ (that is, an affine subspace of dimension m) with $h \cap S_i \neq \emptyset$ for $i = 1, \dots, k$. Transversals are intimately related to well-separation: the sets $S_1, \dots, S_k \subset \mathbb{R}^d$ are well-separated if and only if there is no $(k - 2)$ -transversal of their convex hulls. As mentioned in the introduction, this fact seems to be well known, but as we could not find a reference with all details for it, we give a short proof for the sake of completeness. In particular, a $(k - 2)$ -transversal of the convex hulls is a certificate that S_1, \dots, S_k are not well-separated. For a given $(k - 2)$ -flat h , it can be checked in polynomial time whether h is a $(k - 2)$ -transversal, yielding a proof that checking well-separation is in coNP.

► **Lemma 7.** *Let $S_1, \dots, S_k \subset \mathbb{R}^d$ be k sets in d dimensions. Then S_1, \dots, S_k are well-separated if and only if their convex hulls have no $(k - 2)$ -transversal.*

Proof. In the following, we assume without loss of generality that the sets are convex, that is, they are equal to their convex hulls. Assume first that S_1, \dots, S_k have a $(k - 2)$ -transversal h . Consider the intersection of the sets with h . This gives a collection of k sets S'_1, \dots, S'_k in a $(k - 2)$ -dimensional space, thus by Radon's theorem there is an index set $I \subset [k]$ such that the convex hulls of S'_I and of $S'_{[k] \setminus I}$ intersect. But then also the convex hulls of S_I and of $S_{[k] \setminus I}$ intersect, and thus S_1, \dots, S_k are not well-separated.

For the other direction, assume that S_1, \dots, S_k are not well-separated, that is, there is an index set $I \subset [k]$ such that the convex hulls of S_I and of $S_{[k] \setminus I}$ intersect. Let p be a point in this intersection. The point p can be written as a convex combination of points in S_I . Note that not only can we write it as a convex combination of points in S_I , but we can even ensure that in this combination, we use at most one point of each S_i , for $i \in I$. This is because the sets S_i are convex and so instead of taking two individual points we can take a convex combination of them. This means that in particular, there is a $(|I| - 1)$ -transversal h_I of S_I which contains p . The same holds for $S_{[k] \setminus I}$, giving a $(k - |I| - 1)$ -transversal $h_{[k] \setminus I}$ of $S_{[k] \setminus I}$ which contains p . Then the affine hull of h_I and $h_{[k] \setminus I}$ is a transversal of S_1, \dots, S_k and has dimension at most $|I| - 1 + k - |I| - 1 = k - 2$. ◀

3 Hyperplane Transversals in High Dimensions

Let $S_1, \dots, S_k \subset \mathbb{R}^d$ be k sets in d dimensions, where d is not fixed. Recall that a *hyperplane transversal* of S_1, \dots, S_k is a $(d - 1)$ -transversal. Note that we do not assume the sets to be convex. In particular, the sets can even be finite. We consider the decision problem HYPTRANS: Given sets S_1, \dots, S_k , decide if there is a hyperplane transversal for them. There are different variants of HYPTRANS, depending on what we require from the sets S_i . We consider the finite case and the case of line segments. We also consider the optimisation formulation of HYPTRANS, that we name MAXHYP: Given the sets S_1, \dots, S_k , find a hyperplane that intersects as many of these sets as possible.

3.1 Finite Case

We begin with the case that all S_i are finite point sets. We provide an approximation algorithm for MAXHYP in the situation where every S_i contains a single point, for $i = 1, \dots, k$. Note that in this situation, HYPTRANS can be solved greedily. We also provide some hardness results for HYPTRANS even in the restricted setting where every S_i contains at most two points, for $i = 1, \dots, k$.

3.1.1 Singleton sets

We assume that every S_i contains a single point, for $i = 1, \dots, k$. We denote by P the point set that is the union of all S_i . Let us denote by OPT the maximum number of points in P that a hyperplane may contain.

▶ **Theorem 8.** *It is possible to compute in polynomial time in d and k a hyperplane that contains $\Omega\left(\frac{OPT \log k}{k \log \log k}\right)$ points in P .*

Proof. If $k \leq d$, we just output a hyperplane that contains all points of P . Otherwise, let $f(k) = \log k / \log \log k$. If $f(k) < d$, we pick d points from P , and we output a hyperplane through these points. If $f(k) \geq d$, we partition P into disjoint groups of size $f(k)$. In each group, we compute all hyperplanes that go through some d points from the group. Among all hyperplanes for all groups, we output the hyperplane that contains the most points in P . For each group, we have $O(f(k)^d) = O(f(k)^{f(k)}) = O(k)$ hyperplanes to consider. Thus, the algorithm runs in polynomial time in d and k .

We now analyze the approximation guarantee. If $f(k) < d$, then we output a hyperplane with at least $d > f(k) \geq f(k)OPT/k$ points, since $OPT \leq k$. If $f(k) \geq d$, we let h be an optimal hyperplane. If h contains at least d points in a single group, then we output an optimal solution. Otherwise, h contains less than d points in each group, so $OPT \leq d(k/f(k))$. This means that $d \geq f(k)OPT/k$, and the claim follows from the fact that our solution contains at least d points. ◀

3.1.2 Sets of at most two points

Here, we restrict ourselves to the situation that every S_i contains at most two points, for $i = 1, \dots, k$. We prove that this version of HYPTRANS is NP-hard, with a reduction from SUBSETSUM. In SUBSETSUM, we are given $n + 1$ integers $a_1, \dots, a_n, b \in \mathbb{Z}$, and the goal is to decide whether there exists an index set $I \subseteq \{1, \dots, n\}$ with $\sum_{i \in I} a_i = b$. It is well-known that SUBSETSUM is (weakly) NP-complete.

Given an input $a_1, \dots, a_n, b \in \mathbb{Z}$ for SUBSETSUM, we create an input $S_1, \dots, S_{n+2} \subset \mathbb{R}^{n+1}$ for HYPTRANS, as follows. Note that the number of sets and the dimension are differing by exactly one. First, we define $2n + 1$ vectors $u, v_1, \dots, v_n, w_1, \dots, w_n \in \mathbb{R}^{n+1}$, by setting

$$\begin{aligned} u(1) &= -b \text{ and } u(j) = -1, & \text{for } j &= 2, \dots, n+1, \\ v_i(1) &= a_i \text{ and } v_i(j) = \delta_{i+1,j}, & \text{for } j &= 2, \dots, n+1, i = 1, \dots, n, \text{ and} \\ w_i(1) &= 0 \text{ and } w_i(j) = \delta_{i+1,j}, & \text{for } j &= 2, \dots, n+1, i = 1, \dots, n. \end{aligned}$$

Here, for $i, j \in \mathbb{Z}$,

$$\delta_{i,j} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases}$$

denotes the *Kronecker delta*. Using these vectors, we define the input for HYPTRANS as $S_1 = \{v_1, w_1\}, \dots, S_n = \{v_n, w_n\}, S_{n+1} = \{u\}$, and $S_{n+2} = \{\mathbf{0}\}$, where $\mathbf{0}$ is the origin of \mathbb{R}^{n+1} .

▷ **Claim 9.** We have that a_1, \dots, a_n, b is a yes-input for SUBSETSUM if and only if S_1, \dots, S_{n+2} is a yes-input for HYPTRANS.

Proof. First, suppose that a_1, \dots, a_n, b is a yes-input for SUBSETSUM, and let $I \subset [n]$ be an index set with $\sum_{i \in I} a_i = b$. Then, we define a point set x_1, \dots, x_{n+2} with $x_i \in S_i$ as follows: for $i = 1, \dots, n$, if $i \in I$, we set $x_i = v_i$, and if $i \notin I$, we set $x_i = w_i$. Furthermore, we set $x_{n+1} = u$ and $x_{n+2} = \mathbf{0}$. Then, the points x_1, \dots, x_{n+2} lie on a common hyperplane. For this, it suffices to check that

$$\mathbf{0} = \sum_{i=1}^{n+1} \frac{1}{n+1} x_i,$$

which follows immediately from the definitions and the choice of the x_i . Thus, there is a hyperplane transversal for S_1, \dots, S_{n+2} , as desired.

Conversely, suppose that S_1, \dots, S_{n+2} is a yes-input for HYPTRANS. Thus, there is a choice $x_i \in S_i$, for $i = 1, \dots, n+2$, such that x_1, \dots, x_{n+2} , lie on a common hyperplane. Obviously, we have $x_{n+1} = u$ and $x_{n+2} = \mathbf{0}$, so we can conclude that $\mathbf{0}$ is in the affine span of x_1, \dots, x_n, u and can be written as

$$\mathbf{0} = \sum_{i=1}^n \lambda_i x_i + \lambda_{n+1} u,$$

where $\lambda_i \in \mathbb{R}$ with $\sum_{i=1}^{n+1} \lambda_i = 1$. Let $I \subseteq [n]$ be the set of those indices i for which $x_i = v_i$. By inspecting the coordinates and applying the definitions, we get the following system of equations:

$$\begin{aligned} \sum_{i \in I} \lambda_i a_i &= \lambda_{n+1} b, & \text{and} \\ \lambda_i &= \lambda_{n+1}, & \text{for } i = 1, \dots, n. \end{aligned}$$

From this, it now follows that $\lambda_1 = \dots = \lambda_{n+1}$. Since $\sum_{i=1}^{n+1} \lambda_i = 1$, this implies that $\lambda_i = 1/(n+1)$, for $i = 1, \dots, n+1$. Thus, the first equation implies that a_1, \dots, a_n, b is a yes-input for SUBSETSUM, with I as the certifying index set. \triangleleft

3.1.3 A second reduction

Now, we prove that HYPTRANS is strongly NP-hard, by reducing from BINPACKING. Our reduction will pass through two intermediate problems EQUALBINPACKING and FLATTRANS. We start by defining all the involved problems.

In BINPACKING, we are given a sequence $w_1, \dots, w_n \in \mathbb{Z}_+$ of *weights*, a number k of *bins* and a *capacity* $b \in \mathbb{Z}_+$. The goal is to decide whether there is a partition of n items with weights w_1, \dots, w_n into k bins such that in each bin the total weight of the items does not exceed the capacity b . It is known that BINPACKING is strongly NP-hard. In EQUALBINPACKING, we are given the same input, but now the goal is to decide whether there exists a partition of the items into the bins such that in each bin the total weight of the items equals *exactly* the capacity. Note that BINPACKING can easily be reduced to EQUALBINPACKING by adding the appropriate number of elements of weight 1, so EQUALBINPACKING is strongly NP-hard as well.

Finally, in FLATTRANS, we are given m sets S_0, \dots, S_{m-1} in \mathbb{R}^d , where m and d are both part of the input, and the goal is to decide whether there is an $(m-2)$ -transversal. In other words, the question is whether there exists an $(m-2)$ -dimensional affine subspace h such that for all $i \in \{0, \dots, m-1\}$, we have $S_i \cap h \neq \emptyset$. Note that HYPTRANS with $k = d+1$ is the same as FLATTRANS with $m = d+1$.

► **Theorem 10.** *FLATTRANS is strongly NP-hard even when $S_0 = \{\mathbf{0}\}$ and any other S_i consists of at most two points.*

Proof. We reduce from EQUALBINPACKING. Given an input w_1, \dots, w_n, k, b for to EQUALBINPACKING, we construct an instance of FLATTRANS as follows: we set the dimension $d = k + n + kn$ and the number of sets $m = kn + 2$. For every pair $(i, j) \in [n] \times [k]$, define the vectors

$$v_{i,j}(x) := \begin{cases} w_i, & \text{if } x = j, \\ 1, & \text{if } x = k + i, \\ 1, & \text{if } x = n + k + (i-1)k + j, \\ 0, & \text{otherwise,} \end{cases}, \quad u_{i,j}(x) := \begin{cases} 0, & \text{if } x = j, \\ 0, & \text{if } x = k + i, \\ 1, & \text{if } x = n + k + (i-1)k + j, \\ 0, & \text{otherwise.} \end{cases}$$

Here, we denote by $x \in \{1, \dots, n+k+kn\}$ the entries of the vector, e.g., the first entry of $v_{i,j}$ is denoted by $v_{i,j}(1)$. Furthermore, let c be the vector whose entries are $-b$, for $1 \leq x \leq k$, and -1 everywhere else. Now set $S_0 = \{\mathbf{0}\}$, and $S_l = \{v_{i,j}, u_{i,j}\}$, for $l = (i-1)k + j$, $i = 1, \dots, n$, $j = 1, \dots, k$ (note that this choice of l just gives that the order of the l 's corresponds to the lexicographic order of the (i, j) 's) and $S_{kn+1} = \{c\}$. All these vectors can be constructed in polynomial time.

We claim that there is a kn -transversal of the sets S_0, \dots, S_{kn+1} , if and only if there is a valid solution for the EQUALBINPACKING instance.

Assume first that there is a solution for EQUALBINPACKING. For each S_l , $1 \leq l \leq kn$, $l = (i-1)k + j$, choose $p_l = v_{i,j}$, if item i is placed in bin j , and choose $p_l = u_{i,j}$, otherwise. Furthermore, set $p_0 = 0$, $p_{kn+1} = c$. We claim that there exist coefficients λ_l such that

$$\sum_{l=1}^{kn+1} \lambda_l p_l = \mathbf{0} \tag{1}$$

and

$$\sum_{l=1}^{kn+1} \lambda_l = kn + 1. \quad (2)$$

This implies the claim, because then $\mathbf{0}$ can be written as a non-trivial linear combination of the other points. Set $\lambda_l := 1$, for all l . Then, (2) is certainly satisfied. Consider the x 'th row of (1), where $1 \leq x \leq k$. By construction, and since we assumed a valid solution for the bin packing problem, this row evaluates to

$$\left(\sum_{i:\text{item } i \text{ in bin } x} w_i \right) - b = 0.$$

Similarly, for $k+1 \leq x \leq k+n$, the x 'th row evaluates to $1 - 1 = 0$, since each item is placed in exactly one bin. All remaining rows evaluate to $1 - 1 = 0$, and thus (2) is also satisfied.

Assume now that there exist coefficients λ_l that satisfy (1) and (2) (which must be the case of $\mathbf{0}$ can be written as a non-trivial linear combination of the other points). From the x 'th rows in (1) with $x > k+n$, we get $\lambda_l - \lambda_{kn+1} = 0$, for $1 \leq l \leq kn$, and thus $\lambda_1 = \dots = \lambda_{kn+1}$. Together with (2), we thus get $\lambda_l = 1$, for all l . Put item i into bin j if and only if $p_i = v_{i,j}$ for $l = (i-1)k + j$. Analogous to above we get from the x 'th rows of (1), for $k+1 \leq x \leq k+n$, that each item is placed into exactly one bin. Further, we get from the x 'th rows of (1), for $1 \leq x \leq k$, that each bin is filled exactly to capacity. Thus, we have a valid solution for EQUALBINPACKING, as desired. ◀

Now, there is only one reduction remaining:

► **Theorem 11.** *HYPTRANS is strongly NP-hard even when $S_0 = \{\mathbf{0}\}$ and S_i consists of at most two points for all $i = 1, \dots, m-1$.*

Proof. We reduce from FLATTRANS. Let us assume that $S_0 = \{\mathbf{0}\}$ and let $S_0, S_1, \dots, S_{m-1} \subset \mathbb{R}^d$ be the sets in the instance of FLATTRANS, and assume that $m-1 < d$. We construct sets in \mathbb{R}^{d+2} as follows: First, for each point p in some set S_i we define the point $p' = (p, 0, 0)$ and place it in the set S'_i . For $m \leq i \leq d+2$, define S'_i as the set consisting only of the point $s'_i = (0, \dots, 0, 1, i)$. Additionally, let $S'_0 := \{\mathbf{0}\}$.

We claim that $S_0, S_1, \dots, S_{m-1} \subset \mathbb{R}^d$ have an $(m-2)$ -transversal, if and only if $S'_0, S'_1, \dots, S'_{d+2} \subset \mathbb{R}^{d+2}$ can be transversed by a hyperplane.

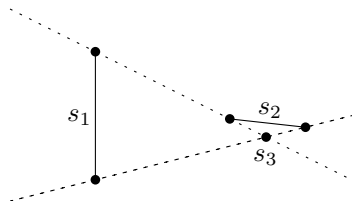
Assume first that $S_0, S_1, \dots, S_{m-1} \subset \mathbb{R}^d$ indeed have an $(m-2)$ -transversal, that is, there are points $p_i \in S_i$ and parameters λ_i such that $\sum_{i=1}^{m-1} \lambda_i p_i = \mathbf{0}$ and $\sum_{i=1}^{m-1} \lambda_i = 1$. Choosing the corresponding points p'_i and setting $\lambda'_i = \lambda_i$ for $i \leq m-1$ and $\lambda'_i = 0$ for $i > m-1$ we get $\sum_{i=1}^{d+2} \lambda'_i p'_i = \mathbf{0}$ and $\sum_{i=1}^{d+2} \lambda'_i = 1$, that is, $S'_0, S'_1, \dots, S'_{d+2} \subset \mathbb{R}^{d+2}$ can be transversed by a hyperplane.

Assume now that $S'_0, S'_1, \dots, S'_{d+2} \subset \mathbb{R}^{d+2}$ can be transversed by a hyperplane, that is, there are points $p'_i \in S'_i$ and parameters λ'_i , such that $\sum_{i=1}^{d+2} \lambda'_i p'_i = \mathbf{0}$ and $\sum_{i=1}^{d+2} \lambda'_i = 1$. The second to last row of the first equation evaluates to $\sum_{i=m}^{d+2} \lambda'_i = 0$, and we thus have $\sum_{i=1}^{m-1} \lambda'_i = 1$. Set $p_i = p'_i$ and $\lambda_i = \lambda'_i$. Then $\sum_{i=1}^{m-1} \lambda_i = 1$ by the observation above. Further, $\sum_{i=1}^{m-1} \lambda_i p_i = \mathbf{0}$ by the first m rows of the first equation. Thus, $S_0, S_1, \dots, S_{m-1} \subset \mathbb{R}^d$ can be transversed by a $(m-2)$ -flat. ◀

3.2 Line segments

In this section, we will show that deciding whether there is a hyperplane transversal for d line segments and the origin in \mathbb{R}^d , where d is not fixed, is NP-hard.

We will reduce this to one of the previous cases shown, that is, to the restricted version of HYPTRANS where the sets S_i contain at most two points, see Section 3.1.2. This is done with the help of a gadget that enforces that every hyperplane transversal must use one of the two endpoints of a given line segment. The gadget is shown in Figure 1.



■ **Figure 1** Every hyperplane transversal through s_1, s_2, s_3 must choose an endpoint of s_1 (and of s_2).

Given a collection of sets of size at most two, for each set we take the line segment formed by its points as s_1 , the origin as point s_3 , and we construct the corresponding new segment s_2 using the gadget presented in Figure 1. This gives a family S of $2k$ line segments that all lie in a k -dimensional space. In order to prove our result, we need to lift our construction to \mathbb{R}^{2k} . Let A_i, B_i in \mathbb{R}^k denote the endpoints of the i 'th original segment (s_1 in Figure 1) and let G_i, H_i in \mathbb{R}^k denote the endpoints of the i 'th gadget segment (s_2 in Figure 1). Denote by ε_j the vector in \mathbb{R}^k which is 0 everywhere except in the j 'th entry, where it is ε . Further, we write $\mathbf{0}^k$ for the zero vector in \mathbb{R}^k . We now lift the points A_i, B_i, G_i, H_i to \mathbb{R}^{2k} as follows:

$$A'_i := \begin{pmatrix} A_i \\ \mathbf{0}^k \end{pmatrix}, B'_i := \begin{pmatrix} B_i \\ \mathbf{0}^k \end{pmatrix}, G'_i := \begin{pmatrix} G_i \\ \varepsilon_i \end{pmatrix}, H'_i := \begin{pmatrix} H_i \\ \varepsilon_i \end{pmatrix}.$$

We denote the corresponding set of line segments $A'_i B'_i$ and $G'_i H'_i$ in \mathbb{R}^{2k} by S' .

► **Lemma 12.** $S \subset \mathbb{R}^k$ has a hyperplane transversal if and only if $S' \subset \mathbb{R}^{2k}$ does.

Proof. We will prove this by explicitly computing affine combinations of points on the line segments that give us the required transversals. In this setting, $S \subset \mathbb{R}^k$ has a hyperplane transversal if and only if there are real numbers $\lambda_i, \gamma_i, \mu_j^{(i)}$, with $i \in [k], j \in \{0, \dots, k\}$ and the following properties

$$\sum_{i=1}^k \mu_0^{(i)} (\lambda_i A_i + (1 - \lambda_i) B_i) = \mathbf{0}, \quad \sum_{i=1}^k \mu_0^{(i)} = 1; \quad (3)$$

and for all $j \in \{1, \dots, k\}$

$$\sum_{i=1}^k \mu_j^{(i)} (\lambda_i A_i + (1 - \lambda_i) B_i) = \gamma_j G_j + (1 - \gamma_j) H_j, \quad \sum_{i=1}^k \mu_j^{(i)} = 1. \quad (4)$$

Here, the λ_i and γ_i fix points on the segments, and the $\mu_j^{(i)}$ write the origin (Equation (3)) and the points on the gadget segments (Equation (4)) as affine combinations of the points on the reduction segments.

Similarly, $S' \subset \mathbb{R}^{2k}$ has a hyperplane transversal if and only if there are real $l_i, g_i, m^{(i)}, n^{(i)}$, with $i \in [k]$ with the following property:

$$\sum_{i=1}^k m^{(i)} (l_i A'_i + (1 - l_i) B'_i) + \sum_{i=1}^k n^{(i)} (g_i G'_i + (1 - g_i) H'_i) = \mathbf{0},$$

$$\sum_{i=1}^k m^{(i)} + n^{(i)} = 1. \quad (5)$$

16:10 Well-Separation and Hyperplane Transversals in High Dimensions

Here, the l_i and g_i fix points on the segments and the $m^{(i)}$ and $n^{(i)}$ write the origin as an affine combination of these points.

Assume first that $S \subset \mathbb{R}^k$ has a hyperplane transversal. Then Equation (5) can be satisfied by setting $l_i = \lambda_i$, $m^{(i)} := \mu_0^{(i)}$, $n^{(i)} := 0$, $g_i := 0$. Thus, if $S \subset \mathbb{R}^k$ has a hyperplane transversal then so does $S' \subset \mathbb{R}^{2k}$.

As for the other direction, assume that $S' \subset \mathbb{R}^{2k}$ has a hyperplane transversal. Note that the $(k+i)$ 'th row of Equation (5) reduces to $n^{(i)}\varepsilon = 0$, so in particular we must have $n^{(i)} = 0$ for every $i \in \{1, \dots, k\}$. Thus, we may set $\lambda_i := l_i$ and $\mu_0^{(i)} := m^{(i)}$ and Equation (3) follows. As for Equation (4), fix some $j \in \{1, \dots, k\}$ and note that by the construction of the gadget segments there exist real numbers α_j and β_j such that $G_j = \alpha_j A_j$ and $H_j = \beta_j B_j$. Pick real numbers γ_j and x_j that satisfy the following two equations:

$$x_j \lambda_j = (1 + x_j) \gamma_j \alpha_j, \quad \text{and} \quad x_j (1 - \lambda_j) = (1 + x_j) (1 - \gamma_j) \beta_j. \quad (6)$$

It is straightforward to show that such numbers always exist, for the sake of readability we will not prove this here. Now, define $\mu_j^{(i)} := \frac{m^{(i)}}{1+x_j}$ for $j \neq i$ and $\mu_j^{(j)} := \frac{m^{(j)} + x_j}{1+x_j}$. Then

$$\sum_{i=1}^k \mu_j^{(i)} (\lambda_i A_i + (1 - \lambda_i) B_i) = \frac{1}{1+x_j} \sum_{i=1}^k m^{(i)} (l_i A_i + (1 - l_i) B_i) + \frac{x_j}{1+x_j} (l_j A_j + (1 - l_j) B_j).$$

By Equation 5, we have $\sum_{i=1}^k m^{(i)} (l_i A_i + (1 - l_i) B_i) = 0$ (recall that $n^{(i)} = 0$), thus we have

$$\sum_{i=1}^k \mu_j^{(i)} (\lambda_i A_i + (1 - \lambda_i) B_i) = \frac{1}{1+x_j} (x_j l_j A_j + x_j (1 - l_j) B_j).$$

From our choice of γ_j and x_j , we thus get

$$\frac{1}{1+x_j} (x_j l_j A_j + x_j (1 - l_j) B_j) = \gamma_j \alpha_j A_j + (1 - \gamma_j) \beta_j B_j = \gamma_j G_j + (1 - \gamma_j) H_j,$$

which is what we want. Further, we have

$$\sum_{i=1}^k \mu_j^{(i)} = \frac{1}{1+x_j} \left(\sum_{i=1}^k m^{(i)} + x_j \right) = \frac{1+x_j}{1+x_j} = 1,$$

so Equation (4) is indeed satisfied. ◀

4 Parametrized complexity

4.1 An FPT algorithm for d sets

Recall that our original motivation comes from determining whether d sets in \mathbb{R}^d are well-separated. Let us consider those d sets, and let us denote by n the total number of extreme vertices on their respective convex hulls (for general convex sets, this might be infinite, but we consider only the finite case). We say that n is the *convex hull complexity* of the set family. We assume that we are given the extreme points of the convex hull of every set and hence have a finite number of points for every set.

► **Theorem 13.** *Checking whether a family of k sets in \mathbb{R}^d with convex hull complexity n is well-separated is FPT with parameter d .*

Proof. For the $O(2^d)$ choices of index sets $I \subset [k]$, we check whether the convex hulls of S_I and $S_{[k] \setminus I}$ intersect. For each I , we check with an LP whether there is a hyperplane separating the points from S_I from the points in $S_{[k] \setminus I}$. This can be done by a linear program with $d+1$ variables a_0, a_1, \dots, a_d describing a hyperplane in \mathbb{R}^d . The hyperplane is separating if the constraints

$$a_0 + \sum_{i=1}^d a_i p_i \geq 0 \quad \text{for all } p = (p_1, \dots, p_d) \in S_I \quad \text{and}$$

$$a_0 + \sum_{i=1}^d a_i q_i \leq 0 \quad \text{for all } q = (q_1, \dots, q_d) \in S_{[k] \setminus I}$$

In total we have $O(n)$ constraints.

If there exists a hyperplane for every I , we output that the family is well-separated. Thus, there exists a constant $c > 0$ such that the total running time of the algorithm is in $O(2^d(nd)^c L)$, where L is the number of input bits. ◀

4.2 A $W[1]$ -hardness proof

► **Theorem 14.** FLATTRANS is $W[1]$ -hard with respect to the dimension.

Proof. We use a framework similar to the one introduced by Marx [15]. The reduction is from the following problem: Given a graph $G = (V, E)$ with n vertices, is there a clique of size k in G ?

Before describing the point sets, we first explain the framework. We define a set of k^2 gadgets, that we call the *encoding gadgets*. To help defining them, we assume that these gadgets lie on k rows and k columns. Note that this representation is purely a help for the definition, but does not correspond to any geometric structure of the point sets we define later. To each gadget we assign a set of admissible tuples (i, j) , with $1 \leq i, j \leq n$. Let us assume that we are considering the gadget in row α and column β , with $1 \leq \alpha, \beta \leq k$. If $\alpha = \beta$, the set of admissible tuples is $\{(i, i) \mid 1 \leq i \leq n\}$. Otherwise, the set of admissible tuples is $\{(i, j) \mid \{i, j\} \in E\}$. We have in addition the *row gadgets* and the *column gadgets*. A row gadget forces the left value of each encoding gadget from the same row to be the same. Similarly, a column gadget forces the right value of every encoding gadget from the same column to be the same. There is a row gadget for each row, and a column gadget for each column. We say that an encoding is *valid* if each encoding gadget is assigned an admissible tuple, and if all the row and column gadgets are satisfied. As shown by Marx [15], G has a clique of size k if and only if there exists a valid encoding. First let us assume that v_1, \dots, v_k form a clique. Then we assign to the encoding gadget in row α and column β the tuple (v_α, v_β) . Observe that this is an admissible tuple (as there is an edge between v_α and v_β), and that the encoding is valid since all rows have the same left value, and all columns have the same right value. Reciprocally, let us assume that we have a valid encoding. Assume that the left value of row α is i , and that the left value of row $\beta \neq \alpha$ is j . Then the encoding gadget in row α and column α is assigned the tuple (i, i) , thus column α is assigned right value i , which implies that the encoding gadget in row β and column α is assigned the tuple (j, i) . We have shown that vertices i and j in G are adjacent.

We now describe how to reduce the valid encoding problem to FLATTRANS. We define $k^2 + 2k + 2$ point sets in \mathbb{R}^{k^2+4k} . Let k' denote $k^2 + 2k$ and let k'' denote $k^2 + 3k$. We consider the k' gadgets from the framework described above, that is, k^2 encoding gadgets as well as k row and k column gadgets, respectively. Let f denote a bijective function

16:12 Well-Separation and Hyperplane Transversals in High Dimensions

from the set of gadgets to $[k']$. For each encoding gadget g in row α and column β , $1 \leq \alpha, \beta \leq k$ we have a point set $P^{\alpha, \beta}$ that contains $O(n^2)$ points. First let us assume $\alpha = \beta$. The point set $P^{\alpha, \alpha}$ contains the points $p_i^{\alpha, \alpha}$, for $1 \leq i \leq n$, where the coordinates of $p_i^{\alpha, \alpha}$ are: $p_i^{\alpha, \alpha}(x) = \delta_{f(g), x} + k^i \delta_{k'+\alpha, x} + k^i \delta_{k''+\alpha, x}$. Now let us assume that $\alpha \neq \beta$. The point set $P^{\alpha, \beta}$ contains the points $p_{i,j}^{\alpha, \beta}$, for $1 \leq i, j \leq n$ and $\{i, j\} \in E$, where the coordinates of $p_{i,j}^{\alpha, \beta}$ are: $p_{i,j}^{\alpha, \beta}(x) = \delta_{f(g), x} + k^i \delta_{k'+\alpha, x} + k^j \delta_{k''+\beta, x}$. Now let g be a row gadget, say for row α . The point set $P^{\alpha, \cdot}$ contains the points $p_i^{\alpha, \cdot}$, for $1 \leq i \leq n$, where $p_i^{\alpha, \cdot}(x) = \delta_{f(g), x} - k^{i+1} \delta_{k'+\alpha, x}$. Similarly, we have a point set $P^{\cdot, \beta}$ for the column gadget g in column β , and $p_i^{\cdot, \beta}(x) = \delta_{f(g), x} - k^{i+1} \delta_{k''+\beta, x}$ for $1 \leq i \leq n$. Finally, we have the point set $P_0 = \{\mathbf{0}\}$ and the point set $P_1 = \{p_1\}$, where for all $1 \leq x \leq k'$, $p_1(x) = -1$, and $p_1(x) = 0$ otherwise. Observe that we have indeed $k^2 + 2k + 2$ point sets of size $O(n^2)$ in \mathbb{R}^{k^2+4k} . The absolute values of all point coordinates are at most k^{n+1} . Thus, we can describe it with $\log(k^{n+1}) = (n+1) \log(k)$ bits. We claim that there is a $(k^2 + 2k)$ -transversal if and only if G has a clique of size k . From the reduction, this immediately implies that FLATTRANS is $W[1]$ -hard with respect to the dimension.

First let us assume that there is a clique of size k in G . From what we argued, it implies that there is a valid encoding of the gadgets. We define a set of $k' + 1$ points as follows. First we take the point p_1 . If the tuple assigned to gadget in row α and column $\beta \neq \alpha$ is (i, j) , then we take the point $p_{i,j}^{\alpha, \beta}$. If the gadget in row α and column α is assigned the tuple (i, i) , then we take the point $p_i^{\alpha, \alpha}$. Likewise, if the left value of row α is i , we take the point $p_i^{\alpha, \cdot}$. Finally, if the right value of column β is j , we take the point $p_i^{\cdot, \beta}$. We denote those $k' + 1$ points by $p_1, \dots, p_{k'+1}$ and claim that they lie on a common hyperplane which contains $\mathbf{0}$. It suffices to show that

$$\sum_{1 \leq \ell \leq k'+1} \frac{1}{k'+1} p_\ell = \mathbf{0}.$$

Consider the first k' coordinates. Recall that f is a bijection between the set of gadgets and $[k']$ and recall that by definition, the points p_ℓ have exactly one entry 1 in the first k' coordinates. Therefore in this sum, we have exactly one entry 1 from exactly one of the gadgets and exactly one entry -1 from the point p_1 in each of these coordinates. So it is clear that this equation is satisfied in the first k' coordinates. Now let us consider the coordinate $k' + \alpha$, for some $1 \leq \alpha \leq k$. As the encoding is valid, it implies that the left value in row α of all encoding gadgets is the same. Let us denote by i this left value. We have indeed

$$\sum_{1 \leq \ell \leq k'+1} \frac{1}{k'+1} p_\ell(k' + \alpha) = \frac{1}{k'+1} \left(\left(\sum_{1 \leq \beta \leq k} k^i \right) - k^{i+1} \right) = 0.$$

Likewise if the coordinate is of the form $k'' + \beta$ for some $1 \leq \beta \leq k$, we argue using the fact that the right value of all encoding gadgets in column β is the same. This completes the first direction of our proof.

For the second direction, let us assume that there is a hyperplane h that contains at least one point from each point set. By assumption one of these points is $\mathbf{0}$, another is p_1 , and we denote the others by $p_2, \dots, p_{k'+1}$. This implies that we have $\mathbf{0} = \lambda_1 p_1 + \sum_{2 \leq \ell \leq k'+1} \lambda_\ell p_\ell$, where $\lambda_\ell \in \mathbb{R}$ and $\sum_{1 \leq \ell \leq k'+1} \lambda_\ell = 1$. By looking at the k' first coordinates, we immediately obtain $\lambda_1 = \lambda_i = \frac{1}{k'+1}$, for all $2 \leq i \leq k'+1$. Let assume that in point set $P^{\alpha, \beta}$ with $1 \leq \alpha, \beta \leq k$, the point $p_{i,j}^{\alpha, \beta}$ is contained in h , for some $1 \leq i, j \leq n$. Note that by definition, (i, j) is an admissible tuple of the encoding gadget in row α and column β . We assign this tuple to this gadget, and do likewise with all other encoding gadgets. It remains to show

that the left value of all encoding gadgets in the same row is the same, and that the same holds with the right value of encoding gadgets from the same column. Let us consider row α . We consider the points contained in h that belong to $P^{\alpha,\beta}$, for some $1 \leq \beta \leq k$. Let us denote by Y the set of their $(k' + \alpha)$ -th coordinate. Let z be equal to $\max\{\log_k(y) \mid y \in Y\}$. By assumption, we know that $\sum_{y \in Y} y = k^i$ for some $2 \leq i \leq n + 1$. This is because the coefficients λ_ℓ for these point sets are equal to the coefficient for the point in $P^{\alpha,\cdot}$ contained in h . As the elements in Y are non-negative, we obtain $i \geq z + 1$. Assume for a contradiction that not all elements in Y are equal. Then we have $\sum_{y \in Y} y < \sum_{y \in Y} k^z = k^{z+1} \leq k^i$. As this is not possible, we know that all elements in Y are equal, which implies that the left value of all encoding gadgets in row α is the same. We can argue likewise for the columns. ◀

5 Conclusion and Open Problems

We showed that the problem of testing well-separability of k sets in \mathbb{R}^d is hard. However, it may be that there exist some algorithms which solve the problem in a smarter way than simply testing the 2^k choices of index set. This question is still open.

It would be interesting to have some inapproximability results, or some better approximation algorithms, for the problem of finding a hyperplane that intersects as many points as possible in a point set P in \mathbb{R}^d , where d is not fixed.

References

- 1 Pankaj K. Agarwal. On stabbing lines for convex polyhedra in 3d. *Comput. Geom. Theory Appl.*, 4(4):177–189, 1994.
- 2 Nina Amenta, Jesús A De Loera, and Pablo Soberón. Helly’s theorem: new variations and applications. *arXiv preprint arXiv:1508.07606*, 2015.
- 3 David Avis and Mike Doskas. Algorithms for high dimensional stabbing problems. *Discrete applied mathematics*, 27(1-2):39–48, 1990.
- 4 David Avis and Rephael Wenger. Algorithms for line transversals in space. In *Proc. 3rd Annu. Sympos. Comput. Geom. (SoCG)*, pages 300–307, 1987.
- 5 Imre Bárány, Alfredo Hubard, and Jesús Jerónimo. Slicing convex sets and measures by a hyperplane. *Discrete Comput. Geom.*, 39(1-3):67–75, 2008.
- 6 Ted Bisztriczky. On separated families of convex bodies. *Archiv der Mathematik*, 54(2):193–199, 1990.
- 7 Federico Castillo, Joseph Doolittle, and Jose Alejandro Samper. Common tangents to polytopes. *arXiv preprint*, 2021. [arXiv:2108.13569](https://arxiv.org/abs/2108.13569).
- 8 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational complexity of the α -ham-sandwich problem. In *Proc. 47th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 31:1–31:18, 2020.
- 9 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 638–649, 2019.
- 10 Jacob E. Goodman, Richard Pollack, and Rephael Wenger. Geometric transversal theory. In *New trends in discrete and computational geometry*, pages 163–198. Springer, 1993.
- 11 Hugo Hadwiger. Über Eibereiche mit gemeinsamer Treffgeraden. *Portugaliae mathematica*, 16(1):23–29, 1957.
- 12 Eduard Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.
- 13 Andreas Holmsen and Rephael Wenger. 4 Helly-type theorems and geometric transversals. *Handbook of Discrete and Computational Geometry*, 2017.

16:14 Well-Separation and Hyperplane Transversals in High Dimensions

- 14 Christian Knauer, Hans Raj Tiwary, and Daniel Werner. On the computational complexity of ham-sandwich cuts, helly sets, and related problems. In *Proc. 28th Sympos. Theoret. Aspects Comput. Sci. (STACS)*, volume 9, pages 649–660, 2011.
- 15 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *International Workshop on Parameterized and Exact Computation*, pages 154–165. Springer, 2006.
- 16 Jiří Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002. doi:10.1007/978-1-4613-0039-7.
- 17 Marco Pellegrini and Peter W. Shor. Finding stabbing lines in 3-space. *Discrete Comput. Geom.*, 8(2):191–208, 1992.
- 18 Richard Pollack and Rephael Wenger. Necessary and sufficient conditions for hyperplane transversals. *Combinatorica*, 10(3):307–311, 1990.
- 19 William Steiger and Jihui Zhao. Generalized ham-sandwich cuts. *Discrete Comput. Geom.*, 44(3):535–545, 2010.
- 20 Arthur H. Stone and John W. Tukey. Generalized “sandwich” theorems. *Duke Math. J.*, 9(2):356–359, June 1942.
- 21 Rephael Wenger. *Geometric permutations and connected components*. DIMACS, Center for Discrete Mathematics and Theoretical Computer Science, 1990.

Lions and Contamination: Monotone Clearings

Daniel Bertschinger ✉

Department of Computer Science, ETH Zürich, Switzerland

Meghana M. Reddy¹ ✉

Department of Computer Science, ETH Zürich, Switzerland

Enrico Mann ✉

Department of Computer Science, ETH Zürich, Switzerland

Abstract

We consider a special variant of a pursuit-evasion game called lions and contamination. In a graph whose vertices are originally contaminated, a set of lions walk around the graph and clear the contamination from every vertex they visit. The contamination, however, simultaneously spreads to any adjacent vertex not occupied by a lion. We study the relationship between different types of clearings of graphs, such as clearings which do not allow recontamination, clearings where at most one lion moves at each time step and clearings where lions are forbidden to be stacked on the same vertex. We answer several questions raised by Adams et al. [2].

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Algorithmic Games, Pursuit-Evasion Games, Graph Contamination, Clearings

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.17

Funding *Meghana M. Reddy*: Supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

1 Introduction

Pursuit-evasion problems have a long and rich history going back more than 50 years [11, 14]. Countless similar problems have been studied under very different names in the past. What they all have in common is that there is a group of pursuers that try to catch an evader. The typical question asked in a pursuit-evasion problem is whether the evader can escape the pursuers, and if so, for how long. Naturally, the more pursuers there are, the harder it is for the evader to escape. Some objectives of the pursuers can be to catch the evader fast (minimize the time taken) or with minimal effort (minimize the distance traveled); the different objectives all have their origins in numerous applications such as robot motion planning, collision avoidance, and intruder detection in networks [1, 8, 13].

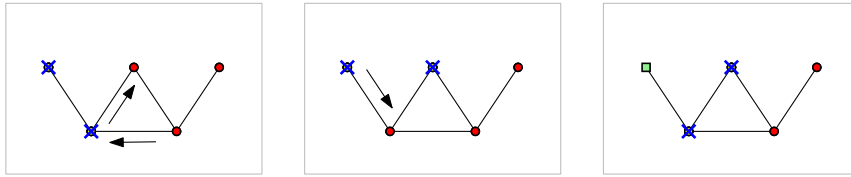
There are different variations of the problem depending on the exact rules. For detailed definitions of the various problems, see the surveys [4, 5, 7, 10] and the references therein. In this paper, we study the problem of lions and contamination [9]. Traditionally, a group of lions tries to eradicate contamination from a graph while the contamination spreads to all adjacent vertices that are not occupied by lions. A scenario that is relevant to current day is where a set of doctors tries to get rid of a disease; people can get tested and quarantined to stop the spread of the disease, whereas the contamination spreads to people that had contact with infected people.

¹ The second author's full last name consists of two words and is *Mallik Reddy*. However, she consistently refers to herself with the first word of her last name being abbreviated.



17:2 Lions and Contamination: Monotone Clearings

Formally, suppose there is a graph $G = (V, E)$. At the very beginning, every vertex occupied by a lion is considered cleared of contamination, whereas the remaining vertices are considered to be contaminated. Time is viewed as discrete; and in each step, the lions and the contamination both move simultaneously. Every lion is allowed to move along an edge that is incident to its current position. Contamination, on the other hand, spreads along every incident edge to every adjacent vertex, unless the edge is used by a lion or a lion occupies the adjacent vertex. Figure 1 illustrates an example. Note that the sequential variant of this problem, where the lions and contamination move one after the other in alternating time steps, results in a setting where the lions are more powerful and hence is a very different problem compared to the one we study.



■ **Figure 1** A graph and two lions (indicated by blue crosses), every vertex that is not occupied by a lion is contaminated initially (indicated by red disks). One lion moves in the first time step, however, contamination moves simultaneously and the vertex gets recontaminated. After the second step, a vertex that is not occupied by a lion is cleared of contamination (indicated by green squares).

Note that for some graphs, it is easy to see whether k lions can clear the graph of contamination. However, finding the minimum number of lions required to clear a graph is a hard question. For example, the minimum number of lions required to clear the $n \times n$ -grid is not known. Nevertheless, it is known that at least $\lfloor \frac{n}{2} \rfloor + 1$ lions are needed [6]. Since n lions can simply sweep the graph from left to right and clear the grid of contamination, n is an upper bound on the number of lions needed for clearing the $n \times n$ -grid, and this is the best upper bound currently known. Further, it is believed that $n - 1$ lions are not sufficient. For higher-dimensional grids, that is for the n^d -grid, it is known that $\Theta(n^{d-1}/\sqrt{d})$ lions are necessary and sufficient [3].

In this paper, we study different types of clearings that were defined by Adams et al. [2] and answer several questions raised in their paper. A clearing of a graph using k lions is denoted as a k -clearing and the graph itself is referred to as k -clearable. We say a clearing is *monotone* if no vertex ever gets recontaminated. The lions and the clearing are said to be *polite* if at most one lion moves in each time step and *non-stacked* if no two lions occupy the same vertex at any point in time.

The remainder of this paper is organized in the following way. In Section 2, we show that there exist k -clearable graphs which require more than k lions for any monotone clearing. This implies that monotone clearings are harder to achieve than non-monotone clearings and stands in contrast to a result in the *graph searching* setting [12], where monotonicity does not matter. In Section 3, we show that any monotone clearing can be paused at any time and no recontamination occurs. This allows us to show in an algorithmic way that any monotone clearing can be converted into a monotone and polite clearing. We then show that polite clearings can be transformed into non-stacked clearings (see Theorem 3.6). Finally, in Section 4, we tackle the subgraph question raised in [2]: given a k -clearable graph G and some subgraph $H \subseteq G$, is H k -clearable as well? We answer this question in some settings.

2 Monotone Clearings

2.1 The $n \times n$ Grid

Let us consider the $n \times n$ grid. As already mentioned, at least $\lfloor \frac{n}{2} \rfloor + 1$ lions are needed, while n lions are sufficient to clear the grid. When restricted to monotone clearings, we can improve the lower bound and close the gap between the bounds.

Let $V(t)$ be the set of cleared vertices at time t . We define a *boundary vertex* as a vertex of $V(t)$ that has a neighbor in $V \setminus V(t)$. Before we state and prove our result, we first make two simple observations and recall a lemma proved by Berger et al. [3].

► **Observation 2.1.** *The number of cleared vertices in a k -clearing cannot increase by more than k in one time step.*

► **Observation 2.2.** *If there are more than k boundary vertices at time t , then at least one vertex gets recontaminated in the next time step.*

► **Lemma 2.3** (Lemma 5 of [3]). *Any vertex set S that is a subset of the $n \times n$ grid and satisfies $\frac{n^2}{2} - \frac{n}{2} \leq |S| \leq \frac{n^2}{2} + \frac{n}{2}$ has at least n boundary vertices.*

► **Theorem 2.4.** *A monotone clearing of the $n \times n$ grid needs at least n lions.*

Proof. Assume that the $n \times n$ grid has a monotone clearing with $n - 1$ lions. Initially, the lions start with at most $n - 1 < \frac{n^2}{2} + \frac{n}{2}$ cleared vertices and eventually have to clear all n^2 vertices. Further, Observation 2.1 implies that at most $n - 1$ vertices are cleared in each time step. Thus, at some time t , the set of cleared vertices $V(t)$ must satisfy the condition $\frac{n^2}{2} - \frac{n}{2} \leq |V(t)| \leq \frac{n^2}{2} + \frac{n}{2}$. The number of boundary vertices at such a time t will be at least n due to Lemma 2.3, and Observation 2.2 then implies that at least one vertex will get recontaminated at time $t + 1$. Hence, no monotone $(n - 1)$ -clearing of the $n \times n$ grid exists. ◀

In hindsight, this might not be a very surprising result. However, this does not necessarily improve the lower bound for non-monotone clearings as we will see in the next subsection.

2.2 Graphs with no Monotone Clearing

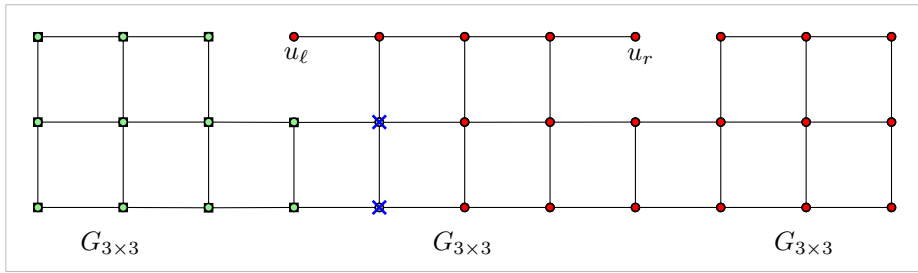
Unfortunately, not every graph with a k -clearing admits a monotone k -clearing. Indeed, we can show that the set of monotone k -clearable graphs is a proper subset of the set of all k -clearable graphs, which implies that monotonicity is a strong assumption on clearings. Theorem 2.4 does not improve the general lower bound for grids due to this reason.

► **Theorem 2.5.** *For any $k \geq 2$ there exist k -clearable graphs with no monotone k -clearing.*

We first describe the construction of one set of such graphs for any $k \geq 2$. We start with three distinct $k \times k$ -grids connected by $(k - 1)$ -grid-like paths from row 2 through row n . The center grid additionally has two vertices of degree one each attached to its leftmost and rightmost vertices of row 1 respectively, let these vertices be denoted by u_ℓ and u_r . For simplicity let us denote graphs constructed in this form as G_k , the graph G_3 is illustrated in Figure 2.

► **Lemma 2.6.** *The graph G_k is k -clearable for any $k \geq 2$, but admits no monotone k -clearing.*

17:4 Lions and Contamination: Monotone Clearings



■ **Figure 2** Illustration of G_k when $k = 3$. The center grid has two additional vertices of degree one attached at distinct corners of the first row. The colors indicate one possible clearing state.

Proof. To see that G_k is k -clearable we describe a clearing. Each lion is assigned to clear one row of the graph. The lions start on the leftmost vertices of the graph and start sweeping the graph from left to right. After the lions have cleared the left grid, the lions on rows $2, \dots, n$ move to the vertices on the $(k - 1)$ -path between the left and center grids and wait for the lion from row 1 to move to u_ℓ . Observe that this can be achieved without stacking lions, by moving the lion of row 1 to the position of the lion of row 2, and moving the lion of row 2 to u_ℓ . All the lions then together sweep further until the center grid and u_r are cleared; and similarly wait on the $(k - 1)$ -path between the center and right grids before finally clearing the right grid.

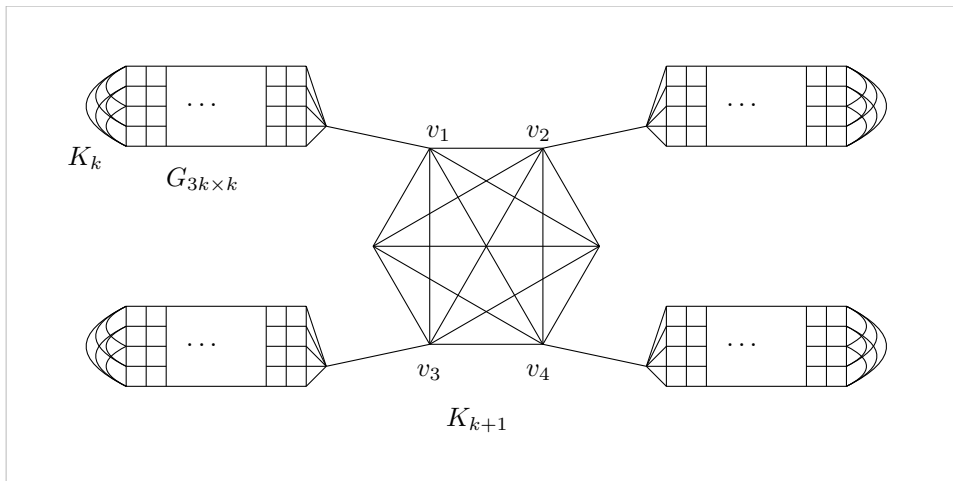
It remains to show that this graph cannot be cleared in a monotone fashion by k lions. To clear the graph monotonically, the lions must all either start on the left or the right grids. All or any subset of the k lions starting in the center grid will always lead to recontamination of some vertex (in the center) since the left and right grids are contaminated. Without loss of generality, assume the lions start in the left grid. Irrespective of the strategy followed by the lions, the lions always end up in a situation where either there is a lion each on u_ℓ and its neighbor, or there are $k - 1$ lions that stop the movement of contamination from the center grid to the left grid. In the first case, the remaining $k - 2$ lions are not sufficient to stop contamination from recontaminating the left grid, and in the second, the neighbor of u_ℓ gets recontaminated when the last lion moves to u_ℓ to clear the vertex. Hence, we can conclude that no monotone clearing exists for G_k . ◀

We give another construction for graphs that have clearings with k lions but do not admit monotone k -clearings. It gives further insights into how such graphs can look like. Interestingly, the following construction has four cut-vertices; which should ideally make it easy to isolate contamination to one section of the graph.

We denote the second type of graph as H_k . The graph H_k consist of several parts. The main building block is a $(k + 1)$ -clique K_{k+1} , to which we add four *arms* at different vertices. Each of the arms consists of a long k -grid (length of $3k$ is enough), where the farthest vertices form a clique themselves; see Figure 3 for an example with $k = 5$. The graph H_k is $(k + 1)$ -clearable, but has no monotone $(k + 1)$ -clearing.

Theorem 2.5 shows that monotonicity is a very restrictive assumption on clearings. This result however raises the following question for further research.

► **Open Question 1.** *Given a k -clearable graph $G = (V, E)$, is it always monotonically clearable with $k + 1$ lions? More formally, is there a non-trivial upper bound on the number of lions required for a monotone clearing?*



■ **Figure 3** The graph H_k consisting of a $(k + 1)$ -clique in the middle and four attached arms, each consisting of a long k -grid with a k -clique at the far end.

3 Transforming between Different Types of Clearings

3.1 Monotone Clearings

We now analyse some properties of monotone clearings. These will then allow us to show that monotone clearings can always be adapted to monotone clearings with polite lions. We start with an important observation.

► **Proposition 3.1.** *Let \mathcal{C} be any clearing of a graph G . Assume that all the lions are paused indefinitely at time t , that is, no lion moves after time t . If no vertex gets recontaminated at time $t + 1$, then no vertex gets recontaminated at a later time either.*

Proof. Recall that contamination spreads along every incident edge at each time step. If no vertex gets recontaminated at time $t + 1$, it implies that every cleared vertex neighboring a contaminated vertex is occupied by a lion that blocks the contamination from spreading. Then, no vertex can get recontaminated after time $t + 1$ either, since the lions continue to block the contamination from spreading. ◀

With this result, we are now able to argue about stopping lions in monotone clearings.

► **Lemma 3.2.** *A monotone clearing can be paused at any time and no vertex gets recontaminated.*

Proof. Assume that every lion is paused at time t in a monotone clearing. For contradiction, assume v is one of the first vertices to get recontaminated. Then no lion occupies vertex v at time t , vertex v is not contaminated at time t and v gets recontaminated earliest at time $t + 1$ due to Proposition 3.1.

Then there must exist $w \in N(v)$ that is contaminated at time t , which contaminates v at time $t + 1$. Since the clearing is monotone until time t , vertex w must have never been cleared of contamination. Therefore, the contamination was present at w at $t - 1$ and must have spread to v at time t (recall that no lion occupies v at time t). This contradicts the fact that no vertex gets recontaminated in a monotone clearing, and our clearing remains monotone until and including time t . ◀

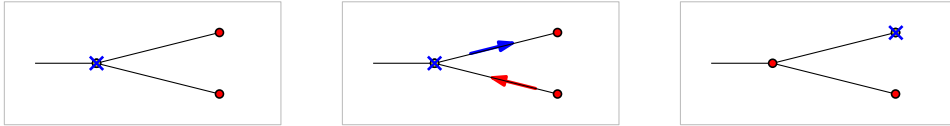
17:6 Lions and Contamination: Monotone Clearings

With a more careful analysis of pausing monotone clearings, we can prove a much stronger result.

► **Theorem 3.3.** *Let G be a graph and \mathcal{C} be a monotone clearing of G with k lions. Then, there exists another monotone clearing which uses k polite lions.*

Before being able to prove Theorem 3.3, we need some observations. Let us reconsider a monotone clearing, denoted by \mathcal{C} . Let us once again pause all the lions at time t . By Lemma 3.2 we know that no recontamination occurs. Consider a lion ℓ_1 and assume that this lion moved from v to w at time $t + 1$ in the clearing \mathcal{C} . We define a new strategy for a clearing, denoted by \mathcal{C}' , where we move all lions according to \mathcal{C} up to time t and at time $t + 1$, we move lion ℓ_1 from v to w . No other lion moves at time $t + 1$ or after.

It is now important to note that this strategy might not be a clearing and it can happen that vertices get recontaminated, see Figure 4. However, we certainly know which vertex gets recontaminated first. More specifically, we can prove vertex v is the only vertex that can get recontaminated at $t + 1$.



■ **Figure 4** A lion moving from the left vertex to one of the vertices on the right. The left vertex gets recontaminated at the same time.

► **Lemma 3.4.** *Let \mathcal{C} be a monotone clearing. Let \mathcal{C}' be a new strategy that mirrors \mathcal{C} until time t , and moves one lion from v to w at time $t + 1$, and no other lion is moved from $t + 1$ onwards. If recontamination occurs in \mathcal{C}' , the first vertex that gets recontaminated must be v and it gets recontaminated at time $t + 1$. Further, no other vertex except v can get recontaminated at time $t + 1$.*

Proof. We know that if \mathcal{C}' is paused at t , no recontamination occurs due to Lemma 3.2. The only difference between \mathcal{C} and \mathcal{C}' at $t + 1$ is that the lion ℓ_1 moved away from v in \mathcal{C}' . Thus, at time $t + 1$, no vertex other than v can get recontaminated. If v is not recontaminated at time $t + 1$, then by Proposition 3.1, we know that no vertex ever gets recontaminated. Hence, v is the only vertex that might get recontaminated at time $t + 1$. ◀

Unfortunately, we may not be able to avoid this recontamination when moving lion ℓ_1 in \mathcal{C}' . Nonetheless, we use the fact that vertex v does not get recontaminated in \mathcal{C} and analyse the situation closely. Let us assume that v gets recontaminated in \mathcal{C}' at time $t + 1$.

Let us take a closer look at this vertex v . Since v is recontaminated at time $t + 1$ in \mathcal{C}' , there must exist $x \in N(v)$ that was contaminated at t in \mathcal{C}' , which spread the contamination to v at $t + 1$. Observe that x must be contaminated at time t in the original clearing \mathcal{C} as well and v is not recontaminated at time $t + 1$ in \mathcal{C} . Therefore some lion must occupy v at time $t + 1$ in \mathcal{C} (otherwise v would also get recontaminated in the monotone clearing \mathcal{C} at time $t + 1$). Let this lion be ℓ_j . Note that $\ell_j \neq \ell_1$ as ℓ_1 moved away from v at time $t + 1$ in \mathcal{C} . If we move ℓ_j before moving ℓ_1 , then the recontamination of vertex v can be avoided at time $t + 1$. Hence, we say that ℓ_1 depends on ℓ_j at time $t + 1$.

We construct a graph on the set of lions, where a directed edge from ℓ_j to ℓ_i is added if and only if lion ℓ_j moves to vertex u at time $t + 1$ in \mathcal{C} , and the lion ℓ_i moves away from u at time $t + 1$ in \mathcal{C} . In this way, we capture all dependencies of the lions and we refer to this graph as the *dependency graph* of lions at time $t + 1$. Note that the dependency graph can contain cycles.

► **Observation 3.5.** *By renaming lions, we can avoid any cycle in the dependency graph.*

To see this, we consider one cycle in the dependency graph. Note that this cycle corresponds to a set of vertices in the underlying graph that are occupied by a set of lions; and occupied by the same set of lions at time $t + 1$ in \mathcal{C} . Each lion moved to a different vertex in the same set. Thus, the state of the corresponding vertices in the underlying graph G is the same at time t and at time $t + 1$; and in particular this does not change if the lions are not moved but are only renamed. Once the lions belonging to one cycle are renamed, the dependency graph clearly changes. In particular, the number of edges (and cycles) strictly decreases by renaming the lions and hence we eventually get an acyclic dependency graph.

With this we can now prove Theorem 3.3.

Proof of Theorem 3.3. We prove the theorem by showing how the number of time steps that use polite lions can be iteratively increased by modifying a monotone clearing. Given a monotone k -clearing \mathcal{C} that is polite up to some time t , we define a new clearing \mathcal{C}' that is identical to \mathcal{C} up to time t and is polite up to some time $t' > t$. In particular, let \mathcal{L}_{t+1} be the set of lions that move at time $t + 1$ in \mathcal{C} . Consider the dependency graph of \mathcal{L}_{t+1} at time $t + 1$ obtained after all the cycles have been removed (and thus some lions might have been renamed). Since this graph is acyclic, a topological ordering τ of the lions can be obtained such that any lion ℓ_i that depends on ℓ_j only comes after ℓ_j in τ . After time t , the new clearing \mathcal{C}' moves lions one by one according to the order τ up to time $t' \leq t + k$ (the lions not in τ do not move). Note that the state of \mathcal{C}' at time t' is identical to the state of \mathcal{C} at time $t + 1$. Finally, \mathcal{C}' follows the same strategy as \mathcal{C} did from time $t + 2$ until the graph is completely cleared.

It is easy to see that \mathcal{C}' is monotone up to time t and since there is no recontamination between time t and $t + k$, we indeed have a clearing. Furthermore, \mathcal{C}' is polite up to time $t' > t$, while \mathcal{C} was polite only until time t . Following this procedure iteratively, we are guaranteed to eventually get a monotone k -clearing that uses only polite lions from any monotone k -clearing. ◀

Note that this proof is algorithmic, in particular, given a monotone clearing, we can compute another monotone clearing that uses polite lions without increasing the number of lions.

3.2 Polite and Non-Stacked Clearings

In this subsection, we study clearings which need not be monotone and consider other restrictions on clearings. In particular, we study the relationship between clearings that use polite lions and clearings that do not stack lions. We can show the following relation.

► **Theorem 3.6.** *Let G be a graph on n vertices and \mathcal{C} be a polite clearing of G with $k \leq n$ lions. Then, there exists another k -clearing that does not stack lions.*

Proof. For each time step in \mathcal{C} , we describe how to move the corresponding lion (and probably some additional lions) in \mathcal{C}' by avoiding stacking while ensuring that the new strategy developed is a valid clearing of G . Let $V_{\mathcal{C}}(t)$ and $V_{\mathcal{C}'}(t)$ denote the set of cleared vertices at time t in \mathcal{C} and \mathcal{C}' respectively. We show that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t)$ at any time t .

Every lion in \mathcal{C} begins at its *starting vertex*, follows a walk in the graph, and finally ends at its *end vertex*, and remains there for the rest of the clearing. In the process of adapting \mathcal{C} to \mathcal{C}' , some lions are labelled as *retired*. These are lions that are stacked on their end vertices in \mathcal{C} , and thus their position in \mathcal{C}' is irrelevant since they do not help in clearing any more vertices.

Before we describe the overall strategy (or algorithm), we describe the procedure of finding the next location for a lion ℓ_i in \mathcal{C}' . Assume ℓ_i has to be placed at v at time $t \geq 0$ because ℓ_i starts at v or moves to v from a neighboring vertex. Note that since \mathcal{C} is a polite clearing, this is the only lion that needs to be moved in this time step. If there is no lion at v , ℓ_i is placed on v (or moved to v). For the other case, let us assume that there is another lion ℓ_j on v at time t . If ℓ_j is a retired lion, then ℓ_i is placed at v , and ℓ_j is moved to the closest vertex with no lion. We do not really care where ℓ_j is placed since the position of a retired lion is irrelevant in the rest of the clearing. If ℓ_j is not retired, the situation is a bit more delicate. If ℓ_i leaves v before ℓ_j in \mathcal{C} , then we instead place ℓ_j at the next vertex in the path of ℓ_i in \mathcal{C} (which must be a neighbor of v). Then we place ℓ_i at v and switch the naming of the lions. This ensures that we do not stack the lions in this time step, while keeping them on their path. In the other case, namely if ℓ_j leaves v before ℓ_i in \mathcal{C} , we place ℓ_i at v , and place ℓ_j at the next vertex in its path in \mathcal{C} . Finally, in the case when v is the end vertex of both ℓ_i and ℓ_j , we label the lion ℓ_i as retired and place ℓ_i on the closest vertex which has no lion. It could happen that all the neighboring vertices of v are occupied by lions, or the vertex that we intend to move ℓ_i or ℓ_j to is occupied by a lion, in which case we have to follow this procedure recursively and move multiple lions. Note that even though multiple lions might have to be moved through this procedure to simulate one time step of the polite clearing, every step will eventually terminate as we reach retired lions (since $k \leq n$).

To find the starting positions of the lions in \mathcal{C}' we use the procedure just described. The overall strategy follows the procedure as well. We consider one lion movement at a time (since \mathcal{C} is polite) and move lions according to the description above. As already mentioned, this might move more than one lion for an individual movement in \mathcal{C} , but most importantly, this ends for sure. As each movement of a single lion individually terminates, the recursive procedure eventually terminates as well.

It remains to show that \mathcal{C}' is indeed a clearing. For this, observe that whenever the last lion ℓ leaves a vertex u in \mathcal{C} at time t , it also holds that ℓ leaves u in \mathcal{C}' at the same time t . This is because the last lion to leave u is the lion that is positioned at u in \mathcal{C}' by construction. Thus, every vertex that has a lion in \mathcal{C} at some time t also has a lion in \mathcal{C}' at t . Thus, it follows that at every time step t , $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t)$ and hence, \mathcal{C}' is a clearing. ◀

Note that this result implies that the set of graphs that are clearable with polite lions is a subset of the set of graphs that admit non-stacked clearings. Though we were unable to prove it, we believe that the converse of Theorem 3.6 is false, because the time taken by polite lions might be much higher when compared to clearings where multiple lions can be moved simultaneously, even when stacking is not allowed.

► **Open Question 2.** *Is the converse of Theorem 3.6 also true or are there graphs with a non-stacked clearing but no polite clearing?*

When the clearing \mathcal{C} is monotone, the converse is indeed true (follows from Theorem 3.3).

4 Clearable Subgraphs

In the final section of this paper, we study clearings of subgraphs of a k -clearable graph. More formally, let $G = (V, E)$ be a k -clearable graph, and let H be a subgraph of G . It is natural to ask if H also admits a k -clearing (see Question 6.1 in [2]). On one hand, the contamination is restricted due to some missing edges; on the other hand, some edges or paths in $G \setminus H$ might be crucial for the lions to clear the graph.

We show that this question can be answered in the affirmative if the clearing on G is monotone, and that surprisingly, this need not be possible in some other restricted settings.

► **Theorem 4.1.** *Let G be a graph with a monotone k -clearing. Then any connected subgraph $H \subset G$ also admits a (polite) k -clearing.*

Proof. Consider a monotone clearing \mathcal{C} of G using k polite lions (which exists by Theorem 3.3). The idea is to use the same clearing in H with some modifications. In \mathcal{C} , if all lions use edges (and vertices) that are present in H , then the clearing \mathcal{C} restricted to H is a clearing of H and the theorem is proved.

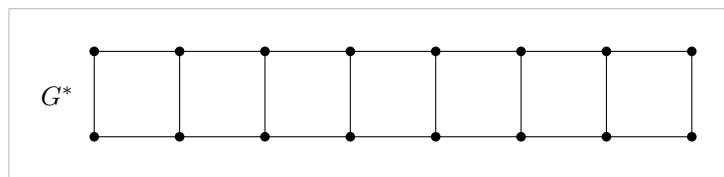
Otherwise, assume for now that $V_G = V_H$. Let t be the first time step such that a lion ℓ_i uses edge $e = (v, w)$ to move from v to w in \mathcal{C} , where $e \in G \setminus H$. Since H is connected, there exists a path $p_{v,w}$ from v to w . We construct a new strategy \mathcal{C}' for H , which mirrors \mathcal{C} until time $t - 1$, then moves lion ℓ_i along the path $p_{v,w}$ instead of moving it along the edge e at time t , and no other lion moves until ℓ_i reaches w . Let $t' > t$ be the time when ℓ_i reaches w in \mathcal{C}' . Let $V_{\mathcal{C}}(t)$ and $V_{\mathcal{C}'}(t')$ denote the set of cleared vertices in \mathcal{C} at time t and \mathcal{C}' at time t' respectively. We claim that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t')$. Note that in \mathcal{C}' , v is the only vertex that might get recontaminated at time t due to Lemma 3.4. However, v gets recontaminated in \mathcal{C}' at time t if and only if v gets recontaminated at time t in \mathcal{C} , and we know that v is not recontaminated in \mathcal{C} since \mathcal{C} is monotone. We further claim that no vertex $u \in V_{\mathcal{C}}(t)$ is recontaminated in \mathcal{C}' after time t while ℓ_i is moving along the path $p_{v,w}$. Assume some vertex $u \in V_{\mathcal{C}}(t)$ is recontaminated in \mathcal{C}' at time $t'' > t$. Since the only vertex that contains a lion in \mathcal{C} at time t but contains no lion in \mathcal{C}' is w , the contamination must have spread to u from w through a path w, u_1, u_2, \dots, u which contains no lion. However, the vertex u_1 must have been contaminated at time t in \mathcal{C} as well since no lion is present on u_1 and \mathcal{C} uses polite lions. The contamination would have then spread to u if the lions were paused at time t in \mathcal{C} , which is a contradiction to Lemma 3.2. This proves that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t')$.

Now, consider the case where $V_H \subsetneq V_G$. In \mathcal{C} , if a lion moves along an edge (v, w) in \mathcal{C} where $v, w \in V_H$, but $(v, w) \notin E_H$, we follow the same strategy as discussed above. Now, let t be the first time when a lion ℓ_i uses edge $e = (v, w)$ to move from v to w in \mathcal{C} , where $v \in V_H$, but $w \notin V_H$. Let the path of ℓ_i in \mathcal{C} be $\dots, v, w, w_1, w_2, \dots, w_k, \dots$ and let w_i be the first vertex in this path after v such that $w_i \in V_H$. We proceed similar to the previous case, and construct a new strategy \mathcal{C}' which mirrors \mathcal{C} until time $t - 1$, and then moves lion ℓ_i to w_i while keeping the other lions stationary. Let ℓ_i reach w_i at time t' . By the same argument as above, we can prove that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t')$. Observe that with this strategy, no lion ever ends up in a vertex of $G \setminus H$, except for the start vertex. However, if the start vertex of a lion lies in $G \setminus H$, we instead move it to the first vertex on its path lying in H .

Thus, we can follow the described strategy and adapt \mathcal{C} to H to construct a new strategy \mathcal{C}' that is a valid clearing of H . Furthermore, note that we only moved one lion at each time step and thus the resulting clearing of the subgraph is polite. ◀

Note that the strategy given in this proof may not be monotone since the vertices that were cleared on a detour might get recontaminated later on. In general, clearings of a subgraph need not be monotone. Figure 5 illustrates a graph G^* that admits a monotone 2-clearing (sweeping from left to right). However, recall graph G_2 described in Section 2.2 (for reference, graph G_3 is illustrated in Figure 2), which is a subgraph of G^* that has no monotone 2-clearing. Note that this recontamination cannot be avoided with the strategy given, not even for induced subgraphs.

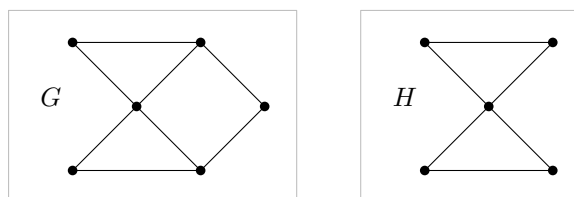
Finally, we illustrate a graph where one of its subgraphs does not admit a 2-clearing with polite and non-stacked lions.



■ **Figure 5** A supergraph of G_2 (graph G_3 is illustrated in Figure 2). G^* is monotone 2-clearable even in the restricted setting of polite and non-stacked lions.

► **Observation 4.2.** *The graph G illustrated in Figure 6 is monotone 2-clearable with polite non-stacked lions. However, its subgraph H is not 2-clearable with polite non-stacked lions.*

Note that H is not only a subgraph but also an induced subgraph of G .



■ **Figure 6** Graph G is 2-clearable with polite and non-stacked lions, whereas subgraph H is not.

5 Conclusion

In the first part of the paper, we studied different types of clearings, namely monotone, polite and non-stacked clearings and we showed some of the relations between them. This gives a good overview over the different restrictions, though a few questions remain open.

In the second part, we focused on the subgraph question raised by Adams et al. [2]. We were able to answer the question in some restricted settings. In the general setting, we believe that there exist graphs with subgraphs that admit no k -clearing. Such a graph might be rather large. A next step in this direction would be to design an algorithm that checks whether a given graph is k -clearable. Such an algorithm, however, may not be easy to find.

References

- 1 H. Adams and G. Carlsson. Evasion paths in mobile sensor networks. *The International Journal of Robotics Research*, 34(1):90–104, 2015. doi:10.1177/0278364914548051.
- 2 H. Adams, L. Gibson, and J. Pfaffinger. Lions and contamination, triangular grids, and cheeger constants. *arXiv*, 2020. arXiv:2012.06702.
- 3 F. Berger, A. Gilbers, A. Grüne, and R. Klein. How many lions are needed to clear a grid? *Algorithms*, 2(3):1069–1086, 2009. doi:10.3390/a2031069.
- 4 A. Bonato and B. Yang. Graph searching and related problems. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 1511–1558, 2013. doi:10.1007/978-1-4419-7997-1_76.
- 5 R. Borie, S. Koenig, and C. Tovey. Section 9.5: Pursuit-evasion problems. In J. Yellen J. Gross and P. Zhang, editors, *Handbook of Graph Theory*, pages 1145–1165. Chapman and Hall/CRC, 2013.
- 6 P. Brass, K. D. Kim, H.-S. Na, and C.-S. Shin. Escaping off-line searchers and a discrete isoperimetric theorem. In *Algorithms and Computation*, pages 65–74, 2007.

- 7 T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics, a survey, 2011. URL: <https://calhoun.nps.edu/handle/10945/45474>.
- 8 V. de Silva and R. Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *The International Journal of Robotics Research*, 25(12):1205–1222, 2006. doi:10.1177/0278364906072252.
- 9 A. Dumitrescu, I. Suzuki, and P. Zylinski. Offline variants of the "lion and man" problem. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, pages 102–111. Association for Computing Machinery, 2007. URL: 10.1145/1247069.1247085.
- 10 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008. Graph Searching. doi:10.1016/j.tcs.2008.02.040.
- 11 R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, 1965.
- 12 A. S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993. doi:10.1145/151261.151263.
- 13 S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*, 14(6):912–925, 1998. doi:10.1109/70.736775.
- 14 T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Applications of Graphs*, pages 426–441, 1978.

Predecessor on the Ultra-Wide Word RAM

Philip Bille  

DTU Compute, Technical University of Denmark, Lyngby, Denmark

Inge Li Gørtz  

DTU Compute, Technical University of Denmark, Lyngby, Denmark

Tord Stordalen  

DTU Compute, Technical University of Denmark, Lyngby, Denmark

Abstract

We consider the predecessor problem on the ultra-wide word RAM model of computation, which extends the word RAM model with *ultrawords* consisting of w^2 bits [TAMC, 2015]. The model supports arithmetic and boolean operations on ultrawords, in addition to *scattered* memory operations that access or modify w (potentially non-contiguous) memory addresses simultaneously. The ultra-wide word RAM model captures (and idealizes) modern vector processor architectures.

Our main result is a simple, linear space data structure that supports predecessor in constant time and updates in amortized, expected constant time. This improves the space of the previous constant time solution that uses space in the order of the size of the universe. Our result is based on a new implementation of the classic x -fast trie data structure of Willard [Inform. Process. Lett. 17(2), 1983] combined with a new dictionary data structure that supports fast parallel lookups.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Ultra-wide word RAM model, predecessor, word-level parallelism

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.18

Related Version *Full Version*: <https://arxiv.org/abs/2201.11550>

Funding *Philip Bille*: Danish Research Council grant DFF-8021-002498.

Inge Li Gørtz: Danish Research Council grant DFF-8021-002498.

Acknowledgements We thank the anonymous reviewers for their comments, which improved the presentation of the article.

1 Introduction

Let S be a set of n w -bit integers. The *predecessor problem* is to maintain S under the following operations.

- $\text{predecessor}(x)$: return the largest $y \in S$ such that $y \leq x$.
- $\text{insert}(x)$: add x to S .
- $\text{delete}(x)$: remove x from S .

The predecessor problem is a fundamental and well-studied data structure problem, both from the perspective of upper bounds [2, 5, 7, 8, 23, 31, 33, 37, 38, 39] and lower bounds [1, 5, 28, 29, 31, 32, 35]. The problem has many applications, for instance integer sorting [2, 3, 23, 25], string sorting [4, 9, 20], and string searching [6, 8, 10, 11, 13]. See Navarro and Rojas-Ledesma [30] for a recent survey.

On the word RAM model of computation, the complexity of the problem is well-understood with the following tight upper and lower bound on the time for operations given by Pătraşcu and Thorup [33].

$$\Theta \left(\max \left[1, \min \left\{ \log_w n, \frac{\log \frac{w}{\log w}}{\log \left(\log \frac{w}{\log w} / \log \frac{\log n}{\log w} \right)}, \log \frac{\log(2^w - n)}{\log w} \right\} \right] \right). \quad (1)$$



© Philip Bille, Inge Li Gørtz, and Tord Stordalen;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 18; pp. 18:1–18:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

From the upper bound perspective, the first branch matches dynamic fusion trees [23], the second branch is based on an extension of the techniques from Beame and Fich [5], and the last branch is based on an extension of dynamic van Emde Boas trees [38]. Note that the lower bound implies that we cannot support operations in constant time for general n and w . Hence, a natural question is if practical models of computation capturing modern hardware can allow us to overcome the superconstant lower bound.

One such model is the *RAM with byte overlap* (RAMBO) by Brodnik et al. [14]. This model extends the word RAM model by adding a set of special words that share bits; flipping a bit in one word will also affect all the other words that share that bit. The precise model is determined by the layout of the shared bits. It is feasible to make hardware based on this model, and prototypes have been built [27]. In the RAMBO model, Brodnik et al. [14] gave a predecessor data structure using constant time per operation with $O(2^w/w)$ space (counting both regular words and shared words). They also gave a randomized version of the solution that uses constant time with high probability and reduces the regular space to $O(n)$ (but still needs $\Omega(2^w/w)$ space for the shared words). In both cases, the total space is near-linear in the size of the universe.

More recently, Farzan et al. [21] introduced the *ultra-wide word RAM model* (UWRAM). The UWRAM extends the word RAM model by adding special *ultrawords* of w^2 bits. The model supports standard boolean and arithmetic operations on ultrawords, as well as *scattered* memory operations that access w words in memory in parallel. The UWRAM model captures (and idealizes) modern vector processing architectures [15, 34, 36] (see Section 2 for details of the model). Farzan et al. [21] showed how to simulate algorithms for the RAMBO model on the UWRAM at the cost of increasing the space by a polylogarithmic factor. Simulating the above RAMBO solution for the predecessor problem, they gave a solution to the predecessor problem on the UWRAM using worst case constant time for all operations and $O(w2^w)$ space.

1.1 Our Results

We revisit the predecessor problem on the UWRAM and show the following main result.

► **Theorem 1.** *Given a set of n w -bit integers, we can construct an $O(n)$ space data structure on a UWRAM which supports predecessor in constant time and insert and delete in amortized expected constant time.*

Compared to the previous result of Farzan et al. [21], Theorem 1 significantly reduces the space from $O(w2^w)$ to linear while maintaining constant time for operations (note that query time is worst-case, while updates are amortized expected).

A key component in our solution is a new dictionary data structure of independent interest that supports fast parallel lookups on the UWRAM. We define the problem as follows. Recall that an ultraword X consists of w^2 bits. We view X as divided into w words of w consecutive bits each, numbered from right to left starting from 0. The i th word in X is denoted $X\langle i \rangle$ (we discuss the model in detail in Section 2). Given a set S of n w -bit integers, the *w -parallel dictionary problem* is to maintain S under the following operations.

- **pMember**(X): return an ultraword I where $I\langle i \rangle = 1$ if $X\langle i \rangle \in S$ and $I\langle i \rangle = 0$ otherwise.
- **insert**(x): Add x to S .
- **delete**(x): Remove x from S .

Thus, **pMember** takes an ultraword X of w integers and returns an ultraword encoding which of these integers are in S . To the best of our knowledge, the w -parallel dictionary problem has not been studied before. We show the following result.

► **Theorem 2.** *Given a set of n w -bit integers, we can construct an $O(n + w)$ -space data structure on a UWRAM which supports `pMember` in worst case constant time and insert and delete in amortized expected constant time.*

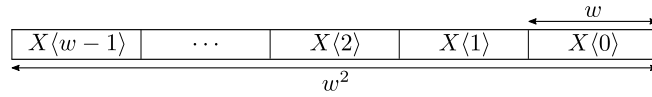
Note that the queries are worst-case constant time, while the updates are amortized expected constant time. The time bounds of Theorem 2 thus match the well-known dynamic perfect hashing structure of Dietzfelbinger et al. [19] (which is also the basis of our solution), except that the queries are parallel. The space is linear except for the additive w term, which is needed even for storing the input to the `pMember` query.

1.2 Techniques

Our results are achieved by novel and efficient parallel implementations of well-known sequential data structures.

Our parallel dictionary structure of Theorem 2 is based on the dynamic perfect hashing structure of Dietzfelbinger et al. [19]. This is a two-level data structure similar to the classic static perfect hashing structure of Fredman et al. [22]. At the first level, a universal hash function partitions the input into smaller subsets, each of which is then resolved at the second level using another universal hash function mapping the elements into sufficiently large tables. The structure supports (sequential) membership queries in worst-case constant time by evaluating the hash functions and navigating the structure accordingly. Updates are supported in amortized expected constant time by carefully rebuilding and rehashing the structure during execution. At any point in time the structure never uses more than $O(n)$ space. We show how to parallelize the evaluation of a universal hash function (the simple and practically efficient *multiply-shift* hash function). Then, using the scattered memory access operations, we show how to access the corresponding entries in the structure in parallel. Our technique requires only small changes to the structure of Dietzfelbinger et al. [19] and we can directly apply their update operations to our solution. Thus, we are able to parallelize the worst-case constant time sequential membership query while maintaining the amortized expected constant update time bound of Dietzfelbinger et al. [19], leading to the bounds of Theorem 2.

Our predecessor data structure of Theorem 1 is based on the *x-fast trie* of Willard [39] combined with our parallel dictionary structure of Theorem 2. The *x-fast trie* consists of the trie T of the binary representation of the input set. Also, at each level i , the structure stores a dictionary containing the length- i prefixes of the input set. In total, this uses $O(nw)$ space. The *x-fast trie* supports predecessor queries in $O(\log w)$ time by binary searching the levels (with the help of the dictionaries) to find the longest common prefix of the query and the input set. Though not designed for it, we can implement updates on the *x-fast trie* in $O(w)$ time by directly updating each level of the dictionary accordingly. Our new predecessor structure, which we call the *xtra-fast trie*, instead stores the compact trie of the binary representation of the input set (i.e., the trie where paths of nodes with a single child are merged into a single edge). We store a dictionary representing the prefixes (similar to in the *x-fast trie*) using our parallel dictionary structure of Theorem 2, but now only for the branching nodes in the compact trie. This reduces the space to $O(n)$. To support predecessor queries for an integer x , we generate all w prefixes of x and apply a parallel membership query on these in the dictionary. We show how to identify the longest match in parallel which in turn allows us to identify the predecessor. In total this takes worst-case constant time for the predecessor query. To handle updates, we show how to modify the trie efficiently using scattered memory access operations and a constant number of dictionary updates, leading to the expected amortized constant time bound of Theorem 1.



■ **Figure 1** The layout of an ultraword X .

In our data structures we only need to store a constant number of ultrawords during the computation. This is important since modern vector processor architectures only have a limited number of ultraword registers.

1.3 Outline

In Section 2 we describe the UWRAM model of computation and some useful procedures. In Sections 3 and 4 we show how to do parallel hash function evaluation and w -parallel dictionaries, proving Theorem 2. Finally, in Section 5 we prove Theorem 1.

2 The Ultra-Wide Word RAM Model

The *word RAM* model of computation [24] consists of an unbounded memory of w -bit words and a standard instruction set including arithmetic, boolean, and bitwise operations (denoted “&”, “|” and “~” for *and*, *or* and *not*) and shifts (denoted “ \gg ” and “ \ll ”) such as those available in standard programming languages (e.g., C). We make the standard assumption that we can store a pointer into the input in a single word and hence $w \geq \log n$, where n is the size of the input, and for simplicity we assume that w is even. We denote the address of x in memory as $\text{addr}(x)$, and the address of an array is the address of its first index. The time complexity of a word RAM algorithm is the number of instructions and the space is the number of words stored by the algorithm.

The *ultra-wide word RAM* (UWRAM) model of computation [21] extends the word RAM model with special *ultrawords* of w^2 bits. As in [21], we distinguish between the *restricted UWRAM* that supports a minimal set of instructions on ultrawords consisting of addition, subtraction, shifts, and bitwise boolean operations, and the *multiplication UWRAM* that additionally supports multiplications. We extend the notation for bitwise operations and shifts to ultrawords. The UWRAM (both restricted and multiplication) also supports contiguous and scattered memory access operations, as described below. The time complexity is the number of instructions (on standard words or ultrawords) and the space complexity is the number of words used by the algorithms, where each ultraword is counted as w words. The UWRAM model captures (and idealizes) modern vector processing architectures [15, 34, 36]. See also Farzan et al. [21] for a detailed discussion of the applicability of the UWRAM model.

2.1 Instructions and Componentwise Operations

Recall that ultrawords consists of w^2 bits. We often view an ultraword X as divided into w words of w consecutive bits each, which we call the *components* of X . We number the components in X from right-to-left starting from 0 and use the notation $X\langle i \rangle$ to denote the i th word in X (see Figure 1). We will also use the notation $X = \langle x_{w-1}, \dots, x_0 \rangle$, denoting that $X\langle i \rangle = x_i$.

We define a number of useful componentwise operations on ultrawords that we will need for our algorithms in the following. Let X and Y be ultrawords. The *componentwise addition* of X and Y , denoted $X + Y$, is the ultraword Z such that $Z\langle i \rangle = X\langle i \rangle + Y\langle i \rangle \bmod 2^w$.

We define *componentwise subtraction*, denoted $X - Y$, and *componentwise multiplication*, denoted XY , similarly. The *componentwise comparison* of X and Y is the ultraword Z such that $Z\langle i \rangle = 1$ if $X\langle i \rangle < Y\langle i \rangle$ and 0 otherwise. Given another ultraword I where each component is either 0 or 1, we define the *componentwise blend* of X , Y , and I to be the ultraword Z such that $Z\langle i \rangle = X\langle i \rangle$ if $I\langle i \rangle = 0$ and $Z\langle i \rangle = Y\langle i \rangle$ if $I\langle i \rangle = 1$.

Except for componentwise multiplication, all of the above componentwise operations can be implemented in constant time on the restricted UWRAM using standard word-level parallelism techniques [12, 24] (see the full version for details on blend). For our purposes, we will need componentwise multiplication as an instruction (for evaluating hash functions in parallel) and thus we include this in the instruction set of the UWRAM. This is the UWRAM model that we will use throughout the rest of the paper. Note that all of the componentwise operations are widely supported directly in modern vector processing architectures. For instance, a componentwise multiplication (e.g., the `vpmullw` operation) is defined in Intel’s AVX2 vector extension [16].

We will need componentwise operations on components that are small constant multiples of w . In particular, we will need a *2w-bit componentwise multiplication* that multiplies $w/2$ components of w bits and returns the $w/2$ resulting components of $2w$ bits. Specifically, let $X = \langle 0, x_{w-2}, \dots, 0, x_2, 0, x_0 \rangle$ and $Y = \langle 0, y_{w-2}, \dots, 0, y_2, 0, y_0 \rangle$, i.e., X and Y store $w/2$ components aligned at the even positions. The *2w-bit componentwise multiplication* is the ultraword $Z = \langle z_{w-2}^+, z_{w-2}^-, \dots, z_2^+, z_2^-, z_0^+, z_0^- \rangle$ where z_i^+ and z_i^- is the leftmost and rightmost w bits, respectively, of the $2w$ -bit product of x_i and y_i . We can implement *2w-bit componentwise multiplication* using standard techniques in constant time on the UWRAM. See the full version for details.

Finally, the UWRAM model supports the `compress` operation that, given X , returns the word that results from concatenating the rightmost bit of each component of X . We do not need the corresponding inverse `spread` operation, defined by Farzan et al. [21].

2.2 Memory Access

The UWRAM supports standard memory access operations that read or write a single word or a sequence of w contiguous words. More interestingly, the UWRAM also supports *scattered* access operations that access w memory locations (not necessarily contiguous) in parallel. Given an ultraword A containing w memory addresses, a *scattered read* loads the contents of the addresses into an ultraword X , such that $X\langle i \rangle$ contains the contents of memory location $A\langle i \rangle$. Given ultrawords X and A a *scattered write* sets the contents of memory location $A\langle i \rangle$ to be $X\langle i \rangle$. Scattered memory accesses captures the memory model used in IBM’s *Cell* architecture [15]. They also appear (e.g., `vpgatherdd`) in Intel’s AVX2 vector extension [16]. Scattered memory access operations were also proposed by Larsen and Pagh [26] in the context of the I/O model of computation. Note that while the addresses for scattered writes must be distinct, we can read simultaneously from the same address. We can use this to efficiently copy x into all w components of an ultraword X . To do so, create the ultraword $\langle 0, \dots, 0 \rangle$ by left-shifting any ultraword by w^2 bits, write x to address 0, and do a scattered read on $\langle 0, \dots, 0 \rangle$. We say that we *load* x into X .

3 Computing Multiply-Shift in Parallel

We show how to efficiently compute a universal hash function in parallel. The *multiply-shift* hashing scheme is a standard and practically efficient family of universal hash functions due to Dietzfelbinger et al. [18]. For some integer $1 \leq c \leq w$, define the class $H_c = \{h_a \mid 0 < a <$

2^w and a is odd} of hash functions where $h_a(x) = (ax \bmod 2^w) \gg (w - c)$. Each function in H_c maps from w -bit to c -bit integers. The class H_c is *universal* in the sense that for any $x \neq y$ and for $h_a \in H_c$ selected uniformly at random, it holds that $P[h_a(x) = h_a(y)] \leq 2/2^c$.

We will show how to evaluate w such functions in constant time. Given $X\langle i \rangle = x_i$, $A\langle i \rangle = a_i$ and $C\langle i \rangle = 2^{c_i}$ where $h_i(x) = (a_i x \bmod 2^w) \gg (w - c_i)$ the goal is to compute $H\langle i \rangle = h_i(x_i)$. To do so we first evaluate the functions in two rounds of $w/2$ functions each, and then combine the results.

Step 1: Evaluate the hash function on the even indices. We construct an ultraword H_{even} containing all the values of $h_i(x_i)$ at all even indices i . First construct the ultrawords

$$C' = \langle 0, 2^{c_{w-2}}, \dots, 0, 2^{c_0} \rangle$$

$$T' = \langle 0, a_{w-2}x_{w-2} \bmod 2^w, \dots, 0, a_0x_0 \bmod 2^w \rangle.$$

To do so, we do componentwise multiplication of C with the constant $M = \langle 0, 1, \dots, 0, 1 \rangle$ and componentwise multiplications of A , X , and M . Then, we do a $2w$ -bit multiplication of C' and T' and right shift the result by w . This produces the ultraword

$$H_{\text{even}} = \langle \star, (a_{w-2}x_{w-2} \bmod 2^w) \gg (w - c_{w-2}), \dots, \star, (a_0x_0 \bmod 2^w) \gg (w - c_0) \rangle$$

Thus, all even indices in H_{even} store the resulting hash values of the integers at the even indices in the input. We will not need the values in the odd indices (resulting from the $2w$ -bit multiplication and the right shift) and therefore these are marked with a wildcard symbol \star .

Step 2: Evaluate the hash function on the odd indices. Symmetrically, we now construct the ultraword H_{odd} containing $h_i(x_i)$ at all odd indices i . To do so, repeat step 1 and modify the shifting to align the computation for the odd indices. More precisely, right shift X , C and A by w and repeat step 1, then left shift the result by w to align the results back to the odd positions. This produces the ultraword

$$H_{\text{odd}} = \langle (a_{w-1}x_{w-1} \bmod 2^w) \ll c_{w-1}, \star, \dots, (a_1x_1 \bmod 2^w) \ll c_1, \star \rangle$$

Step 3: Combine the results. Finally, we combine the results by blending H_{even} and H_{odd} using $I = \langle 1, \dots, 1 \rangle - M$, producing the ultraword H of the even indices of H_{even} and the odd indices of H_{odd} .

This takes constant time since componentwise multiplication, $2w$ -bit multiplication, shifting, blending, loading 1 into $\langle 1, \dots, 1 \rangle$, and componentwise subtraction all run in constant time. Hence, we can evaluate each case of $w/2$ hash functions in constant time and combine the results in constant time. In summary, we have the following result.

► **Lemma 3.** *Given $X\langle i \rangle = x_i$, $A\langle i \rangle = a_i$, $C\langle i \rangle = 2^{c_i}$, and the constant $M = \langle 0, 1, \dots, 0, 1 \rangle$ we can evaluate each of the w multiply-shift hash functions $h_i(x) = (a_i x \bmod 2^w) \gg (w - c_i)$ by computing the ultraword $H = \langle h_{w-1}(x_{w-1}), \dots, h_0(x_0) \rangle$ in constant time on a UWRAM.*

4 The w -Parallel Dictionary

We now show how to construct the w -parallel dictionary of Theorem 2. To do so we use a dictionary by Dietzfelbinger et al. that implements a dynamic perfect hashing strategy [19]. Their dictionary already supports insert and delete in amortized expected constant time. Furthermore, it supports sequential member queries (i.e. “is $x \in S$ ”) in worst case constant time. We will show that we can use scattered memory operations to run w member queries simultaneously, thus implementing pMember in constant time.

4.1 Dynamic Perfect Hashing

In this section we briefly describe the contents of the data structure of Dietzfelbinger et al. [19]. Note that we use the multiply-shift hashing scheme, while they use another class of universal hash functions. Multiply-shift satisfies all the necessary constraints and the analysis from [19] still works. It does however incur a multiplicative, constant space overhead for our arrays since the range of a multiply-shift function is a power of two.

The main idea of the data structure is as follows. Let S be a set of w -bit integers. Choose $h \in H_c$ and partition S into $2^c = \Theta(n)$ sets S_0, \dots, S_{2^c-1} where $S_i = \{x \mid x \in S \text{ and } h(x) = i\}$. Each set S_i is stored in a separate array using a hash function h_i . Dietzfelbinger et al. show how to implement the operations `insert` and `delete` such that they maintain that h_i has no collisions on S_i .

The data structure consists of the following.

- For each S_i , store an array T_i of size 2^{c_i} . Let $h_i(x) = (a_i x \bmod 2^w) \gg (w - c_i)$. For each $x \in S_i$ let $T_i[h_i(x)] = x$, i.e. the position that x hashes to stores x . If there is no $x \in S_i$ that hashes to j , then $T_i[j] = 2^{w-1}$ if $j = 0$ and $T_i[j] = 0$ otherwise. We claim that $h_i(0)$ is always zero and $h_i(2^{w-1})$ is never zero, so it follows from this construction that $x \in S_i$ if and only if $T_i[h_i(x)] = x$. We have that $h_i(2^{w-1})$ is not zero because

$$h_i(2^{w-1}) = (a_i 2^{w-1} \bmod 2^w) \gg (w - c_i) = 2^{w-1} \gg (w - c_i) \geq 1.$$

The second step follows since a_i is odd; then $a_i 2^{w-1} = 2^{w-1} + (a_i - 1)2^{w-1}$, and the latter term is 0 modulo 2^w since $a_i - 1$ is even. The last step follows because $c_i \geq 1$.

- An array T of size 2^c . At index $T[i]$ we store the 5-tuple $(\text{addr}(T_i), 2^{c_i}, a_i, \star, \star)$ where \star are book-keeping values used by `insert` and `delete`. Note that 2^{c_i} and a_i encode h_i .
- The integers a and 2^c representing the top-level hash function $h(x) = (ax \bmod 2^w) \gg (w - c)$, as well as $\text{addr}(T)$.

It follows from this construction that $x \in S$ if and only if $T_i[h_i(x)] = x$ where $i = h(x)$. Dietzfelbinger et al. show that the data structure uses linear space, that `member` runs in worst-case constant time, and that `insert` and `delete` run in amortized expected constant time [19].

Extending the Data Structure. We extend this data structure by storing the constant $M = \langle 0, 1, \dots, 0, 1, 0, 1 \rangle$ from Section 3 used to evaluate multiply-shift functions in parallel. This increases the space of the data structure to $O(n + w)$. Note that linear space in w is needed even to store the input to a `pMember` query.

4.2 Parallel Queries

In this section, we begin by describing a single `member` query, before we show how to run w copies of the `member` query in parallel to support `pMember`. We compute `member(x)` as follows.

1. Using a and 2^c , compute $j = h(x)$.
2. Let $q = \text{addr}(T) + 5j = \text{addr}(T[j])$ (recall that each index in T stores five words). Read the values stored at q , $q + 1$ and $q + 2$ to get respectively $\text{addr}(T_j)$, 2^{c_j} and a_j , the first three words stored at $T[j]$. Compute $k = h_j(x)$.
3. Check whether the value stored at $\text{addr}(T_j) + k = \text{addr}(T_j[k])$ is equal to x .

The parallel algorithm runs this algorithm for all w inputs simultaneously. Given $X = \langle x_{w-1}, \dots, x_0 \rangle$ we implement `pMember(X)` as follows. Each of the steps below executes the corresponding step above in parallel for each of the w inputs.

Step 1: Evaluate the top-level hash function. Load the two ultrawords $A = \langle a, \dots, a \rangle$ and $C = \langle 2^c, \dots, 2^c \rangle$. Compute the ultraword $J = \langle h(x_{w-1}), \dots, h(x_0) \rangle$ using the multiply-shift algorithm of Lemma 3.

Step 2: Evaluate each of the second-level hash functions. Load $F = \langle 5, \dots, 5 \rangle$ and $P = \langle \text{addr}(T), \dots, \text{addr}(T) \rangle$. Compute $Q = P + FJ$. Then $Q\langle i \rangle = \text{addr}(T) + 5J\langle i \rangle = \text{addr}(T[J\langle i \rangle])$. Do scattered reads of Q , $Q + \langle 1, \dots, 1 \rangle$, and $Q + \langle 2, \dots, 2 \rangle$ to produce the ultrawords P' , C' , and A' . We have that

$$\begin{aligned} P' &= \langle \text{addr}(T_{J\langle w-1 \rangle}), \dots, \text{addr}(T_{J\langle 0 \rangle}) \rangle \\ C' &= \langle 2^{c_{J\langle w-1 \rangle}}, \dots, 2^{c_{J\langle 0 \rangle}} \rangle \\ A' &= \langle a_{J\langle w-1 \rangle}, \dots, a_{J\langle 0 \rangle} \rangle \end{aligned}$$

Compute the ultraword $K = \langle h_{J\langle w-1 \rangle}(x_{w-1}), \dots, h_{J\langle 0 \rangle}(x_0) \rangle$ using the multiply-shift algorithm of Lemma 3.

Step 3: Check whether the inputs are present in the dictionary. Do a scattered read of $P' + K$ and name the result R . Then $R\langle i \rangle = T_j[h_j(x_i)]$ where $j = h(x_i)$. Return the result I of componentwise equality between X and R . That is

$$I\langle i \rangle = \begin{cases} 1 & \text{if } X\langle i \rangle = R\langle i \rangle \\ 0 & \text{otherwise} \end{cases}$$

Evaluating the hash functions in steps 1 and 2 takes constant time according to Lemma 3. The remaining operations are scattered reads, loads and componentwise operations, all of which run in constant time. Since there is only a constant number of operations, `pMember` runs in constant time. This concludes the proof of Theorem 2.

4.3 Satellite Data

Suppose we associate some value `data(x)` with each $x \in S$. We extend the data structure to support the following operation, where $X = \langle x_{w-1}, \dots, x_0 \rangle$ as above.

■ `pRetrieve(X)`: returns a pair (I, D) where I is the result of `pMember(X)` and

$$D\langle i \rangle = \begin{cases} \text{addr}(\text{data}(x_i)) & \text{if } I\langle i \rangle = 1, \text{ i.e. if } x_i \in S \\ \text{undefined} & \text{otherwise} \end{cases}$$

We return `addr(data(x))` instead of `data(x)` since the data would not fit into an ultraword if `data(x)` requires more than one word to store.

We extend the data structure as follows to support `pRetrieve`. Store two words for each index in T_i . For each $x \in S_i$, the first word in $T_i[h_i(x)]$ stores x and the second stores `addr(data(x))`. The remaining entries store either 0 or 2^{w-1} , as above.

To do the retrieval, first compute $I = \text{pMember}(X)$. However, in step 3, multiply K by $\langle 2, \dots, 2 \rangle$ before the scattered read since each index in T_i now stores two words. Also, add $\langle 1, \dots, 1 \rangle$ to $P' + \langle 2, \dots, 2 \rangle K$ and do a scattered read to compute the ultraword D . The space of the data structure remains $O(n + w)$ (assuming that `data(x)` uses constant space), and `pRetrieve` runs in constant time.

5 The *xtra-fast* Trie

In this section we describe our data structure, the *xtra-fast* trie, which supports predecessor in worst case constant time and insert and delete in amortized expected constant time.

Below we assume that we have keys of $w - 1$ bits each and give a solution that uses $O(n + w)$ space. At the end of this section we will reduce the space to $O(n)$ and extend the solution to w -bit keys, proving Theorem 1.

5.1 Data Structure

Consider the compacted trie T over the binary representation of the elements in S . For each node $v \in T$ define $\text{str}(v)$ to be the bitstring encoded by the path from the root to v in T . Also let $\min(v)$ and $\max(v)$ be the smallest and largest leaves in the subtree of v , respectively. By $\min(v)$ and $\max(v)$ we refer both to a leaf and to the value the leaf represents.

For each edge $(u, v) \in T$, let $\text{label}(u, v)$ be $\text{str}(u)$ followed by the first bit on the edge (u, v) . Define $\text{key}(u, v)$ to be $\text{label}(u, v)$ followed by a single 1-bit and $w - |\text{label}(u, v)| - 1$ zeroes. Note that $|\text{key}(u, v)| = w$ and that the keys of two distinct edges in T always differ. See Figure 2 for an example.

We define the *exit edge* for an integer x to be the edge in T where the match of x ends. In other words, it is the edge $(u, v) \in T$ such that $\text{label}(u, v)$ is a prefix of x and $|\text{label}(u, v)|$ is maximum. See Figure 2 for an example. It is possible that x has no exit edge if the root has fewer than two children.

Our data structure consists of the following:

- A sorted, doubly linked list L of the leaves of T , i.e., the elements of S .
- A dictionary D supporting parallel queries using Theorem 2. For each edge $(u, v) \in T$ we store an entry in D with the key $\text{key}(u, v)$ and $\text{data}(u, v) = (\text{addr}(\min(v)), \text{addr}(\max(v)))$. Here, $\text{addr}(\min(v))$ and $\text{addr}(\max(v))$ are the addresses to the corresponding elements in L , and we denote the addresses to $\min(v)$ and $\max(v)$ as the *min-* and *max-pointer* of (u, v) .
- The two ultraword constants M' and H described in the next section.

Storing L and the ultraword constants takes $O(n + w)$ space combined. Since T is compacted there are $O(n)$ entries in D , so by Theorem 2 the dictionary also uses $O(n + w)$ space.

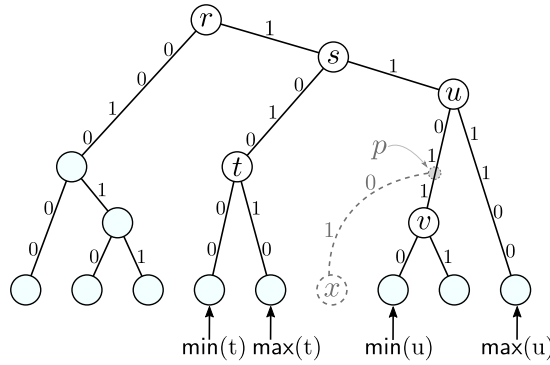
5.2 Predecessor Queries

The main idea of the predecessor query for x is to first find the exit edge of x by simultaneously searching for all prefixes of x in D . Then we use the min- and max-pointer of the exit edge to find the predecessor of x . If x has no exit edge, then the root does not have an outgoing edge matching the leftmost bit of x . If the leftmost bit of x is 1, the predecessor of x is the largest leaf in the left subtree of the root, and otherwise x has no predecessor. Assuming that x has an exit edge, the procedure has three steps.

Step 1: Compute all prefixes of x . Let $b_{w-2}b_{w-3}\cdots b_0$ be the binary representation of x of length $w - 1$. We compute the ultraword

$$\bar{X} = \langle b_{w-2}b_{w-3}\cdots b_01, b_{w-2}b_{w-3}\cdots b_010, \dots, 10\cdots 0 \rangle.$$

That is, $\bar{X}\langle i \rangle$ contains the prefix of x of length i followed by a 1-bit and $w - i - 1$ zeroes. Thus, for any edge $(u, v) \in T$ such that $\text{label}(u, v)$ is the length- i prefix of x , we have $\bar{X}\langle i \rangle = \text{key}(u, v)$. We compute \bar{X} as follows.



■ **Figure 2** An *xtra-fast* trie for $S = \{001000, 001010, 001011, 101000, 101010, 110110, 110111, 111100\}$. The dashed edge and nodes illustrate how the trie would change if $x = 110101$ were inserted. The exit edge for x is (u, v) since we match the bitstring 1101 but do not match the next 1 on (u, v) . Similarly, the exit edge for 100100 is (s, t) . We have that $\text{key}(u, v) = \text{label}(u, v)\underline{1000} = 110\underline{1000}$ where the underlined part is what we append to the labels to disambiguate the keys. Similarly, $\text{key}(r, s) = 1\underline{100000}$ and $\text{key}(s, t) = 10\underline{10000}$. The dictionary entry of (s, u) has $\text{key}(s, u) = 11\underline{10000}$, and the min- and max-pointer of (s, u) are $\text{addr}(\min(u))$ and $\text{addr}(\max(u))$. Similarly, the min-pointer of (r, s) is to $\min(s) = \min(t)$ and the max-pointer is to $\max(s) = \max(u)$. Note that if we insert x we would have to update the min-pointer of (s, u) , since $x < \min(v)$. However, the min-pointer of (r, s) remains unchanged since $\min(t) < x$.

Let M' be the constant such that $M'\langle i \rangle$ consists of i consecutive 1-bits followed by $w - i$ consecutive 0-bits. Let H be the constant where the $(i + 1)$ th leftmost bit in $H\langle i \rangle$ is 1 and the remaining bits are zeroes. First load x into X such that $X = \langle x, x, \dots, x \rangle$. Then compute $\bar{X} = (X \& M') \mid H$.

Step 2: Find the exit edge (u, v) of x . First do $(I, P) = \text{pRetrieve}(\bar{X})$ on D . Then compute $c = \text{compress}(I)$ such that the i th rightmost bit in c is 1 if $I\langle i \rangle = 1$ and zero otherwise. Note that x has no exit edge if $c = 0$. Find the index k of the leftmost bit in c that is 1 (see [23]). Then $\bar{X}\langle k \rangle = \text{key}(u, v)$ where (u, v) is the exit edge of x . Furthermore, the values stored at the addresses $P\langle k \rangle$ and $P\langle k \rangle + 1$ are the min- and max-pointers of (u, v) , respectively.

Step 3: Find the predecessor of x . Use the min- and max-pointer of (u, v) found in step 2 to retrieve $\min(v)$ and $\max(v)$. If $x \geq \max(v)$ then return $\max(v)$, otherwise return the element immediately left of $\min(v)$ in L . Note that there might not be an element immediately left of $\min(v)$ if x is smaller than than everything in S , in which case x has no predecessor.

Since we search for all prefixes of x and take the edge corresponding to the longest prefix found, we find the exit edge (u, v) of x . If $x \in S$, then $x = v = \max(v)$ and we correctly return that x is the predecessor of itself. If $x \notin S$ then the path to where x would have been located if it were in T branches off (u, v) either to the left (if $x < \min(v)$) or right (if $x > \max(v)$). In the first case, $\text{predecessor}(x)$ is the element located immediately left of $\min(v)$ in T , and in the second case $\text{predecessor}(x)$ is $\max(v)$.

By Theorem 2 the parallel dictionary query in step 2 takes worst case constant time. Finding the leftmost bit that is 1 takes constant time on the word RAM [23]. The remaining operations are standard operations available in the model, so the procedure runs in constant time.

5.3 Insertions

The main idea of the insertion procedure is as follows. Since T is compacted, inserting a new leaf x will cause only a constant number of edges to be inserted and removed, so we can make these changes sequentially. Furthermore, some of the at most $w - 1$ edges on the path from the root to x might have their min- or max-pointers changed, and we will update these edges in parallel.

Consider inserting $x = 110101$ in the trie in Figure 2. When x is inserted we add a new leaf for x , as well as a new node p at the location where the path to x branches off the exit edge (u, v) of x . This removes the edge (u, v) , but adds the three new edges (u, p) , (p, x) and (p, v) . Furthermore, we must update the min-pointer of (s, u) , because $\min(v)$ was replaced by x as the smallest leaf under u . On the other hand, we do not update the min-pointer of (r, s) because $\min(t)$ is smaller than x . Note that we do not explicitly store internal nodes and therefore do not add p anywhere in the data structure.

We now describe the insertion procedure. First we note that if x does not have an exit edge it is because the root does not have an outgoing edge which shares the same leftmost bit as x . This case is easily solved by adding an edge from the root to the new leaf x and adding x to either the start or end of L . We will now assume that x has an exit edge, and also that x branches off its exit edge to the left; the other case is symmetric.

Step 1: Find the predecessor of x . Do a predecessor query as described in Section 5.2, which determines

- The predecessor of x in L .
- The exit edge (u, v) of x , $\text{label}(u, v)$ and $\text{data}(u, v) = (\text{addr}(\min(v)), \text{addr}(\max(v)))$.
- The result (I, P) of $\text{pRetrieve}(\overline{X})$ on D .

Step 2: Insert x in L . Insert x immediately to the right of its predecessor in L .

Step 3: Update edges. We insert (u, p) , (p, x) and (p, v) and remove (u, v) from D . We find the labels of the three edges to insert as follows. We have that $\text{label}(u, p) = \text{label}(u, v)$ since (u, p) is the edge (u, v) shortened by adding the node p and since only the first character of the edge affects the label. By definition, $\text{label}(p, x)$ and $\text{label}(p, v)$ consist of $\text{str}(p)$ with a zero and a one appended, respectively. We compute $\text{str}(p)$ by finding the longest common prefix \hat{p} of x and $\min(v)$. To do so, do bitwise XOR between x and $\min(v)$ and find the index k of the leftmost bit that is 1 in the result (see [23]). Now k indicates the leftmost bit where x and $\min(v)$ differ. To extract the longest common prefix compute $\hat{p} = x \ \& \ \sim((1 \ll (k + 1)) - 1)$. Given the labels we can easily construct the keys for the edges.

We now construct the satellite data for the edges. Both the min- and max-pointer for (p, x) are $\text{addr}(x)$ since x is a leaf. For (p, v) they are $\text{addr}(\min(v))$ and $\text{addr}(\max(v))$, which were determined during the predecessor query. Finally, the min-pointer for (u, p) is $\text{addr}(x)$ and the max-pointer is $\text{addr}(\max(v))$.

Step 4: Update min-pointers. We update the min-pointers for the edges on the path from the root to u that are incorrect after inserting x . Note that inserting x cannot invalidate any max-pointers since we assumed that x branched off its exit edge to the left. The edges that must be updated are exactly those that have a min-pointer to $\min(v)$, since x has replaced $\min(v)$ as the smallest leaf under u .

18:12 Predecessor on the Ultra-Wide Word RAM

Consider the result (I, P) from the pRetrieve query. We begin by setting $I\langle k' \rangle = 0$ for the index k' corresponding to the exit edge (u, v) of x (we know k' from the predecessor query). The indices in I that now contain 1 indicate the edges on the path from the root to u .

Next we identify the edges that needs to be updated by creating I' where $I'\langle i \rangle = 1$ if and only if both $I\langle i \rangle = 1$ and what is stored at address $P\langle i \rangle$ is the address of $\min(v)$. To do so, first do a scattered read of P and store the result in M . Now M contains $\text{addr}(\min(b))$ for each edge (a, b) on the path to u .¹ Note the value of $P\langle i \rangle$ is arbitrary if $I\langle i \rangle = 0$, i.e. if no edge has the length- i prefix of x as its label. Load $\text{addr}(\min(v))$ into the ultraword V . Let E be the result of componentwise equality between M and V . Then $E\langle i \rangle = 1$ if and only if what is stored at address $P\langle i \rangle$ is $\text{addr}(\min(v))$. Finally compute $I' = I \& E$.

Now we use P and I' to update the incorrect min-pointers. First, load the address of the node for x into U . Then compute B by blending M (the result of the scattered read of P) and U conditioned on I' such that

$$B\langle i \rangle = \begin{cases} M\langle i \rangle & \text{if } I'\langle i \rangle = 0 \quad (\text{i.e. the value already at the address } P\langle i \rangle) \\ U\langle i \rangle & \text{if } I'\langle i \rangle = 1 \quad (\text{i.e. the address of } x) \end{cases}$$

Finally, do a scattered write of B to the addresses in P . Hence, what is stored at the address $P\langle i \rangle$ remains the same if $I'\langle i \rangle = 0$ and is replaced by the address of x otherwise.

The predecessor query in step 1 takes constant time. The operations in step 2 and step 4 are all standard RAM or UWRAM operations, except for finding the leftmost 1-bit which takes constant time [23]. The dictionary updates in step 3 run in amortized expected constant time by Theorem 2. Since the rest of step 3 consists of standard operations, the running time for insertions is amortized expected constant.

5.4 Deletions

The deletion procedure is essentially the inverse of the insertion procedure. We assume that x is the left child of its parent p ; the other case is symmetric.

Step 1: Find x . Do a predecessor query for x . Since $x \in S$, the predecessor of x is itself. This determines

- The position of x in L .
- The exit edge (p, x) for x , along with $\text{label}(p, x)$. Since $x \in S$, this edge must end in the leaf for x .
- The result (I, P) of pRetrieve(\bar{X}) on D .

Step 2: Update min-pointers. If p is the root (i.e. if $|\text{label}(p, x)| = 1$) we remove the edge (p, x) from D and remove x from L which completes the deletion of x . Otherwise p is an internal node and must have another child which we denote by v . Consider the edges on the path to p . Any min-pointer to x should be replaced by the address of $\min(v)$, since $\min(v)$ is the successor of x and also in the subtree of all of these edges. We find $\min(v)$ in the node immediately right of x in L . As we did for insertions, replace any min-pointer that is an address of x by the address of $\min(v)$ in parallel using I and P .

¹ If x branched off to the right of its exit edge, we would do a scattered read of $P + \langle 1, \dots, 1 \rangle$ to load the max-pointers instead of min-pointers.

Step 3: Delete edges. We delete (p, x) and (p, v) from D . Determine $\text{label}(p, v)$ by flipping the last bit in $\text{label}(p, x)$. Using the labels we easily find the keys. Note that we do not explicitly delete the edge (u, p) or insert the edge (u, v) . These two edges share the same key, and the min-pointer of (u, p) was changed to the address of $\min(v)$ in step 2.

Step 4: Update L . Remove x from L .

Steps 1, 2 and 4 all take constant time (see Sections 5.2 and 5.3). The two deletions in step 3 take amortized constant time according to Theorem 2. The remainder of step 3 takes constant time, so deletions run in amortized expected constant time.

5.5 Reducing to Linear Space and Supporting w -bit Keys

Here, we reduce the space to $O(n)$ and show how to support w -bit keys, concluding the proof of Theorem 1.

The $O(w)$ term in the space bound above is due to the w -parallel dictionary D and $O(1)$ ultraword constants. To avoid this when $n = o(w)$, we will initially support predecessor, insert and delete using the *dynamic fusion tree* by Pătraşcu and Thorup [33] (based on the fusion tree by Fredman and Willard [23]), which uses linear space and supports all three operations in constant time for sets of size $w^{O(1)}$. Simultaneously, we build the ultraword constants we need over the course of $\Theta(w)$ insertions, maintaining linear space. When $n \geq w$, the constants have been built and we move all elements into the trie. If at any point $n \leq w/2$, we move all elements from the trie into a fusion tree and remove the trie and the ultraword constants, leaving us with linear space and $\Theta(w)$ insert operations in which to rebuild the constants. Updates still run in amortized expected constant time since we always do $\Omega(w)$ updates before we move $O(w)$ elements.

To extend the solution to work with w -bit keys, we partition the input set S into S_0 and S_1 where $S_i = \{s \mid s \in S \text{ and the leftmost bit of } s \text{ is } i\}$, and store an x tra-fast trie for each set. Suppose the leftmost bit of an integer x is i . An insert, delete or predecessor operation on x is performed on the data structure for S_i . Additionally, if $i = 1$ and the predecessor query on S_1 returns that x has no predecessor, we return the largest element in S_0 , or report that x has no predecessor if S_0 is empty.

6 Conclusion and Open Problems

We have studied the predecessor problem on the UWRAM model of computation. We have given a linear space data structure that supports predecessor queries in worst case constant time and updates in amortized expected constant time.

Furthermore, we have shown how to implement a w -parallel dictionary on the UWRAM. The dictionary supports w simultaneous membership queries in worst case constant time and individual updates in amortized expected constant time.

We wonder if it is possible to achieve constant time with high probability for all operations in the predecessor problem. The limiting factor for our solution is the time for updates in the w -parallel dictionary. There are dictionaries that achieve constant time with high probability for all operations in the word RAM model, e.g. [17]. However, such dictionaries seem to require hash functions that are difficult to evaluate in parallel on the UWRAM. For instance, [17] uses the modulo operator, for which we cannot see an obvious way to make a component-wise version.

References

- 1 Miklós Ajtai. A lower bound for finding predecessors in Yao’s cell probe model. *Comb.*, 8(3):235–247, 1988. doi:10.1007/BF02126797.
- 2 Arne Andersson. Faster deterministic sorting and searching in linear space. In *Proc. 37th FOCS*, pages 135–141, 1996. doi:10.1109/SFCS.1996.548472.
- 3 Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? *J. Comput. Syst. Sci.*, 57(1):74–93, 1998. doi:10.1006/jcss.1998.1580.
- 4 Lars Arge, Paolo Ferragina, Roberto Grossi, and Jeffrey Scott Vitter. On sorting strings in external memory (extended abstract). In *Proc. 29th STOC*, pages 540–548, 1997. doi:10.1145/258533.258647.
- 5 Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci.*, 65(1):38–72, 2002. doi:10.1006/jcss.2002.1822.
- 6 Djamel Belazzougui. Worst-case efficient single and multiple string matching on packed texts in the word-RAM model. *J. Discrete Algorithms*, 14:91–106, 2012. doi:10.1016/j.jda.2011.12.011.
- 7 Djamel Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Monotone minimal perfect hashing: searching a sorted table with $O(1)$ accesses. In *Proc. 20th SODA*, pages 785–794, 2009. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496856>.
- 8 Djamel Belazzougui, Paolo Boldi, and Sebastiano Vigna. Dynamic z-fast tries. In *Proc. 17th SPIRE*, pages 159–172, 2010. doi:10.1007/978-3-642-16321-0_15.
- 9 Michael A. Bender, Martin Farach-Colton, and Bradley C. Kuszmaul. Cache-oblivious string B-trees. In *Proc. 25th PODS*, pages 233–242, 2006. doi:10.1145/1142351.1142385.
- 10 Philip Bille, Mikko Berggren Ettienne, Inge Li Gørtz, and Hjalte Wedel Vildhøj. Time-space trade-offs for Lempel-Ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018. doi:10.1016/j.tcs.2017.12.021.
- 11 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *Proc. 28th CPM*, pages 6:1–6:11, 2017. doi:10.4230/LIPIcs.CPM.2017.6.
- 12 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Partial sums on the ultra-wide word RAM. *Theor. Comput. Sci.*, 905:99–105, 2022. Announced at TAMC 2020. doi:10.1016/j.tcs.2022.01.002.
- 13 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi:10.1137/130936889.
- 14 Andrej Brodnik, Svante Carlsson, Michael L. Fredman, Johan Karlsson, and J. Ian Munro. Worst case constant time priority queue. *J. Syst. Softw.*, 78(3):249–256, 2005. doi:10.1016/j.jss.2004.09.002.
- 15 Thomas Chen, Ram Raghavan, Jason N. Dale, and Eiji Iwata. Cell Broadband engine architecture and its first implementation - A performance view. *IBM J. Res. Dev.*, 51(5):559–572, 2007. doi:10.1147/rd.515.0559.
- 16 Intel Corporation. Intel® advanced vector extensions programming reference. *Intel Corporation*, 2011.
- 17 Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proc. 17th ICALP*, pages 6–19, 1990. doi:10.1007/BFb0032018.
- 18 Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25(1):19–51, 1997. doi:10.1006/jagm.1997.0873.
- 19 Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994. doi:10.1137/S0097539791194094.
- 20 Martin Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th FOCS*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

- 21 Arash Farzan, Alejandro López-Ortiz, Patrick K. Nicholson, and Alejandro Salinger. Algorithms in the ultra-wide word model. In *Proc. 12th TAMC*, pages 335–346, 2015. doi:10.1007/978-3-319-17142-5_29.
- 22 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
- 23 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.
- 24 Torben Hagerup. Sorting and searching on the word RAM. In *Proc. 15th STACS*, pages 366–398, 1998. doi:10.1007/BFb0028575.
- 25 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.
- 26 Kasper Green Larsen and Rasmus Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proc. 23rd SODA*, pages 583–592, 2012. doi:10.1137/1.9781611973099.49.
- 27 R. Leben, M. Miletic, M. Špegel, A. Torst, A. Brodnik, and K. Karlsson. Design of high performance memory module on PC100. In *Proc. Electrotechnical and Computer Science Conference (ERK)*, pages 75–78, 1999.
- 28 Peter Bro Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proc. 26th STOC*, pages 625–634, 1994. doi:10.1145/195058.195415.
- 29 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, 1998. doi:10.1006/jcss.1998.1577.
- 30 Gonzalo Navarro and Javiel Rojas-Ledesma. Predecessor search. *ACM Comput. Surv.*, 53(5):105:1–105:35, 2020. doi:10.1145/3409371.
- 31 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th STOC*, pages 232–240, 2006. doi:10.1145/1132516.1132551.
- 32 Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *Proc. 18th SODA*, pages 555–564, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283443>.
- 33 Mihai Pătraşcu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *Proc. 55th FOCS*, pages 166–175, 2014. doi:10.1109/FOCS.2014.26.
- 34 James Reinders. Intel® AVX-512 instructions. Intel® Corporation, 2013.
- 35 Pranab Sen and Srinivasan Venkatesh. Lower bounds for predecessor searching in the cell probe model. *J. Comput. Syst. Sci.*, 74(3):364–385, 2008. doi:10.1016/j.jcss.2007.06.016.
- 36 Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanaël Prémillieu, Alastair Reid, Alejandro Rico, and Paul Walker. The ARM scalable vector extension. *IEEE Micro*, 37(2):26–39, 2017. doi:10.1109/MM.2017.35.
- 37 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.
- 38 Peter van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Math. Syst. Theory*, 10:99–127, 1977. doi:10.1007/BF01683268.
- 39 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Inform. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

An Optimal Algorithm for Product Structure in Planar Graphs

Prosenjit Bose  

School of Computer Science, Carleton University, Ottawa, Canada

Pat Morin  

School of Computer Science, Carleton University, Ottawa, Canada

Saeed Odak 

Department of Computer Science and Electrical Engineering, University of Ottawa, Canada

Abstract

The *Product Structure Theorem* for planar graphs (Dujmović et al. *JACM*, **67**(4):22) states that any planar graph is contained in the strong product of a planar 3-tree, a path, and a 3-cycle. We give a simple linear-time algorithm for finding this decomposition as well as several related decompositions. This improves on the previous $O(n \log n)$ time algorithm (Morin. *Algorithmica*, **85**(5):1544–1558).

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Graph algorithms

Keywords and phrases Planar graphs, product structure

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.19

Related Version *Full Version*: <https://arxiv.org/abs/2202.08870>

Funding This research was partly funded by NSERC.

Acknowledgements This research was initiated at the BIRS 21w5235 Workshop on Graph Product Structure Theory, held November 21–26, 2021 at the Banff International Research Station. The authors are grateful to the workshop organizers and participants for providing a stimulating research environment. We are especially grateful to Vida Dujmović for sharing Theorem 1.b with us.

1 Introduction

For two graphs G and X , the notation $G \subseteq X$ denotes that G is isomorphic to some subgraph of X . The following *planar product structure theorems* have recently been used as a key tool in resolving a number of longstanding open problems on planar graphs, including queue number [8], nonrepetitive chromatic number [11], adjacency labelling [10], universal graphs [12], p -centered colouring [6], and vertex ranking [5].¹²

► **Theorem 1** (Dujmović et al. [8], Ueckerdt et al. [18]). *For any planar graph G , there exists:*

- (a) *a planar graph H of treewidth at most 3 and a path P such that $G \subseteq H \boxtimes P \boxtimes K_3$ [8];*
- (b) *a planar graph H of treewidth at most 4 and a path P such that $G \subseteq H \boxtimes P \boxtimes K_2$; and*
- (c) *a planar graph H of treewidth at most 6 and a path P such that $G \subseteq H \boxtimes P$ [18].*

¹ For graphs G and X , an X -decomposition of G is a collection $\mathcal{X} := (B_x : x \in V(X))$ of subsets of $V(G)$ called *bags* indexed by the vertices of X and such that (i) for each $v \in V(G)$, $X[\{x \in V(X) : v \in B_x\}]$ is connected; and (ii) for each $vw \in E(G)$, there exists some $x \in V(X)$ such that $\{v, w\} \subseteq B_x$. The *width* of \mathcal{X} is $\max\{|B_x| : x \in V(X)\} - 1$. In the special case where X is a tree, \mathcal{X} is called a *tree decomposition* of G . The *treewidth* $tw(G)$ of G is the minimum width of any tree decomposition of G .

² For two graphs G_1 and G_2 , the *strong graph product* of G_1 and G_2 , denoted $G_1 \boxtimes G_2$, is a graph whose vertex set is $V(G_1 \boxtimes G_2) := V(G_1) \times V(G_2)$ and that contains an edge between distinct vertices $v = (v_1, v_2)$ and $w = (w_1, w_2)$ if and only if (i) $v_1 = w_1$ and $v_2 w_2 \in E(G_2)$; (ii) $v_2 = w_2$ and $v_1 w_1 \in E(G_1)$; or (iii) $v_1 w_1 \in E(G_1)$ and $v_2 w_2 \in E(G_2)$.



© Prosenjit Bose, Pat Morin, and Saeed Odak;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 19; pp. 19:1–19:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In each of the applications of Theorem 1, the proofs are constructive and lead to algorithms whose running-time is dominated by the time required to compute the relevant decomposition. The proofs of each part of Theorem 1 are constructive and lead to $O(n^2)$ time algorithms as observed already by Dujmović et al. [8]. Morin [16] later showed that there exists an $O(n \log n)$ time algorithm to find the decomposition in Theorem 1.a. In the current note, we show that there exists a linear time algorithm for finding each of the three decompositions guaranteed by Theorem 1. This immediately gives an $O(n)$ -time algorithm for each of the following problems on any n -vertex planar graph G :

- computing an $O(1)$ -queue layout of G [8];
- nonrepetitively vertex-colouring G with $O(1)$ colours [11];
- assigning $(1 + o(1)) \log n$ -bit labels to the vertices of G so that one can determine from the labels of vertices v and w whether or not v and w are adjacent in G [10];
- mapping the vertices of G into a universal graph U_n that has $n^{1+o(1)}$ vertices and edges so that any pair of vertices that are adjacent in G maps to a pair of vertices that are adjacent in U_n [12];
- colouring the vertices of G with $O(p^3 \log p)$ colours so that each connected subgraph H of G contains a vertex whose colour is unique in H or contains vertices of at least $p + 1$ different colours [6]; and
- colouring the vertices of G with $O(\log n / \log \log \log n)$ integers so that the maximum colour that appears on any path P of length at most ℓ appears at exactly one vertex of P (for any fixed $\ell \geq 2$) [5].

In addition to planar graphs, product structure theorems similar to Theorem 1 exist for k -planar graphs, h -framed graphs, bounded-genus graphs, and graphs from apex-minor-free families [8, 9, 2]. The existence of each of these decompositions relies at some point on Theorem 1. Thus, the algorithm presented here can also serve as an optimal subroutine in the computation of product structure decompositions of graphs from these classes. However, these larger graph classes also require additional machinery that (at the time of writing) makes finding these decompositions largely impractical.³

The remainder of this paper is organized as follows: Section 2 presents some necessary background and notation. Section 3 reviews the proof of Theorem 1.a. Section 4 presents the linear time algorithm for finding the decomposition in Theorem 1.a. Section 5 describes the algorithms for finding the decompositions in Theorem 1.b and Theorem 1.c.

2 Preliminaries

Throughout this paper we use standard graph theory terminology as used in the textbook by Diestel [7]. All graphs discussed here are simple and finite. For a graph G , $V(G)$ and $E(G)$ denote the vertex and edge sets of G , respectively. We use the terms *vertex* and *node* interchangeably, though we typically refer to the vertices of some primary graph G of interest and refer to the nodes of some auxiliary graph (such as a spanning tree) related to G . We say that a subgraph G' of a graph G *spans* a set $S \subseteq V(G)$ if $S \subseteq V(G')$.

Quotient Graphs

Given a graph G and a partition \mathcal{P} of $V(G)$, the *quotient graph* G/\mathcal{P} is the graph with vertex set $V(G/\mathcal{P}) := \mathcal{P}$ and in which two nodes $X, Y \in V(G/\mathcal{P})$ are adjacent if G contains at least one edge xy with $x \in X$ and $y \in Y$.

³ Possible exceptions here are cases in which a bounded-genus graph, k -planar graph, or h -framed graph is given along with its embedding.

Embeddings, Planar Graphs, and (Near-)Triangulations

An *embedding* ψ of a graph G associates each vertex v of G with a point $\psi(v) \in \mathbb{R}^2$ and each edge vw of G with a simple open curve $\psi(vw) : (0, 1) \rightarrow \mathbb{R}^2$ whose endpoints⁴ are $\psi(v)$ and $\psi(w)$. We do not distinguish between such a curve $\psi(vw)$ and the point set $\{\psi(vw)(t) : 0 < t < 1\}$. We let $\psi(V(G)) := \{\psi(v) : v \in V(G)\}$, $\psi(E(G)) := \bigcup_{vw \in E(G)} \psi(vw)$, and $\psi(G) := \psi(V(G)) \cup \psi(E(G))$. An embedding ψ of G is *plane* if $\psi(vw) \cap \psi(V(G)) = \emptyset$ and $\psi(vw) \cap \psi(xy) = \emptyset$ for each distinct pair of edges $vw, xy \in E(G)$. A graph G is *planar* if it has a plane embedding. A *triangulation* is an edge-maximal planar graph.

If ψ is a plane embedding of a planar graph G , then we call the pair (G, ψ) an *embedded graph* and we will not distinguish between a vertex v of G and the point $\psi(v)$ or between an edge vw of G and the curve $\psi(vw)$. Similarly, we will not distinguish between G and the point set $\psi(G)$. Any cycle in an embedded graph defines a Jordan curve. For such a cycle C , $\mathbb{R}^2 \setminus C$ has two components, one bounded and the other unbounded. We will refer to the bounded component as the *interior* of C and the unbounded component as the *exterior* of C . If G is an embedded triangulation, then the subgraph of G consisting of all edges and vertices of G contained in the closure of the interior of C is called a *near-triangulation*.

Each component of $\mathbb{R}^2 \setminus G$ is a *face* of G and we let $F(G)$ denote the set of faces of G . If G is 2-connected then, for any face $f \in F(G)$, the set of vertices and edges of G contained in the boundary of f forms a cycle. We may therefore treat a face f of a 2-connected graph G as a component of $\mathbb{R}^2 \setminus G$ or as the cycle of G on the boundary of f , relying on context to distinguish between the two usages. Note that every embedded graph contains exactly one face – the *outer face* – that is unbounded.

Duals and Cotrees

The *dual* G^* of an embedded graph G is the graph with vertex set $V(G^*) := F(G)$ and edge set $E(G^*) := \{fg \in \binom{F(G)}{2} : E(f) \cap E(g) \neq \emptyset\}$.⁵ If T is a spanning tree of G then the *cotree* \bar{T} of (G, T) is the graph with vertex set $V(\bar{T}) := V(G^*)$ and edge set $E(\bar{T}) := \{ab \in E(G^*) : E(a) \cap E(b) \setminus E(T) \neq \emptyset\}$. It is well known that, if G is connected and T is a spanning tree of G then \bar{T} is a spanning tree of G^* [7, Chapter 4, Exercise 42].

For our purposes, a *binary tree* is a rooted tree of maximum degree 3 whose root has degree at most 2 and in which each child v of a node u is either the unique *left child* or the unique *right child* of u . If G is a triangulation and we root \bar{T} at any face $f_0 \in F(G)$ that contains an edge of T , then \bar{T} is a binary tree, with the classification of left and right children determined by the embedding of G .⁶

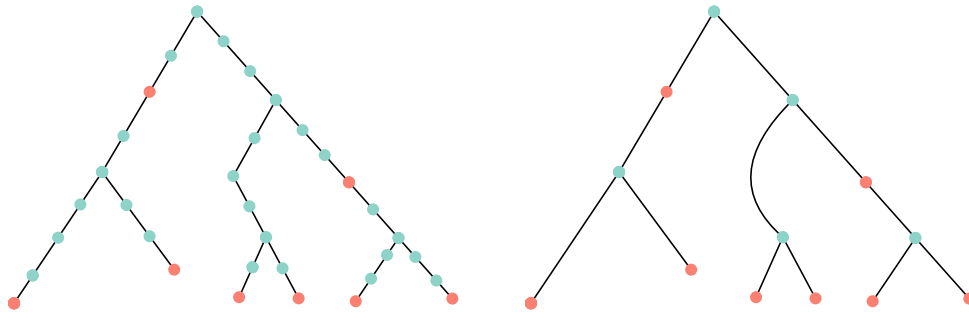
Paths and Distances

A *path* in G is a (possibly empty) sequence of distinct vertices v_0, \dots, v_r with the property that $v_{i-1}v_i \in E(G)$, for each $i \in \{1, \dots, r\}$. The *endpoints* of a path v_0, \dots, v_r are the vertices v_0 and v_r . The *length* of a non-empty path v_0, \dots, v_r is the number, r , of edges in the path.

⁴ The *endpoints* of an open curve $\psi : (0, 1) \rightarrow \mathbb{R}^2$ are the two points $\lim_{\epsilon \downarrow 0} \psi(\epsilon)$ and $\lim_{\epsilon \downarrow 0} \psi(1 - \epsilon)$.

⁵ For a set S , $\binom{S}{2}$ denotes the $\binom{|S|}{2}$ -element set $\binom{S}{2} := \{\{x, y\} : x, y \in S, x \neq y\}$.

⁶ There is a small ambiguity here when T contains two edges of f_0 , in which case the unique child of f_0 in \bar{T} can be treated as the left or right child of f_0 .



■ **Figure 1** A binary tree T with set $S \subseteq V(T)$ depicted in red and the model of T with respect to S .

Trees, Depth, Ancestors, and Descendants

Let T be a tree rooted at a vertex $v_0 \in V(T)$. For any vertex $w \in V(T)$, $P_T(w)$ denotes the path in T from w to v_0 . For any $w_0 \in V(T)$, any prefix w_0, \dots, w_r of $P_T(w_0)$ is called an *upward path* in T ; w_0 is the *lower endpoint* of this path and w_r is the *upper endpoint*. The *T -depth* of a node $w \in V(T)$ is the length of the path $P_T(w)$. The second node in $P_T(v)$ (if any) is the *T -parent* of v . A vertex $a \in V(T)$ is a *T -ancestor* of $w \in V(T)$ if $a \in V(P_T(w))$. If a is a *T -ancestor* of w then w is a *T -descendant* of a .

Lowest Common Ancestors

For any two vertices $v, w \in V(T)$, the *lowest common ancestor* $\text{lca}_T(v, w)$ of v and w is the node a in $P_T(v) \cap P_T(w)$ having maximum T -depth. The *lowest common ancestor problem* is a well-studied data structuring problem that asks to preprocess a given n -vertex rooted tree so that one can quickly return $\text{lca}_T(v, w)$ for any two nodes $v, w \in V(T)$. A number of optimal solutions to this problem exist that, after $O(n)$ time preprocessing using $O(n)$ space, can answer queries in $O(1)$ time [4, 17, 15, 1, 3, 13]. The most recent work in this area includes simple and practical data structures that achieve this optimal performance [1, 3, 13].

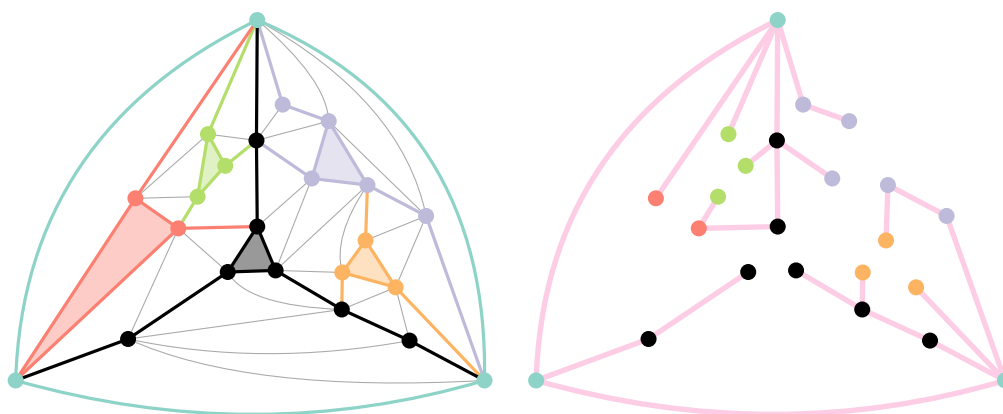
Reconstructing Binary Tree Models

Let T be a binary tree and $S \subseteq V(T)$. An upward path v_0, \dots, v_r in a binary tree T is *S -non-branching* if v_i has degree 2 and $v_i \notin S$ for each $i \in \{1, \dots, r-1\}$. For any binary tree T and set $S \subseteq V(T)$, the *model* T' of T with respect to S is the binary tree obtained by replacing each maximal S -non-branching path v_0, \dots, v_r with the edge v_0v_r ; if v_{r-1} is the left (respectively, right) child of v_r then v_0 becomes the left (respectively, right) child of v_r . See Figure 1.

► **Lemma 2.** *Let T be a binary tree, let $S = \{x_1, \dots, x_d\} \subseteq V(T)$, and let T_0 be the minimal subtree of T that spans S . Then there exists an algorithm that, given an $O(1)$ -query time lowest common ancestor data structure for T , computes the model T'_0 of T_0 with respect to S in $O(d^2)$ time.*

Proof. The proof is by induction on $|S|$. The base case $|S| = 1$ is trivial, since then $T'_0 = T_0$ is the tree with one node, which is the unique element in S .

If $|S| \geq 2$, then the first step is to determine the root r of T_0 , which must also be the root of T'_0 . This is easily done by first setting $r := x_1$ and then repeatedly setting $r := \text{lca}_T(r, x_i)$ for each $i \in \{2, \dots, d\}$. This step takes $O(d)$ time.



■ **Figure 2** A (G, T) -tripod decomposition of a triangulation G (and the underlying spanning tree T).

If r has no left child in T , then we can immediately apply induction on $S \setminus \{r\}$ and make the right child of r in T'_0 the root of the model obtained by induction. The case in which r has no right child can be handled similarly. If r has both a left child r_1 and a right child r_2 , then the next step is to partition $S \setminus \{r\}$ into a set S_1 of descendants of r_1 and a set S_2 of descendants of r_2 . For each $x \in S \setminus \{r\}$ there are only two possibilities for $\text{lca}_T(r_1, x)$

1. If $\text{lca}_T(r_1, x) = r_1$ then $x \in S_1$.
2. If $\text{lca}_T(r_1, x) = r$ then $x \in S_2$.

Therefore, using $O(d)$ lowest common ancestor queries, we can determine the root r of T' and partition $S \setminus \{r\}$ into sets S_1 and S_2 that define the left and right subtrees of r . We can now recurse on S_1 to obtain a tree with root r'_1 and recurse on S_2 to obtain a tree with root r'_2 . We make r'_1 the left child of r and r'_2 the right child of r to obtain the model T'_0 of T_0 . The running-time of this algorithm obeys the recurrence $T(d) \leq O(d) + T(d_1) + T(d_2)$, where $d_1 + d_2 \leq d$ and $d_1, d_2 \leq d - 1$. This recurrence resolves to $T(d) \in O(d^2)$. ◀

3 Tripod Decompositions

Refer to Figure 2. Let G be an n -vertex triangulation and let T be a spanning tree of G . For a face uvw of G , a (G, T) -tripod Y with crotch uvw is the vertex set of three disjoint (and each possibly empty) upward paths in T (the legs of Y) whose lower endpoints are u , v , and w . A (G, T) -tripod decomposition is a partition of $V(G)$ into (G, T) -tripods. Dujmović et al. [8] proved the following result:

► **Theorem 3.** *Let G be a triangulation and T be a spanning tree of G . Then there exists a (G, T) -tripod decomposition \mathcal{Y} such that G/\mathcal{Y} has treewidth at most 3.*

It is straightforward to verify that Theorem 3 implies Theorem 1.a by first triangulating the given graph and then taking T to be a breadth-first spanning tree of the resulting triangulated graph [8, Observation 35].

3.1 Tripod Decompositions from Face Orderings

We now describe how a (G, T) -tripod decompositions can be obtained from a sequence of distinct faces of G . Throughout this section (and for the remainder of the paper):

- G is an embedded triangulation with outer face f_0 and
- T is a spanning-tree of G rooted at a vertex $v_0 \in V(f_0)$.

19:6 An Optimal Algorithm for Product Structure

For any subgraph f of G , we define $Y_T(f) := f \cup \bigcup_{v \in V(f)} P_T(v)$.⁷ In words, $Y_T(f)$ is the subgraph of G that includes all the vertices and edges of f and all the vertices and edges of each path from each vertex of f to the root of T .

Let $\mathcal{F} := f_0, \dots, f_r$ be a sequence of distinct faces of G whose first element is the outer face f_0 . Let G_{-1} denote the graph with no vertices and, for each $i \in \{0, \dots, r\}$, define the graph $G_i := \bigcup_{j=0}^i Y_T(f_j)$ and let $Y_i := V(G_i) \setminus V(G_{i-1})$. Let $\mathcal{G}_{\mathcal{F}} := G_0, \dots, G_r$ and let $\mathcal{Y}_{\mathcal{F}} := Y_0, \dots, Y_r$.

Informally, we require that each of the *legs* of each tripod Y_i have a *foot* on a different vertex of G_{i-1} and that the tripods Y_1, \dots, Y_r cover all the vertices and edges of G . Formally, we say that the sequence \mathcal{F} is *proper* if, for each $i \in \{1, \dots, r\}$, and each distinct $v, w \in V(f_i)$, $V(Y_T(v) \cap G_{i-1}) \neq V(Y_T(w) \cap G_{i-1})$. The sequence \mathcal{F} is *complete* for G if $G_r = G$. Note that, if \mathcal{F} is complete, then $\{Y_0, \dots, Y_r\}$ is a tripod decomposition of G .

From the preceding definitions it follows that, if \mathcal{F} is proper, then G_i is 2-connected for each $i \in \{0, \dots, r\}$. For any $i \in \{0, \dots, r\}$, consider any face f of G_i , that we now treat as a cycle in G . An easy proof by induction shows that, for any $j \in \{0, \dots, i\}$, the induced graph $f[Y_j]$ is connected. We are interested in keeping the number of tripods in Y_0, \dots, Y_i that contribute to $V(f)$ as small as possible, which motivates our next definition.

The sequence \mathcal{F} is *good* if the resulting sequence of graphs $\mathcal{G}_{\mathcal{F}} := G_0, \dots, G_r$ and tripods $\mathcal{Y}_{\mathcal{F}} := Y_0, \dots, Y_r$ satisfy the following condition: For each $i \in \{0, \dots, r\}$ and each face f of G_i ,

$$|\{\ell \in \{0, \dots, i\} : V(f) \cap Y_\ell \neq \emptyset\}| \leq 3 .$$

In words, each face of each graph G_i has vertices from at most three tripods of Y_0, \dots, Y_i on its boundary. Even more, the vertices of f can be partitioned into at most three paths where the vertices of each path belong to a single tripod. Dujmović et al. [8] prove Theorem 3 by proving the next lemma.

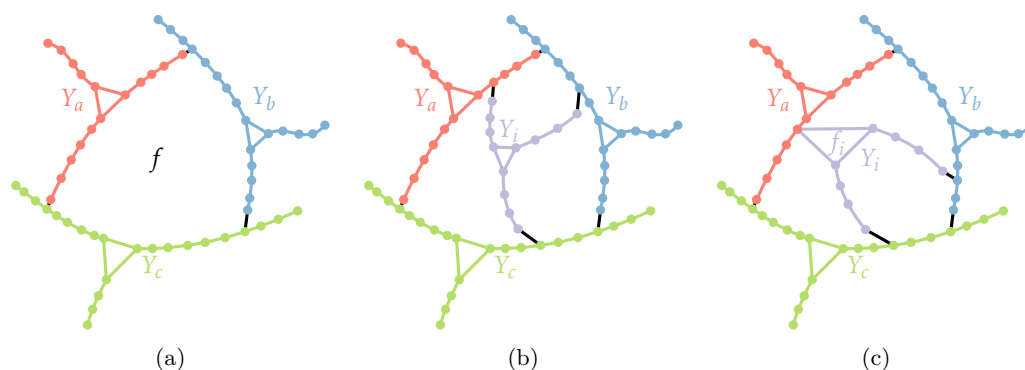
► **Lemma 4.** *Let G be a triangulation with a vertex v_0 on its outer face f_0 and let T be a spanning tree of G rooted at v_0 . Then there exists a sequence $\mathcal{F} := f_0, \dots, f_r$ of distinct faces of G that is proper, good, and complete.*

► **Remark 5.** Lemma 4 is stated in terms of sequences only for convenience and could be rephrased in terms of partial orders. Indeed, consider the partial order \prec defined as follows: For each $i \in \{1, \dots, r\}$ let f'_i be the face of G_{i-1} that contains f_i ; then $f_\ell \prec f_i$ for each $\ell \in \{0, \dots, i-1\}$ such that $V(f'_i) \cap Y_\ell \neq \emptyset$. It is straightforward to check that any linearization of this partial order will result in the same tripod decomposition $\mathcal{Y}_{\mathcal{F}} := \{Y_0, \dots, Y_r\}$.

Dujmović et al. [8] prove Lemma 4 by giving a recursive algorithm that constructs the face sequence \mathcal{F} . For a face f of G_i , define the set $I_f := \{\ell \in \{0, \dots, i\} : V(f) \cap Y_\ell \neq \emptyset\}$. They begin with the outer face f_0 of G . To find the face f_i , $i > 0$, they consider some face $f \notin \{f_0, \dots, f_{i-1}\}$ of G_{i-1} and use Sperner's Lemma to show that there is an appropriate face f_i of G (called a *Sperner triangle*) that is contained in f . In particular, f_i is chosen so that the three upward paths in $Y_{\mathcal{F}}(f_i)$ lead back to each of the (at most 3) tripods in $\{Y_j : j \in I_f\}$. See Figure 3.

This proof leads to a divide-and-conquer algorithm: After finding f_i , the algorithm recursively decomposes each of the near-triangulations that are bounded by the at most three new faces in $S_i := F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$. The Sperner triangle f_i can easily be found

⁷ In all of our examples, the subgraph f will always be a single edge or single face of G .



■ **Figure 3** Each face f in G_{i-1} is bounded by at most three tripods Y_{a_f} , Y_{b_f} , and Y_{c_f} and the tripod Y_i is chosen so that it connects each of these.

in time proportional to the number of faces of G in the interior of f . However, because the resulting recursion is not necessarily balanced, a straightforward implementation of this yields an algorithm with $\Theta(n^2)$ worst-case running time.

Morin [16] later showed that, using an appropriate data structure for T , this approach can be implemented in such a way that the resulting algorithm runs in $O(n \log n)$ time. Essentially, Morin’s algorithm works by finding the Sperner triangle f_i in time proportional to the minimum number of faces of G contained in any of the faces in S_i . In the next section, we will show that, by using a lowest common ancestor data structure for the cotree \bar{T} along with Lemma 2, the Sperner triangle f_i can be found in constant time, yielding an $O(n)$ time algorithm.

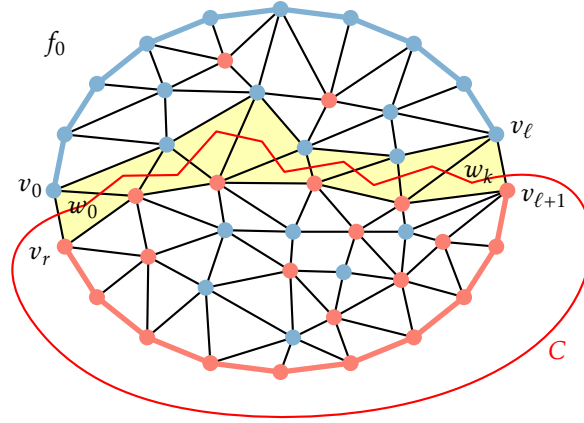
By now, our presentation of this material differs somewhat from that in [8, 18]. Therefore, we now pause to explain how Lemma 4 implies Theorem 3. Let G be a triangulation, let T be spanning tree of G , let $\mathcal{F} := f_0, \dots, f_r$ be the proper good face sequence guaranteed by Lemma 4, and let $\mathcal{Y}_{\mathcal{F}} := \{Y_0, \dots, Y_r\}$ be the resulting tripod decomposition. We now show that there exists a chordal graph H whose largest clique has size at most 4 and that contains $G/\mathcal{Y}_{\mathcal{F}}$. We construct the graph H so that for each $i \in \{0, \dots, r\}$ and each face f of G_i , H contains a clique on $\{Y_j : j \in I_f\}$. To accomplish this, for each $i \in \{1, \dots, r\}$ let f be the face of G_{i-1} that contains f_i and form a clique on $\{Y_i\} \cup \{Y_j : j \in I_f\}$. Inductively, the elements of $\{Y_j : j \in I_f\}$ already form a clique, so this operation is equivalent to attaching Y_i to all the vertices of an existing clique of size at most 3. Therefore, this results in a chordal graph H whose largest clique has size at most 4 and therefore H has treewidth at most 3 [14].

4 An $O(n)$ -Time Algorithm

Refer to Figure 4 for an illustration of the following (probably well-known) simple lemma, which is closely related to Sperner’s Lemma:

► **Lemma 6.** *Let N be a near-triangulation with outer face v_0, \dots, v_r and colour each vertex of N red or blue in such a way that v_0, \dots, v_ℓ are coloured red for some $\ell \in \{0, \dots, r-1\}$ and $v_{\ell+1}, \dots, v_r$ are coloured blue. Then there exists a path w_0, \dots, w_k in N^* such that*

1. w_0 is the inner face of N with v_0v_r on its boundary;
2. w_k is the inner face of N with $v_\ell v_{\ell+1}$ on its boundary; and
3. for each $i \in \{1, \dots, k\}$, the single edge in $E(w_{i-1}) \cap E(w_i)$ has an endpoint of each colour.



■ **Figure 4** Lemma 6.

Proof. If $w_0 = w_k$, the lemma is immediately true, so assume $w_0 \neq w_k$. Say that an edge of N is *bichromatic* if one of its endpoints is red and the other is blue. Any edge that is not bichromatic is *monochromatic*. The outer face f_0 of N has exactly two bichromatic edges v_0v_r and v_lv_{l+1} and any inner face of N has either zero or two bichromatic edges. Consider the subgraph H of N^* obtained removing each edge $fg \in E(N^*)$ such that the edge in $E(f) \cap E(g)$ is monochromatic. Every vertex in H has degree 0 or 2, so each connected component of H is either an isolated vertex or a cycle. The face f_0 has degree 2 so it is contained in a cycle C of H . The two neighbours of f_0 in H are w_0 and w_k . Therefore C contains a path w_0, \dots, w_k that satisfies the conditions of the lemma. ◀

The next lemma, which is the main new insight in this paper, allows us to use Lemma 2 to find Sperner triangles in constant time.

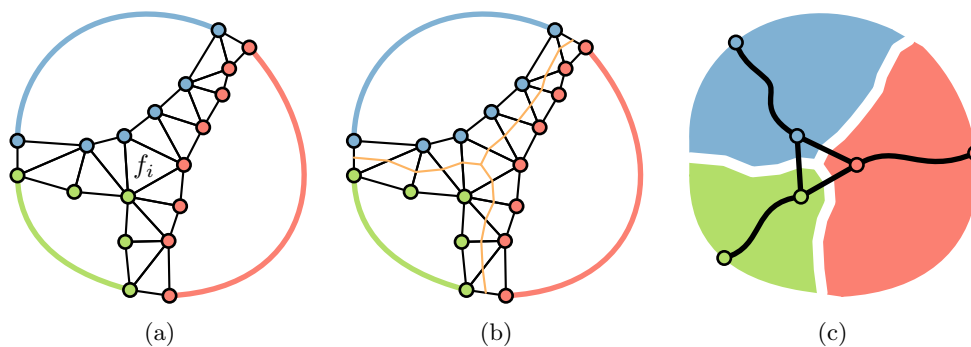
► **Lemma 7.** *Let G be a triangulation with a vertex v_0 on its outer face f_0 ; let T be a spanning tree of G rooted at v_0 ; let \bar{T} be the cotree of (G, T) rooted at f_0 ; let f_0, \dots, f_{i-1} be a good proper sequence of faces of G that yields a sequence $\mathcal{G}_{\mathcal{F}} := G_0, \dots, G_{i-1}$ of graphs and a sequence $\mathcal{Y}_{\mathcal{F}} := Y_0, \dots, Y_{i-1}$ of tripods; let $f \notin \{f_0, \dots, f_{i-1}\}$ be a face of G_{i-1} , and let $S \subseteq F(G)$ contain exactly the (at most three) faces $g \in F(G)$ such that*

- (i) g is contained in the interior of f ;
- (ii) g contains an edge $vw \in E(f)$ with $v \in Y_a$ and $w \in Y_b$ for some distinct $a, b \in I_f$.

Let \bar{T}_0 be the minimal subtree of \bar{T} that spans S . Then, if S is non-empty and $f_i \in V(\bar{T}_0)$ is such that each component of $\bar{T}_0 - f_i$ contains at most one element of S , Then f_0, \dots, f_i is good.

Proof. Let N be the near-triangulation consisting of all vertices and edges of G contained in the closure of the interior of f . Recall that $I_f := \{j \in \{0, \dots, i-1\} : Y_j \cap V(f) \neq \emptyset\}$. Since f_0, \dots, f_{i-1} is good, $|I_f| \leq 3$. Since S is non-empty $|I_f| \geq 2$. For each $j \in I_f$, colour each vertex v of N with the colour j if the first vertex of $P_T(v)$ in $V(f)$ is contained in Y_j . Say that an edge or face of N is *monochromatic*, *bichromatic*, or *trichromatic* if it contains vertices of one, two, or three colours, respectively.

$E(f)$ contains exactly $|I_f|$ bichromatic edges. Since each element of S is an inner face of N that contains a bichromatic edge of f , $|S| \leq |I_f| \leq 3$. Let X be the subgraph of N^* that contains an edge $fg \in E(N^*)$ if and only if f and g are inner faces of N and the edge



■ **Figure 5** The proof of Lemma 7.

in $E(f) \cap E(g)$ is bichromatic. We claim that X is a subgraph of \bar{T} . In order to show this, we need only argue that each edge uv of T in the interior of f is monochromatic. Consider any $uv \in E(N) \setminus E(f)$ where u is the T -parent of v . If $v \notin V(f)$ then, by definition, v has the same colour as u , so uv is monochromatic. The case where $v \in V(f)$ and $u \notin V(f)$ can not occur since $v \in V(f)$ implies that $P_T(v) \subseteq G_{i-1}$, but $u \notin V(G_{i-1})$. Similarly, the case in which $u \in V(f)$ and $v \in V(f)$ can not occur since this implies that $P_T(v) \subseteq G_{i-1}$, but $uv \notin E(G_{i-1})$.

Next we claim that all the elements of S are in a single connected component of X . If $|I_f| = 2$, then this follows immediately from Lemma 6. If $|I_f| = 3$, then let $\{a, b, c\} := I_f$ and consider a pair $g_1, g_2 \in S$ where (without loss of generality) g_1 contains a bichromatic edge of f with colours a and b and g_2 contains a bichromatic edge of f with colours b and c . By treating a and c as a single colour we may again apply Lemma 6 to conclude that g_1 and g_2 are in the same component of X .

Refer to Figure 5(a). Therefore X is a subgraph of \bar{T} that has a component containing all the elements of S . Therefore X contains \bar{T}_0 . By choice, \bar{T}_0 contains a path from f_i to each $g \in S$ and each of these paths is disjoint except for their shared starting location f_i .

Refer to Figure 5(b). Now, consider the embedded graph X_0 obtained as follows: For each $g \in V(\bar{T}_0)$, place a vertex on the center of each bichromatic edge of g and, if g is trichromatic, then place a vertex in the center of g . Next,

1. add an edge joining the center of each trichromatic triangle to each of the centers of its bichromatic edges; and
2. add an edge (embedded as a straight line segment) joining the centers of each pair of bichromatic edges that are on a common bichromatic face $g \in V(\bar{T}_0)$.

The graph X_0 is a tree of maximum-degree 3 that has $|I_f|$ leaves. (Each leaf in X_0 is the center of a bichromatic edge in $E(f)$). With the exception of these three leaves, every point in the embedding of X_0 is contained in the interior of f .

Refer to Figure 5(c). Now treat X_0 as a point set and consider the point set f' obtained by removing X_0 from the closure of f . Now f' has $|I_f|$ connected components and each vertex of f_i is in a different component. Each of the components of f' contains vertices of Y_j for exactly one $j \in I_f$; call this the *colour* of the component. Since no edge of T crosses X_0 , the colour of each vertex in f_i is equal to the colour the component of f' that contains it.

Finally, to see that f_0, \dots, f_i is good first observe that we need only be concerned with the at most three faces in $F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$ and each of these shares a bichromatic edge with f_i . If g is a face in $F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$ with $E(g) \cap E(f_i) = \{uv\}$ and uv is coloured with a and b , then $V(g) \cap Y_j = \emptyset$ for any $j \in \{0, \dots, i\} \setminus \{a, b, i\}$. This completes the proof. ◀

► **Theorem 8.** *There exists an $O(n)$ time algorithm that, given any n -vertex triangulation G and any rooted spanning tree T of G , produces a (G, T) -tripod decomposition \mathcal{Y} such that $\text{tw}(G/\mathcal{Y}) \leq 3$.*

Proof. Let v_0 be the root of T and let f_0 be a face of G incident to v_0 that contains an edge of T incident to v_0 . In a preprocessing step, we compute the cotree \bar{T} of (G, T) and construct a lowest common ancestor data structure for \bar{T} in $O(n)$ time that allows us to compute $\text{lca}_{\bar{T}}(f, g)$ for any two faces $f, g \in F(G)$ in $O(1)$ time.

After this preprocessing, we construct the good sequence f_0, \dots, f_r recursively. Conceptually, during any recursive invocation, the input is a near-triangulation N bounded by a cycle C in G whose vertices belong to at most three tripods computed in previous steps. Each vertex of G starts initially *unmarked* and we *mark* a vertex once we have placed it in a tripod. The precise input to a recursive invocation is defined as follows:

1. If C intersects three tripods then the input consists of the three inner faces g_1, g_2 , and g_3 of N that contain bichromatic edges of C . Lemma 7 characterizes the face f_i in terms of the minimum subtree \bar{T}_0 of \bar{T} that contains g_1, g_2 , and g_3 . Indeed, f_i is either the unique degree-3 node of \bar{T}_0 (if g_1, g_2 , and g_3 are all leaves of \bar{T}_0) or f_i is the unique node among g_1, g_2 , or g_3 that has degree 2. By Lemma 2 we can construct the model \bar{T}'_0 of \bar{T}_0 in constant time and find the node f_i .
2. If C intersects two tripods, then the input consists of two inner faces g_1, g_2 , of N with bichromatic edges of C on their boundary. In this case, we let $f_i := g_1$ or $f_i = g_2$, either choice satisfies our requirements.
3. If C intersects only one tripod, then the input consists of any inner face g_1 of N that contains an edge in $E(f)$. In this case $f_i := g_1$ satisfies our requirements.

Once we have found the Sperner triangle f_i , we can compute the tripod Y_i and mark its vertices by following the path in T from each vertex of f_i to its nearest marked ancestor in T . This takes $O(1 + |Y_i|)$ time. Once we have done this, we have also found the at most three bichromatic edges of G_i that are needed to perform the at most three recursive invocations on the near triangulations whose outer faces coincide with each of the new faces in $F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$.

After setting f_0 , the initial recursive call falls into the third case above, so its input is any of the three inner faces that shares an edge with the outer face, f_0 . Each recursive invocation adds a new face f_i to the good face sequence f_0, \dots, f_r and takes $O(1 + |Y_i|)$ time. Since Y_0, \dots, Y_r is a partition of $V(G)$, the running time of this algorithm is therefore $\sum_{i=0}^r O(1 + |Y_i|) = O(n)$. ◀

5 Variations

In this section we show that there are $O(n)$ time algorithms for computing the decompositions in Theorem 1.b and Theorem 1.c. In the same way that Theorem 1.a follows from the tripod decomposition of Theorem 3, Theorem 1.b follows from a bipod decomposition given by Theorem 10 and Theorem 1.c follows from a monopod decomposition given by Theorem 11.

5.1 Bipod Decompositions

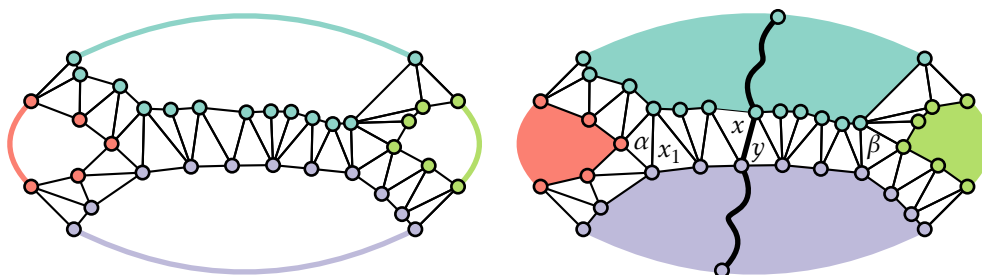
We begin with the decomposition in Theorem 1.b, which was communicated to us by Vida Dujmović, and has not appeared before. This decomposition is obtained by selecting a proper sequence $\mathcal{E} := e_0, \dots, e_k$ of distinct edges of G , which define a sequence of graphs $\mathcal{G}_{\mathcal{E}} := G_0, \dots, G_k$ where $G_i := \bigcup_{j=0}^i P_T(e_j)$ and a sequence of *bipods* $\mathcal{I}_{\mathcal{E}} := \Lambda_0, \dots, \Lambda_k$ where $\Lambda_i = V(G_i) \setminus V(G_{i-1})$. We call \mathcal{E} *good* if, for each $i \in \{0, \dots, k\}$ and each face $f \in F(G_i)$, $V(f)$ has a non-empty intersection with at most 4 bipods in $\Lambda_0, \dots, \Lambda_i$.

Exactly the same argument used in Section 3.1 to show that $G/\mathcal{Y}_{\mathcal{F}}$ is contained in a chordal graph of maximum clique size 4 also shows that if \mathcal{E} is a good edge sequence that produces a bipod partition $\mathcal{I}_{\mathcal{E}}$ of $V(G)$, then $G/\mathcal{I}_{\mathcal{E}}$ is contained in a chordal graph of maximum clique size 5, so $G/\mathcal{I}_{\mathcal{E}}$ has treewidth at most 4.

We now explain why a good edge sequence e_0, \dots, e_r exists.⁸ As before, we set f_0 to be any face of G such that $E(f_0)$ contains an edge of T incident to the root v_0 of T . The edge e_0 is any edge of $E(f_0) \setminus E(T)$. Next we take special care to ensure that G_i is biconnected for $i \geq 1$. In particular, if G_0 contains only two edges of f_0 , then we take e_1 to be the edge of f_0 that does not appear in G_0 . Otherwise, we choose e_1 using the general strategy for choosing e_i , described next.

Refer to Figure 6. Now we may assume that G_{i-1} is biconnected. To choose the edge e_i , we consider any face $f \in F(G_{i-1}) \setminus F(G)$. Inductively, $V(f)$ contains vertices from at most four bipods in $\Lambda_0, \dots, \Lambda_{i-1}$. Let $I_f := \{j \in \{0, \dots, i-1\} : \Lambda_j \cap V(f) \neq \emptyset\}$. If $|I_f| < 4$ then we can select e_i to be any edge in the interior of f . Therefore, we focus on the case $|I_f| = 4$. As before we colour vertices in the near triangulation N using colours in the set I_f ; we let S be the set of inner faces in N that contain a bichromatic edge in $E(f)$; and let \bar{T}_0 be the minimal subtree of \bar{T} that spans S . The same argument in the proof of Lemma 7 shows that every node of \bar{T}_0 is contained in f .

Claim 9, below, shows that \bar{T}_0 contains an edge xy such that each component of $\bar{T}_0 - xy$ contains at most two elements of S . It is straightforward to verify that, if we choose e_i to be the edge in $E(x) \cap E(y)$ then we obtain a graph G_i in which each of the two new faces containing vertices from Λ_i contains vertices from at most three bipods in $\{\Lambda_j : j \in I_f\}$, as required.



■ **Figure 6** Choosing the next e_i in a good edge sequence.

▷ **Claim 9.** \bar{T}_0 contains an edge xy such that each component of $\bar{T}_0 - xy$ contains at most two nodes of S .

Proof. Direct each edge xy of \bar{T}_0 in the direction \overrightarrow{xy} if the component of $\bar{T}_0 - xy$ that contains y contains three or more nodes of S . It is sufficient to show that this process leaves some edge xy of \bar{T}_0 undirected. Assume for the sake of contradiction that every edge of \bar{T}_0 is directed. Then some node x of \bar{T}_0 has only incoming edges. Certainly x does not have degree 1 in \bar{T}_0 .

If x has degree 2 in \bar{T}_0 then \bar{T}_0 contains two subtrees T_1 and T_2 that have only the node x in common and such that $|V(T_1) \cap S| \geq 3$ and $|V(T_2) \cap S| \geq 3$, which implies that $|S| \geq 3 + 3 - 1 > 4$, a contradiction.

⁸ The existence of this edge sequence is more easily proven using Sperner's Lemma, but we want a proof that lends itself to a linear time algorithm.

Suppose therefore that x has degree 3 in \bar{T}_0 . Each face in S contains an edge in $E(f)$, so each face in S has degree at most 2 in \bar{T}_0 . Therefore $x \notin S$. Therefore $\bar{T}_0 - x$ contains three components T_1, T_2, T_3 such that each pair of components contains at least 3 elements of S . But this implies that $|S| \geq (3 \times 3)/2 > 4$, a contradiction. \triangleleft

Algorithmically, using Lemma 2, we can construct the model \bar{T}'_0 of \bar{T}_0 in constant time given the set S . The model \bar{T}'_0 will also contain an edge $\alpha\beta$ such that each component of $\bar{T}'_0 - \alpha\beta$ contains at most two nodes in S . We claim that $E(\alpha)$ contains an edge that makes a suitable choice for e_i , and this edge can be found in constant time. Indeed, the edge $\alpha\beta$ in \bar{T}'_0 corresponds to a path $\alpha, x_1, \dots, x_k, \beta$ in \bar{T}_0 and the unique edge in $E(\alpha) \cap E(x_1)$ is a suitable choice for e_i .

The rest of the details of the algorithm are similar to those given in the proof of Theorem 8: Each subproblem is a near-triangulation N bounded by a cycle C and the input that defines the subproblem consists of the (at most four) faces $S \subseteq F(N)$ incident to bichromatic edges of C .⁹

► **Theorem 10.** *There exists an $O(n)$ time algorithm that, given any n -vertex triangulation G and any rooted spanning tree T of G , produces a (G, T) -bipod decomposition \mathcal{I} such that $\text{tw}(G/\mathcal{I}) \leq 4$.*

5.2 Monopod Decompositions

Finally we consider the decomposition described in Theorem 1.c. This decomposition is obtained from a tripod decomposition $\mathcal{Y} := Y_0, \dots, Y_r$, obtained by a sequence $\mathcal{F} := f_0, \dots, f_r$ of faces of G in the same manner described in Section 3.1. However in this setting, the sequence f_0, \dots, f_r is *good* if, for each $i \in \{0, \dots, r\}$ and each face f of $G_i := \bigcup_{j=0}^i Y_T(f_j)$, $V(f)$ contains vertices from at most 5 legs of tripods in Y_0, \dots, Y_i . Under these conditions, Ueckerdt et al. [18] are able to show that the *monopod decomposition* \mathcal{I} obtained by splitting each tripod Y_i into three upward paths yields a quotient graph G/\mathcal{I} of treewidth at most 6.

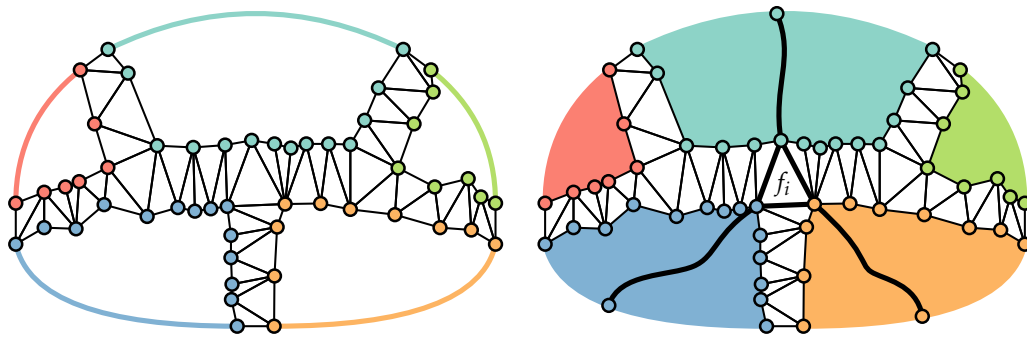
As before we focus on the extreme case when $V(f)$ contains vertices from exactly 5 legs of tripods. Refer to Figure 7. Following the same strategy used for the previous two decompositions, the set S in this case has size at most 5 and the face f_i corresponds to a node of \bar{T}_0 such that each component of $\bar{T}_0 - f_i$ contains at most 2 nodes in S . (This is always possible because $\lfloor 5/2 \rfloor = 2$.) Again, a suitable choice for f_i can be found in the model \bar{T}'_0 of \mathcal{T}_0 in constant time.

► **Theorem 11.** *There exists an $O(n)$ time algorithm that, given any n -vertex triangulation G and any rooted spanning tree T of G , produces a (G, T) -monopod decomposition \mathcal{I} such that $\text{tw}(G/\mathcal{I}) \leq 6$.*

References

- 1 Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory Comput. Syst.*, 37(3):441–456, 2004. doi:10.1007/s00224-004-1155-5.
- 2 Michael A. Bekos, Giordano Da Lozzo, Petr Hliněný, and Michael Kaufmann. Graph product structure for h -framed graphs. *CoRR*, abs/2204.11495, 2019. arXiv:2204.11495.

⁹ In the degenerate case where C has no bichromatic edges, the input is any face of N incident to an edge of C .



■ **Figure 7** The selection of a tripod by Ueckerdt et al. [18].

- 3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
- 4 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993. doi:10.1137/0222017.
- 5 Prosenjit Bose, Vida Dujmović, Mehrnoosh Javarsineh, and Pat Morin. Asymptotically optimal vertex ranking of planar graphs. *CoRR*, abs/2007.06455, 2020. arXiv:2007.06455.
- 6 Michal Debski, Stefan Felsner, Piotr Micek, and Felix Schröder. Improved bounds for centered colorings. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2212–2226. SIAM, 2020. doi:10.1137/1.9781611975994.136.
- 7 Reinhard Diestel. *Graph Theory, Fifth Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.
- 8 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. URL: <https://dl.acm.org/doi/10.1145/3385731>.
- 9 Vida Dujmovic, Pat Morin, and David R. Wood. Graph product structure for non-minor-closed classes. *CoRR*, abs/1907.05168, 2019. arXiv:1907.05168.
- 10 Vida Dujmović, Louis Esperet, Cyril Gavoille, Gwenaël Joret, Piotr Micek, and Pat Morin. Adjacency labelling for planar graphs (and beyond). *J. ACM*, 68(6):42:1–42:33, 2021. doi:10.1145/3477542.
- 11 Vida Dujmović, Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David R. Wood. Planar graphs have bounded nonrepetitive chromatic number. *CoRR*, abs/1904.05269, 2019. arXiv:1904.05269.
- 12 Louis Esperet, Gwenaël Joret, and Pat Morin. Sparse universal graphs for planarity. *CoRR*, abs/2010.05779, 2020. arXiv:2010.05779.
- 13 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.
- 14 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16:47–56, 1974. doi:doi:10.1016/0095-8956(74)90094-X.
- 15 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

19:14 An Optimal Algorithm for Product Structure

- 16 Pat Morin. A fast algorithm for the product structure of planar graphs. *Algorithmica*, 83(5):1544–1558, 2021. doi:10.1007/s00453-020-00793-5.
- 17 Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988. doi:10.1137/0217079.
- 18 Torsten Ueckerdt, David R. Wood, and Wendy Yi. An improved planar graph product structure theorem. *CoRR*, abs/2108.00198, 2021. arXiv:2108.00198.


Online Unit Profit Knapsack with Untrusted Predictions

Joan Boyar   

University of Southern Denmark, Odense, Denmark

Lene M. Favrholdt   

University of Southern Denmark, Odense, Denmark

Kim S. Larsen   

University of Southern Denmark, Odense, Denmark

Abstract

A variant of the online knapsack problem is considered in the settings of trusted and untrusted predictions. In Unit Profit Knapsack, the items have unit profit, and it is easy to find an optimal solution offline: Pack as many of the smallest items as possible into the knapsack. For Online Unit Profit Knapsack, the competitive ratio is unbounded. In contrast, previous work on online algorithms with untrusted predictions generally studied problems where an online algorithm with a constant competitive ratio is known. The prediction, possibly obtained from a machine learning source, that our algorithm uses is the average size of those smallest items that fit in the knapsack. For the prediction error in this hard online problem, we use the ratio $r = \frac{a}{\hat{a}}$ where a is the actual value for this average size and \hat{a} is the prediction. The algorithm presented achieves a competitive ratio of $\frac{1}{2r}$ for $r \geq 1$ and $\frac{r}{2}$ for $r \leq 1$. Using an adversary technique, we show that this is optimal in some sense, giving a trade-off in the competitive ratio attainable for different values of r . Note that the result for accurate advice, $r = 1$, is only $\frac{1}{2}$, but we show that no deterministic algorithm knowing the value a can achieve a competitive ratio better than $\frac{e-1}{e} \approx 0.6321$ and present an algorithm with a matching upper bound. We also show that this latter algorithm attains a competitive ratio of $r \frac{e-1}{e}$ for $r \leq 1$ and $\frac{e-r}{e}$ for $1 \leq r < e$, and no deterministic algorithm can be better for both $r < 1$ and $1 < r < e$.

2012 ACM Subject Classification Theory of computation \rightarrow Online algorithms

Keywords and phrases online algorithms, untrusted predictions, knapsack problem, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.20

Related Version *Full Version:* <https://arxiv.org/abs/2203.00285>

Funding Supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B.

1 Introduction

In this paper, we consider the Online Unit Profit Knapsack Problem: The request sequence consists of n item with sizes in $(0, 1]$. An online algorithm receives them one at a time, with no knowledge of future items, and makes an irrevocable decision for each, either accepting or rejecting the item. It cannot accept any item if its size, plus the sum of the sizes of the already accepted items, is greater than 1. The goal is to accept as many items as possible. The obvious greedy algorithm solves the offline Unit Profit Knapsack Problem, since the set consisting of as many of the smallest items that fit in the knapsack is an optimal solution.

Even for this special case of the Knapsack Problem, no competitive online algorithms can exist. Thus, we study the problem under the assumption that (an approximation of) the average item size, a , in an optimal solution is known to the algorithm. We study the



© Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 20; pp. 20:1–20:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

case, where the exact value of a is given to the algorithm as advice by an oracle, as well as the case where a is untrusted, e.g., estimated using machine learning. For instance, the characteristics of the input may be different depending on the time of day the input is produced, which source produced the input, etc. This could be learned to some extent and result in a prediction, which could be provided to the algorithm.

When considering machine-learned advice, the concepts of consistency and robustness are often considered, describing the balance between performing well on accurate advice and not doing too poorly when the advice is completely wrong. Our setting is different from most work on online algorithms with machine-learned advice, where there is generally a known online algorithm with a constant competitive ratio for the problem without advice. For this problem, if the advice is completely wrong, the algorithm cannot be competitive, since the problem without advice does not allow for competitive algorithms. Despite this hardness for the standard online version of the problem, we obtain results with untrusted predictions that are surprisingly consistent and robust.

1.1 Previous Work

The Knapsack Problem is well studied and comes in many variants; see Kellerer et al. [24]. Cygan et al. [18] refer to the online version we study, where all items give the same profit, as the *unit* case. They mention that it is well-known that no online algorithm for this version of the problem is competitive, i.e., has a finite competitive ratio. To verify this result, consider, for instance, the family of input sequences σ_j consisting of items of sizes $\frac{1}{j}$, $i = 1, 2, 3, \dots, j$.

In the General Knapsack Problem, each item comes not only with a size, but also with a profit, and the goal is to accept as much profit as possible given that the total size must be at most 1. The ratio of the profit to the size is the *importance*¹ of an item.

The Online Knapsack Problem was first studied by Marchetti-Spaccamela and Vercellis [34]; they prove that the problem does not allow for competitive online algorithms, even for Relaxed Knapsack (fractions of items may be accepted), where all item sizes are 1. They concentrate on a stochastic version of the problem, where both the profit and size coefficients are random variables.

The Online Unweighted (or Simple) Knapsack Problem with advice was studied in [14]. This is also called the proportional or uniform case. In this version, the importance of each item is equal to 1. They show that 1 bit of advice is sufficient to be $\frac{1}{2}$ -competitive, $\Omega(\log n)$ bits are necessary to be better than $\frac{1}{2}$ -competitive, and $n - 1$ advice bits are necessary and sufficient to be optimal. (As mentioned later, they also considered the General Knapsack Problem in the advice model.) The fundamental issues and many of the early results on oracle-based advice algorithms, primarily in the direction of advice complexity, can be found in [15], though many newer results for specific problems have been published since.

In [45], a knapsack problem is considered in a setting with machine-learned advice, with results incomparable to ours. In their setting, the General Knapsack Problem is considered, and results depend on upper and lower bounds on the importance of the items. The authors define limited classes of algorithms, based on a parameter, leading to some controlled degradation compared to an optimal competitive ratio. Within the defined classes, focus is then on tuning compared with historical data. Decisions to accept or reject an item are based on a threshold function based on the item's importance. Though the definition of this function is ad hoc, in the sense that it is not derived from some direct optimality criterion, it is well-motivated, aiming to coincide with the behavior found in optimal algorithms for the standard online algorithms setting.

¹ This is sometimes called *value*, but we want to avoid confusion with other uses of the word.

Recently, in [21], the General Knapsack Problem is revisited, again with upper and lower bounds on the possible importance of items. Machine-learned advice is given for each importance v , both an upper and a lower bound for the sum of the sizes of the items with importance v . The authors present an algorithm which has some similarities to ours. In particular their budget function has a similar function to our threshold function; both specify the maximum number of the low importance, large items that need to be accepted to obtain the proven competitive ratios. Their results can be extended to the case where the predictions are off by a small amount, the lower bounds can be divided by $1 + \varepsilon$, and the upper bounds can be multiplied by $1 + \varepsilon$. This is in contrast to ours, where robustness results are proven for arbitrarily large errors in the predictions, but only a is predicted. Since we have no bounds on the ratio of the largest to smallest size, those values do not enter into our results. Their algorithm obtains what they prove to be the optimal competitive ratio (for the given predictions), up to an additive factor that goes to zero as the size of the largest item goes to zero; this result has some of the flavor of our negative result. The authors also consider two related problems.

The Bin Packing Problem is closely related to the Knapsack Problem. This is especially true for the dual variant where the number of bins is fixed and the objective is to pack as many items as possible [17]; the Unit Price Knapsack Problem is Dual Bin Packing with one bin. The standard Bin Packing Problem was considered with machine learning in [3], considering a model of machine learning where, for a given algorithm, ALG, they consider a pair of values, $(r_{\text{ALG}}, w_{\text{ALG}})$, representing worst case ratios compared to the optimal offline algorithm, OPT. The value r_{ALG} gives the ratio for the best (trusted) advice and w_{ALG} gives the ratio for the worst possible (untrusted) advice. They use a parameter α in their algorithm, and show that their algorithm achieves values $(r, f(r))$ with $1.5 < r \leq 1.75$ and $f(r) = \max\{33 - 18r, 7/4\}$.

Bin Packing is also studied in [5] in the standard setting for online algorithms with machine learning, giving a trade-off between consistency and robustness, with the performance degrading as a function of the prediction error. They also have experimental results. Since the problem is so difficult, they have restricted their consideration to integer item sizes.

Much additional work has been done for other online problems, studying variants with untrusted predictions (machine-learned advice, for instance), initiated by the work of Lykouris and Vassilvitskii [32, 33] and Purohit et al. [39] in 2018, with further work in the directions of search-like problems [2, 6, 13, 29, 30, 35], scheduling [1, 4, 9, 20, 26, 27, 31, 36], rental problems [19, 25, 42], caching/paging [12, 22, 23, 40, 43], and other problems [5, 7, 8, 11, 37, 41], while some papers attack multiple problems [3, 10, 28, 44]. For a survey, see [38].

1.2 Preliminaries

We let a denote the average size of items accepted by the offline, optimal algorithm, OPT, that accepts as many of the smallest items as possible. Moreover, we let \hat{a} denote the “guessed” or predicted value of a . In the case of accurate advice (received from an oracle), $\hat{a} = a$. If \hat{a} may not be accurate, possibly determined via machine learning, and therefore not necessarily exactly a , we define a ratio r such that $a = r \cdot \hat{a}$. This particular advice is considered as a value that might be available or predictable, and the competitive ratios we present are a function of r .

We use the asymptotic competitive ratio throughout this paper. Thus, an algorithm ALG, is c -competitive if there exists a constant b such that for all request sequences σ , $\text{ALG}(\sigma) \geq c \text{OPT}(\sigma) - b$, where $\text{ALG}(\sigma)$ denotes ALG’s profit on σ . ALG’s competitive ratio is then $\sup\{c \mid \text{ALG is } c\text{-competitive}\}$. Note that this is a maximization problem and all competitive ratios are in the interval $[0, 1]$.

We use the notation $\mathbb{N} = \{0, 1, 2, \dots\}$. Define $\sum_{i=x}^y f(k)$ for some function f and real-valued x and y such that $y - x \in \mathbb{N}$ as $f(x) + f(x+1) + \dots + f(y)$. We generalize the Harmonic numbers by defining $H_k = \sum_{i=1+k-\lfloor k \rfloor}^k \frac{1}{i}$, for any real-valued $k \geq 1$. The following easy result is proven in the full paper [16].

► **Lemma 1.** *If $k \geq p \geq 1$ and $k - p \in \mathbb{N}$, then $\ln k - \ln(p+1) \leq H_k - H_p \leq \ln k - \ln p$.*

At any given time during the processing of the input sequence, the *level* of the knapsack denotes the total size of the items accepted.

1.3 Our Results

We consider both the case where the advice \hat{a} is known to be accurate, so $r = 1$, and the case where it might not be accurate. Different algorithms are presented for these two cases, but they have a common form.

For our algorithm ADAPTIVE THRESHOLD (AT) where the advice is accurate and, thus, $\hat{a} = a$, the competitive ratio is $\frac{e-1}{e}$, and we prove a matching upper bound that applies to any deterministic algorithm knowing a . This upper bound limits how well any algorithm using trusted predictions can do; the competitive ratio cannot be better than $\frac{e-1}{e} \approx 0.6321$ for $r = 1$.

If AT is used for untrusted predictions, it obtains a competitive ratio of $r \frac{e-1}{e}$ for $r \leq 1$, $\frac{e-r}{e}$ for $1 \leq r \leq e$, and 0 for $r \geq e$. No deterministic algorithm can be better than this for both $r < 1$ and $1 < r < e$.

For our algorithm, ADAPTIVE THRESHOLD FOR UNTRUSTED PREDICTIONS (ATUP) we have two cases: for $r \leq 1$ the competitive ratio is $\frac{r}{2}$, and for $r \geq 1$ the competitive ratio is $\frac{1}{2r}$. Thus, for accurate advice, the competitive ratio of ATUP is $\frac{1}{2}$, slightly less good than for the other algorithm. We show a negative result implying that a deterministic online algorithm cannot both be $\frac{1}{2r}$ -competitive for a range of large r -values and better than $\frac{1}{2}$ -competitive for $r = 1$.

Exact, oracle-based advice is not our focus point, though it is a crucial step in our work towards an algorithm for untrusted predictions. Thus, we do not emphasize the direction of advice complexity, where the focus is on the number of bits of oracle advice used to obtain given competitive ratios (or optimality), but we include a brief discussion in the full paper [16]. Instead, we focus on advice that may be easy to obtain. It seems believable that the average size of requests in an optimal solution would be information easily obtainable. The average size is probably a crucial component with regards to the profit secured by a process and quite possibly crucial with regards to supplying resources (knapsacks) over time. It is a single number (or two numbers: number of items and total size) to collect and store, as opposed to more detailed information about a distribution. So little storage is required that one could keep multiple copies if, for instance, the expected average changes during the day.

Given the simple optimal algorithm for the offline version of unit price knapsack, it seems obvious to consider another possibility for advice, the maximum size, s , for items to accept. However, this is insufficient, as there might be many items of that size, but the optimal solution may contain very few of them. Thus, one also needs further advice, including, for example, the fraction of the knapsack filled by items of size s . With these parameters given as advice, there would be two possibilities for the error. An extension of this idea is presented in [14], where the minimum importance is used, instead of the maximum size, for the General Knapsack Problem, giving k -bit approximations to the advice.

The full paper [16] contains the proofs missing from this paper. We treat the trusted as well as the untrusted case, and they build on similar ideas. To make this exposition accessible, we have emphasized giving a full account of the simpler, the trusted case, here, and refer the reader to the full paper for the more technical elements from the untrusted case.

2 The Adaptive Threshold Algorithm

In Algorithm 1, we introduce an algorithm template, which can be used to establish an oracle-based advice algorithm as well as an algorithm for untrusted predictions. The template omits the definition of a threshold function, T , since it is different for the two algorithms. In both algorithms, the threshold functions have the property that $T(i) > T(i + 1)$ for $i \geq 1$. We use the notation n_x to denote the number of accepted items strictly larger than x .

■ **Algorithm 1** Algorithm ADAPTIVE THRESHOLD.

```

1:  $\hat{a} \leftarrow$  predicted average size of OPT's accepted items
2: level  $\leftarrow$  0
3: for each input item  $x$  do
4:    $i = \max_{j \geq 0} \{n_{T(j+1)} = j\}$ 
5:   if  $\text{size}(x) \leq T(i + 1)$  and  $\text{level} + \text{size}(x) \leq 1$  then
6:     Accept  $x$ 
7:     level  $+= \text{size}(x)$ 
8:   else
9:     Reject  $x$ 

```

Intuitively, ADAPTIVE THRESHOLD accepts items that fit as long as it has not accepted too many items larger than the current item. The threshold functions are used to determine how many larger items is too many; no more than i items of size larger than $T(i + 1)$ are accepted. For smaller item sizes, this number of larger items is larger, since we need to accept more items if there are many small items.

Note that using $\max_{j \geq 0} \{n_{T(j+1)} \geq j\}$ instead of $\max_{j \geq 0} \{n_{T(j+1)} = j\}$ in Line 4 would result in the same algorithm. Thus, i is nondecreasing through the processing of the input sequence, and the value of the threshold function, $T(i)$, is decreasing in i , so larger items cannot be accepted after i increases.

3 Accurate Predictions

In this section, we give an $\frac{e-1}{e}$ -competitive algorithm which receives a , the average size of the items in OPT, as advice and prove that it is optimal among algorithms that get only a as advice.

3.1 Positive Result

To define an advice-based algorithm, we define a threshold function; see Algorithm 2. Throughout this section, we assume that $\hat{a} = a$, but the algorithm is also be used for untrusted predictions in Subsection 4.1.

First, we prove that AT with $\hat{a} = a$ has competitive ratio at least $\frac{e-1}{e} \approx 0.6321$.

► **Theorem 2.** *For $\hat{a} = a$, AT, as defined in Algorithm 2, is $\frac{e-1}{e}$ -competitive.*

20:6 Online Unit Profit Knapsack with Untrusted Predictions

■ **Algorithm 2** ADAPTIVE THRESHOLD with advice, AT.

-
- 1: Define $T(i) = \frac{\hat{a}e}{\hat{a}e(i-1) + 1}$ for $i \geq 1$
 - 2: Run ADAPTIVE THRESHOLD, Algorithm 1
-

Proof. If AT never rejects an item, it performs optimally. So assume it rejects an item at some point in the request sequence σ . Considering the conditional statement in the algorithm, if AT rejects an item, x , then either $\text{size}(x) > T(i+1)$ or $\text{level} + \text{size}(x) > 1$.

Case 1. This is the case where, at some point, AT rejects an item, x , because $\text{level} + \text{size}(x) > 1$.

The value of $T(k)$ from Algorithm 1 is an upper bound on the size of the k th largest item accepted by the algorithm. Thus, the k th largest accepted item has size at most

$$T(k) = \frac{ae}{ae(k-1) + 1} = \frac{1}{k-1 + \frac{1}{ae}}.$$

Using the definitions of sums over non-integer values (see Section 1.2), this gives an upper bound on the total size of items accepted by AT of

$$\text{level} \leq \sum_{k=1}^{\text{AT}(\sigma)} \frac{1}{k-1 + \frac{1}{ae}} = \sum_{k=\frac{1}{ae}}^{\text{AT}(\sigma) + \frac{1}{ae} - 1} \frac{1}{k} = H_{\text{AT}(\sigma) + \frac{1}{ae} - 1} - H_{\frac{1}{ae} - 1}.$$

Simple calculations (detailed in Lemma 1) give,

$$H_{\text{AT}(\sigma) + \frac{1}{ae} - 1} - H_{\frac{1}{ae} - 1} < \ln \left(\text{AT}(\sigma) + \frac{1}{ae} - 1 \right) - \ln \left(\frac{1}{ae} - 1 \right) = \ln \left(\frac{\text{AT}(\sigma) + \frac{1}{ae} - 1}{\frac{1}{ae} - 1} \right)$$

By assumption, $\text{level} + \text{size}(x) > 1$, and since $\text{level} \leq \ln \left(\frac{\text{AT} + \frac{1}{ae} - 1}{\frac{1}{ae} - 1} \right)$, we have

$$\begin{aligned} & \ln \left(\frac{\text{AT}(\sigma) + \frac{1}{ae} - 1}{\frac{1}{ae} - 1} \right) > 1 - \text{size}(x) \\ \Leftrightarrow & \frac{\text{AT}(\sigma) + \frac{1}{ae} - 1}{\frac{1}{ae} - 1} > e^{1 - \text{size}(x)} \\ \Leftrightarrow & \text{AT}(\sigma) > \left(\frac{1}{ae} - 1 \right) e^{1 - \text{size}(x)} - \frac{1}{ae} + 1 \\ \Leftrightarrow & \text{AT}(\sigma) > \frac{e^{1 - \text{size}(x)} - 1}{ae} + 1 - e^{1 - \text{size}(x)}. \end{aligned}$$

In the algorithm, i is at least zero, so we cannot accept items larger than $T(1) = ae$.

$$\begin{aligned}
 \text{AT}(\sigma) &> \frac{e^{1-\text{size}(x)} - 1}{ae} + 1 - e, \quad \text{since } -e^{1-\text{size}(x)} > -e \\
 &> \frac{e^{1-ae} - 1}{ae} + 1 - e, \quad \text{by the observation above} \\
 &\geq \frac{e - e^2a - 1}{ae} + 1 - e, \quad \text{by simple calculations; see the full paper [16]} \\
 &= \frac{e - 1}{ae} - 2e + 1 \\
 &\geq \frac{e - 1}{e} \text{OPT}(\sigma) - 2e + 1, \quad \text{since } \text{OPT}(\sigma) \leq \frac{1}{a}
 \end{aligned}$$

So, $\lim_{\text{OPT} \rightarrow \infty} \frac{\text{AT}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{e-1}{e}$.

Case 2. This is the case where AT never rejects any item, x , when $\text{size}(x) \leq T(i + 1)$. Let i_t denote the final value of i as the algorithm terminates. Suppose OPT accepts ℓ items larger than $T(i_t + 1)$ and s items of size at most $T(i_t + 1)$. Since OPT accepts ℓ items larger than $T(i_t + 1)$ and $\ell + s$ items in total, we have $a > \ell \cdot T(i_t + 1) / (\ell + s)$, which is equivalent to

$$s > \left(\frac{T(i_t + 1)}{a} - 1 \right) \ell \tag{1}$$

By the definition of T , we have that $T(i_t + 1) = \frac{ae}{ae^{i_t+1}}$. Solving for the i_t on the right-hand side, we get

$$i_t = \frac{1}{T(i_t + 1)} - \frac{1}{ae}. \tag{2}$$

Thus, AT has accepted at least $i_t = \frac{1}{T(i_t+1)} - \frac{1}{ae}$ items of size greater than $T(i_t + 1)$. Further, due to the assumption in this second case, AT has accepted all of the s items no larger than $T(i_t + 1)$. To see this, note that the i s of the algorithm can only increase, so at no point has there been a size demand more restrictive than $T(i_t + 1)$.

We split in two subcases, depending on how $T(i_t + 1)$ relates to OPT's average size, a .

Subcase 2a: $T(i_t + 1) > a$. In this subcase, the lower bound on s of Ineq. (1) is positive.

$$\begin{aligned}
 \frac{\text{AT}(\sigma)}{\text{OPT}(\sigma)} &\geq \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae} \right) + s}{\ell + s}, \quad \text{by Eq. (2)} \\
 &> \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae} \right) + \left(\frac{T(i_t+1)}{a} - 1 \right) \ell}{\ell + \left(\frac{T(i_t+1)}{a} - 1 \right) \ell}, \quad \text{by Ineq. (1)} \\
 &= \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae} \right) + \left(\frac{T(i_t+1)}{a} - 1 \right) \ell}{\frac{T(i_t+1)}{a} \ell}.
 \end{aligned}$$

The second inequality follows since the ratio is smaller than one and s is replaced by a smaller, positive term in the numerator as well as the denominator.

20:8 Online Unit Profit Knapsack with Untrusted Predictions

We prove that this is bounded from below by $\frac{e-1}{e}$:

$$\begin{aligned}
 & \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae}\right) + \left(\frac{T(i_t+1)}{a} - 1\right)\ell}{\frac{T(i_t+1)\ell}{a}} \geq \frac{e-1}{e} \\
 \Leftrightarrow & \frac{e}{T(i_t+1)} - \frac{1}{a} + e\left(\frac{T(i_t+1)}{a} - 1\right)\ell \geq e\frac{T(i_t+1)}{a}\ell - \frac{T(i_t+1)}{a}\ell \\
 \Leftrightarrow & \frac{e}{T(i_t+1)} - \frac{1}{a} \geq \left(e - \frac{T(i_t+1)}{a}\right)\ell \\
 \Leftrightarrow & \frac{ea - T(i_t+1)}{aT(i_t+1)} \geq \frac{ea - T(i_t+1)}{a}\ell \\
 \Leftrightarrow & \frac{1}{T(i_t+1)} \geq \ell
 \end{aligned}$$

For the last biimplication, we must argue that $ea - T(i_t+1) \geq 0$, but this holds since $T(1) = ea$ and T is decreasing. Finally, the last statement, $\frac{1}{T(i_t+1)} \geq \ell$ holds regardless of the relationship between $T(i_t+1)$ and a , since the knapsack obviously cannot hold more than $\frac{1}{T(i_t+1)}$ items of size greater than $T(i_t+1)$.

Subcase 2b: $T(i_t+1) \leq a$.

$$\begin{aligned}
 \frac{\text{AT}(\sigma)}{\text{OPT}(\sigma)} & \geq \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae}\right) + s}{\ell + s}, \text{ by Eq. (2)} \\
 & \geq \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae}\right)}{\ell}, \quad \text{since } s \geq 0 \text{ and } \frac{\text{AT}(\sigma)}{\text{OPT}(\sigma)} \leq 1 \\
 & > \frac{\left(\frac{1}{T(i_t+1)} - \frac{1}{ae}\right)}{\frac{1}{T(i_t+1)}}, \quad \text{since, as above, } \ell \leq \frac{1}{T(i_t+1)} \\
 & = 1 - \frac{T(i_t+1)}{ae} \\
 & \geq 1 - \frac{a}{ae}, \quad \text{by the subcase we are in} \\
 & = \frac{e-1}{e}.
 \end{aligned}$$

This concludes the second case, and, thus, the proof. ◀

3.2 Negative Result

Now, we show that AT is optimal among online algorithms knowing a and nothing else.

► **Theorem 3.** *Any deterministic algorithm getting only a as advice has a competitive ratio of at most $\frac{e-1}{e}$.*

■ **Algorithm 3** Adversarial sequence establishing optimality with advice.

▷ Assume $a < \frac{1}{2e}$ and $\frac{1}{a} \in \mathbb{N}$

- 1: $\varepsilon \leftarrow \frac{a^2}{10}$
- 2: $k \leftarrow \lfloor \frac{1}{ae} \rfloor$
- 3: **while** ALG's level $\leq 1 - \frac{1}{k} - k\varepsilon$ **do**
- 4: **for** k times **do**
- 5: Give an item of size $\frac{1}{k} - \varepsilon$
- 6: **if** ALG accepts **then**
- 7: $k++$
- 8: **continue** (* the while-loop *)
- 9: ▷ ALG did not accept any of the k items of this round.
- 9: Give $\frac{1}{a} - k$ items of size $\frac{k a \varepsilon}{1 - k a}$
- 10: **terminate** ▷ Case 1
- 11: Give $\frac{1}{a}$ items of size a ▷ Case 2

Proof. Let ALG denote the online algorithm with advice, and let σ be the adversarial sequence defined by Algorithm 3, which explains how the adversary defines its sequence based on ALG's actions.

Let k_t be the value of k at the beginning of the last iteration of the while-loop. We perform a case analysis based on how the generation of the adversarial sequence terminates.

Case 1. OPT accepts the k_t items of size $\frac{1}{k_t} - \varepsilon$ in the last iteration of the while-loop and the $\frac{1}{a} - k_t$ items of size $\frac{k_t a \varepsilon}{1 - k_t a}$ for a total of $\frac{1}{a}$ items of total size

$$k_t \left(\frac{1}{k_t} - \varepsilon \right) + \left(\frac{1}{a} - k_t \right) \frac{k_t a \varepsilon}{1 - k_t a} = 1 - k_t \varepsilon + (1 - k_t a) \frac{k_t \varepsilon}{1 - k_t a} = 1.$$

Note that the average size of the items accepted by OPT is a , consistent with the advice.

ALG accepts one item in each iteration of the while-loop, except the last iteration, and at most $\frac{1}{a} - k_t$ items after that, so no more than

$$k_t - \left\lfloor \frac{1}{ae} \right\rfloor + \frac{1}{a} - k_t < \frac{1}{a} - \frac{1}{ae} + 1 = \frac{e-1}{e} \cdot \frac{1}{a} + 1.$$

$$\text{Thus, } \text{ALG}(\sigma) \leq \frac{e-1}{e} \cdot \frac{1}{a} + 1 = \frac{e-1}{e} \text{OPT}(\sigma) + 1.$$

Case 2. OPT accepts the $\frac{1}{a}$ items of size a .

For the analysis of ALG, we start by establishing an upper bound on k_t . The following inequality holds since ALG accepts one item per round, and ALG's level just before the last round is at most $1 - \frac{1}{k_t} - k_t \varepsilon$ before the last item of size $\frac{1}{k_t} - \varepsilon$ is accepted.

$$\begin{aligned}
 & \sum_{k=\lfloor \frac{1}{ae} \rfloor}^{k_t} \left(\frac{1}{k} - \varepsilon \right) \leq 1 - (k_t + 1)\varepsilon \\
 \Downarrow & \\
 & H_{k_t} - H_{\lfloor \frac{1}{ae} \rfloor - 1} - k_t \varepsilon < 1 - k_t \varepsilon \\
 \Updownarrow & \\
 & H_{k_t} - H_{\lfloor \frac{1}{ae} \rfloor - 1} < 1 \\
 \Downarrow & \\
 & \ln(k_t) - \ln \left(\left\lfloor \frac{1}{ae} \right\rfloor \right) < 1, \text{ by Lemma 1} \\
 \Updownarrow & \\
 & k_t < e \left\lfloor \frac{1}{ae} \right\rfloor
 \end{aligned}$$

In the case we are treating, ALG leaves the while-loop because its level is more than $1 - \frac{1}{k_t + 1} - (k_t + 1)\varepsilon$. Now, we give a bound on the amount of space available at that point. For the first inequality, note that by the initialization of k in the algorithm, $k_t \geq \lfloor \frac{1}{ae} \rfloor$.

$$\begin{aligned}
 \frac{1}{k_t + 1} + (k_t + 1)\varepsilon &< \frac{1}{\lfloor \frac{1}{ae} \rfloor + 1} + \left(e \left\lfloor \frac{1}{ae} \right\rfloor + 1 \right) \varepsilon < ae + \left(\frac{1}{a} + 1 \right) \frac{a^2}{10} \\
 &< \left(e + \left(\frac{1+a}{10} \right) \right) a < 3a
 \end{aligned}$$

Thus, after the while-loop, ALG can accept at most two of the items of size a . Clearly, the number of rounds in the while-loop is $k_t - \lfloor \frac{1}{ae} \rfloor + 1$. Using $k_t < e \lfloor \frac{1}{ae} \rfloor$, we can now bound ALG's profit:

$$\text{ALG}(\sigma) \leq k_t - \left\lfloor \frac{1}{ae} \right\rfloor + 1 + 2 < (e-1) \left\lfloor \frac{1}{ae} \right\rfloor + 3 \leq \frac{e-1}{e} \frac{1}{a} + 3 = \frac{e-1}{e} \text{OPT}(\sigma) + 3$$

This establishes the bound on the competitive ratio of $\frac{e-1}{e}$.

Finally, to ensure that our proof is valid, we must argue that the number of rounds we count in the algorithm and the sizes of items we give are non-negative. For the remainder of this proof, we go through the terms in the algorithm, thereby establishing this.

The largest value of k in the algorithm is k_t , and we have established that $k_t < e \lfloor \frac{1}{ae} \rfloor < \frac{1}{a}$. Additionally, from the start value of k , we know that $\lfloor \frac{1}{ae} \rfloor \leq k$. Using these facts, together with the assumption from the algorithm that $a < \frac{1}{2e}$, we get the following bounds on the various terms.

$$1 - \frac{1}{k} - k\varepsilon > 1 - \frac{1}{\lfloor \frac{1}{ae} \rfloor} - \frac{1}{a} \frac{a^2}{10} > 1 - \frac{1}{\lfloor \frac{1}{1/2} \rfloor} - \frac{1}{20e} > 0$$

Further, $\frac{1}{k} - \varepsilon \geq \frac{1}{k_t} - \varepsilon > \frac{1}{\frac{1}{a}} - \frac{a^2}{10} > 0$ and $\frac{1}{a} - k \geq \frac{1}{a} - k_t > \frac{1}{a} - \frac{1}{a} = 0$.

For the last relevant value, $1 - ka \geq 1 - k_t a > 1 - \frac{1}{a} a = 0$ and from Case 1, we know that the $\frac{1}{a} - k_t$ items given in Line 9 of the algorithm sum up to at most one. \blacktriangleleft

4 Untrusted Predictions

Some of our main results are found in this section, but the proofs build on ideas from the trusted case. To make the exposition accessible, we have emphasized explaining the ideas in the simpler setting. The reader is referred to the full paper [16] for the missing details.

For the case where the predictions may be inaccurate, the algorithm AT can be used with \hat{a} possibly not being a as long as $r < e$, see Subsection 4.1. In Subsection 4.2, we give an adaptive threshold algorithm, ATUP, that works for all r .

For $r < \frac{1}{2}(e + \sqrt{e^2 - 2e}) \approx 2.06$, AT has a better competitive ratio than ATUP. Thus, if an upper bound on r of approximately 2 (or lower) is known, AT may be preferred, and if a guarantee for any r is needed, ATUP should be used.

4.1 Semi-Trusted Predictions

In this section, we consider the algorithm AT with a semi-trusted (being guaranteed that $r < e$) prediction, \hat{a} , instead of a . See the full paper [16] for the proofs.

Using a proof analogous to the proof of Theorem 2, we get the following result.

► **Theorem 4.** *For untrusted advice, AT has a competitive ratio of at least*

$$c_{AT}(r) \geq \begin{cases} \frac{e-1}{e} \cdot r, & \text{if } r \leq 1 \\ \frac{e-r}{e}, & \text{if } r \geq 1 \end{cases}$$

► **Theorem 5.** *If a deterministic algorithm is $\frac{e-r}{e}$ -competitive for all $1 \leq r < e$, it cannot be better than $r \cdot \frac{e-1}{e}$ -competitive for any $r \leq 1$.*

If a deterministic algorithm is better than $r \cdot \frac{e-1}{e}$ -competitive for some $r \leq 1$, it cannot be $\frac{e-r}{e}$ -competitive for all $1 \leq r < e$.

Combining Theorems 4 and 5, we obtain that, if r is guaranteed to be smaller than e , no deterministic algorithm can be better than AT for both $r < 1$ and $r > 1$. Moreover, we get the following tight result on the performance of AT.

► **Theorem 6.** *AT has a competitive ratio of*

$$c_{AT}(r) = \begin{cases} \frac{e-1}{e} \cdot r, & \text{if } r \leq 1 \\ \frac{e-r}{e}, & \text{if } 1 \leq r \leq e \\ 0, & \text{if } r \geq e \end{cases}$$

4.2 Untrusted Predictions

4.2.1 Positive Result

When considering the case where the average item size is estimated to be \hat{a} , and the accurate value is $a = r \cdot \hat{a}$, we consider two cases, $r > 1$ and $r < 1$. In either case, we have the problem that we do not even know which case we are in, so, when large items arrive, we have to accept some to be competitive. The algorithm we consider when the value of r is not necessarily one achieves similar competitive ratios in both cases. Algorithm 4, ATUP, is ADAPTIVE THRESHOLD with a different threshold function than was used for accurate advice (and in AT).

Since we need to accept larger items than in the case of accurate advice, we need a threshold function that decreases faster than the threshold function used in Section 3, in order not to risk filling up the knapsack before the small items arrive. Therefore, it may seem surprising that we are using a threshold function that decreases as $\frac{1}{\sqrt{i}}$, when the threshold function of Section 3 decreases as $\frac{1}{i}$. However, the $\frac{1}{i}$ -function of the algorithm for accurate advice is essentially offset by $\frac{1}{ae}$.

20:12 Online Unit Profit Knapsack with Untrusted Predictions

■ **Algorithm 4** ADAPTIVE THRESHOLD FOR UNTRUSTED PREDICTIONS, ATUP.

-
- 1: Define $T(i) = \sqrt{\frac{\hat{a}}{2^i}}$ for $i \geq 1$
 - 2: Run ADAPTIVE THRESHOLD, Algorithm 1
-

We prove a number of more or less technical results before stating the positive results for $r \leq 1$ (Theorem 11) and $r \geq 1$ (Theorem 10). See the full paper [16] for those missing here, along with the missing proofs.

► **Lemma 7.** *For any $k \geq 1$, the total size of the k largest items accepted by ATUP is at most $\sqrt{2k\hat{a}}$.*

The following corollary implies that ATUP never rejects an item based on the level being too high if $r > 2$. This is because $r > 2$ means that the items in OPT are relatively large compared to \hat{a} . Since OPT accepts the smallest items of the sequence, it means that the sequence contains relatively few small items. Thus, the algorithm reserves space for small items that never arrive.

► **Corollary 8.** *If ATUP rejects an item based on the level being too high, $ATUP(\sigma) > \frac{r}{2} OPT(\sigma) - 1$.*

► **Lemma 9.** *Assume that OPT accepts ℓ items larger than $\sqrt{\frac{\hat{a}}{2^{(i+1)}}}$ and s items of size at most $\sqrt{\frac{\hat{a}}{2^{(i+1)}}}$, $i \geq 0$. Then, the following inequalities hold:*

1. $s > \ell \left(\frac{1}{r\sqrt{2\hat{a}(i+1)}} - 1 \right)$
2. $\ell < \sqrt{\frac{2(i+1)}{\hat{a}}}$

For the case where the actual average size in OPT's packing is at least as large as the predicted average size, we get the following result.

► **Theorem 10.** *For all request sequences σ , such that $r \geq 1$,*

$$ATUP(\sigma) \geq \frac{1}{2r} OPT(\sigma) - 1.$$

Proof. By Corollary 8, if ATUP rejects an item in σ due to the knapsack not having room for the item, $ATUP(\sigma) \geq \frac{r}{2} OPT(\sigma) - 1 \geq \frac{1}{2r} OPT(\sigma) - 1$ for $r \geq 1$.

Now, suppose that ATUP does not reject any item due to it not fitting in the knapsack. If it is not optimal, it must reject due to the size of the item.

Let i_t denote the final value of i when the algorithm is run. This means that ATUP has accepted i_t items of size greater than $\sqrt{\frac{\hat{a}}{2^{(i_t+1)}}}$. We perform a case analysis based on whether this value is smaller or larger than $r\hat{a}$.

Case 1: $r \geq \frac{1}{\sqrt{2\hat{a}(i_t+1)}}$. In this case, $i_t + 1 \geq \frac{1}{2r^2\hat{a}}$ and $OPT(\sigma) \leq \frac{1}{r\hat{a}} \leq \sqrt{\frac{2(i_t+1)}{\hat{a}}}$. Thus,

$$\frac{ATUP(\sigma) + 1}{OPT(\sigma)} \geq \frac{i_t + 1}{\sqrt{\frac{2(i_t+1)}{\hat{a}}}} = \sqrt{\frac{\hat{a}(i_t+1)}{2}} \geq \sqrt{\frac{\hat{a} \cdot \frac{1}{2r^2\hat{a}}}{2}} = \frac{1}{2r}.$$

Therefore, $ATUP(\sigma) \geq \frac{1}{2r} OPT(\sigma) - 1$.

Case 2: $r < \frac{1}{\sqrt{2\hat{a}(i_t+1)}}$. Suppose OPT accepts ℓ items larger than $\sqrt{\frac{\hat{a}}{2(i_t+1)}}$ and s items of size at most $\sqrt{\frac{\hat{a}}{2(i_t+1)}}$. Note that ATUP also accepts the s items of size at most $\sqrt{\frac{\hat{a}}{2(i_t+1)}}$, since we are in the case where it does not reject items because of the knapsack being too full.

Given the input sequence σ , we consider the ratio

$$\frac{\text{ATUP}(\sigma) + 1}{\text{OPT}(\sigma)} \geq \frac{(i_t + 1) + s}{\ell + s}.$$

The result follows if this ratio is always at least $\frac{1}{2r}$.

Subcase 2a: $i_t + 1 \geq \frac{1}{2\hat{a}}$. In this case, $\text{ATUP}(\sigma) \geq i_t \geq \frac{1}{2\hat{a}} - 1$, while $\text{OPT}(\sigma) \leq \frac{1}{r\hat{a}}$. Thus, $\text{ATUP}(\sigma) \geq \frac{r}{2} \text{OPT}(\sigma) - 1$.

Subcase 2b: $i_t + 1 < \frac{1}{2\hat{a}}$. By Ineq. 1 of Lemma 9, and since $\frac{\text{ATUP}(\sigma)+1}{\text{OPT}(\sigma)} \leq 1$,

$$\frac{\text{ATUP}(\sigma) + 1}{\text{OPT}(\sigma)} \geq \frac{(i_t + 1) + s}{\ell + s} \geq \frac{(i_t + 1) + \left(\frac{1}{r\sqrt{2\hat{a}(i_t+1)}} - 1\right) \ell}{\frac{\ell}{r\sqrt{2\hat{a}(i_t+1)}}}.$$

In the full paper, we show that this is at least $\frac{1}{2r}$. ◀

For the case where the actual average size in OPT's packing is no larger than the predicted average size, we get the following result.

► **Theorem 11.** *For all request sequences σ , such that $r < 1$,*

$$\text{ATUP}(\sigma) \geq \frac{r}{2} \text{OPT}(\sigma) - 1.$$

4.2.2 Negative Result

In Section 3, we showed that, even with accurate advice, no deterministic algorithm can be better than $\frac{e-1}{e}$ -competitive. In this section, we give a trade-off in the competitive ratio attained for different values of r .

► **Theorem 12.** *Let $0 < z \leq 2$ and consider a deterministic algorithm, ALG.*

If ALG is $\frac{1}{zr}$ -competitive for every r between $\frac{2}{z}$ and $\frac{1}{\sqrt{z\hat{a}}}$, it cannot be better than $\frac{zr}{4}$ -competitive, for any $r \leq \frac{2}{z}$.

Moreover, if ALG is better than $\frac{zr}{4}$ -competitive for some $r \leq \frac{2}{z}$, it cannot be $\frac{1}{\sqrt{z\hat{a}}}$ -competitive for all r between $\frac{2}{z}$ and $\frac{1}{\sqrt{z\hat{a}}}$.

Proof. We consider the adversary that gives the input sequence σ_z defined by Algorithm 5. Consider an online algorithm, ALG, and assume that there exists a constant, b , such that $\text{ALG}(\sigma) \geq \frac{1}{zr} \text{OPT}(\sigma) - b$, for any sequence σ and any r such that $\frac{2}{z} \leq r \leq \frac{1}{\sqrt{z\hat{a}}}$. Now, consider the adversary that gives the input sequence σ_z defined by Algorithm 5.

If the adversarial algorithm terminates in Line 6, then, ALG has accepted at most $k - b - 1$ items. In this case, $a = \sqrt{\frac{\hat{a}}{zk}}$, and OPT accepts exactly the $\left\lfloor \sqrt{\frac{zk}{\hat{a}}} \right\rfloor$ items from the last iteration of the while-loop. Since $a = r\hat{a}$, $r = \sqrt{\frac{1}{zk\hat{a}}}$, which lies between

20:14 Online Unit Profit Knapsack with Untrusted Predictions

■ **Algorithm 5** Adversarial sequence establishing trade-off on robustness versus consistency. The adversarial algorithm takes parameters, z , q , and b , such that $0 < z \leq 2$, $0 < q < \frac{1}{\sqrt{z\hat{a}}}$, and $b \geq 0$.

▷ Assume $\frac{1}{q\hat{a}} \in \mathbb{N}$

- 1: $p \leftarrow \lfloor \frac{z}{4\hat{a}} \rfloor$
- 2: $k \leftarrow 0$
- 3: **while** $k \leq p - 1$ **do**
- 4: $k++$
- 5: Give $\lfloor \sqrt{\frac{zk}{\hat{a}}} \rfloor$ items of size $\sqrt{\frac{\hat{a}}{zk}}$
- 6: **if** ALG has accepted fewer than $k - b$ items **then terminate**
- 7: Give $\frac{1}{q\hat{a}}$ items of size $q\hat{a}$

$\sqrt{\frac{1}{zp\hat{a}}} \geq \sqrt{\frac{1}{z\hat{a}} \cdot \frac{4\hat{a}}{z}} = \frac{2}{z}$ and $\frac{1}{\sqrt{z\hat{a}}}$. Thus,

$$\begin{aligned} \text{ALG}(\sigma_z) &\leq k - b - 1 \leq \frac{k - 1}{\lfloor \sqrt{\frac{zk}{\hat{a}}} \rfloor} \text{OPT}(\sigma_z) - b < \frac{k - 1}{\sqrt{\frac{zk}{\hat{a}}} - 1} \text{OPT}(\sigma_z) - b \\ &< \frac{k}{\sqrt{\frac{zk}{\hat{a}}}} \text{OPT}(\sigma_z) - b = \sqrt{\frac{k\hat{a}}{z}} \text{OPT}(\sigma_z) - b = \frac{1}{zr} \text{OPT}(\sigma_z) - b, \end{aligned}$$

where the second strict inequality holds because 1 is added to the numerator and denominator of a positive fraction less than 1. This contradicts the assumption that for each r between $\frac{2}{z}$ and $\frac{1}{\sqrt{z\hat{a}}}$, $\text{ALG}(\sigma) \geq \frac{1}{zr} \text{OPT}(\sigma) - b$, for any sequence σ , when the adversarial algorithm terminates in Line 6. Thus, the adversarial algorithm does not terminate there.

If the adversarial algorithm does not terminate in Line 6, $r = q$ and $\text{OPT}(\sigma_z) = \frac{1}{q\hat{a}} = \frac{1}{r\hat{a}}$. Moreover, for ALG, the i th accepted item must have size at least $\sqrt{\frac{\hat{a}}{z(i+b)}}$, for $1 \leq i \leq p - b$. Thus, these first $p - b$ items fill the knapsack to at least

$$\sum_{i=b+1}^p \sqrt{\frac{\hat{a}}{zi}} \geq \sqrt{\frac{\hat{a}}{z}} \int_{b+1}^{p+1} \frac{1}{\sqrt{i}} di = \sqrt{\frac{\hat{a}}{z}} (2\sqrt{p+1} - 2\sqrt{b+1}),$$

where we use that $\frac{1}{\sqrt{i}}$ is a decreasing function.

Since the items of size $r\hat{a}$ are the smallest items of the sequence, this means that

$$\begin{aligned} \text{ALG}(\sigma_z) &\leq p + \frac{1 - \sqrt{\frac{\hat{a}}{z}} (2\sqrt{p+1} - 2\sqrt{b+1})}{r\hat{a}} \\ &\leq \frac{z}{4\hat{a}} + \frac{1 - \sqrt{\frac{\hat{a}}{z}} (2\sqrt{\frac{z}{4\hat{a}}} - 2\sqrt{b+1})}{r\hat{a}} \\ &= \frac{z}{4\hat{a}} + \frac{1 - 1 + 2\sqrt{\frac{\hat{a}(b+1)}{z}}}{r\hat{a}} \\ &= \frac{1}{r\hat{a}} \left(\frac{zr}{4} + 2\sqrt{\frac{\hat{a}(b+1)}{z}} \right) \\ &= \left(\frac{zr}{4} + 2\sqrt{\frac{\hat{a}(b+1)}{z}} \right) \text{OPT}(\sigma_z). \end{aligned}$$

As a function of \hat{a} , the upper bound is $\frac{zr}{4} + 2\sqrt{\frac{\hat{a}(b+1)}{z}}$, but the second term becomes insignificant as \hat{a} approaches zero. This proves the first part of the theorem.

The second part of the theorem is the contrapositive of the first part. ◀

Setting $z = 2$ in Theorem 12 demonstrates a Pareto-like trade-off between consistency and robustness for ATUP:

► **Corollary 13.** *Consider a deterministic algorithm, ALG.*

If ALG is $\frac{1}{2r}$ -competitive for every r between 1 and $\frac{1}{\sqrt{2a}}$, it has a competitive ratio of at most $\frac{r}{2}$, for any positive $r \leq 1$.

Moreover, if ALG is better than $\frac{r}{2}$ -competitive for some $r \leq 1$, it cannot be $\frac{1}{2r}$ -competitive for all r between 1 and $\frac{1}{\sqrt{2a}}$.

References

- 1 Sara Ahmadian, Hossein Esfandiari, Vahab Mirrokni, and Binghui Peng. Robust load balancing with machine learned advice. In *2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 20–34. SIAM, 2022.
- 2 Spyros Angelopoulos. Online search with a hint. In *12th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 185 of *LIPICs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 3 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *LIPICs*, pages 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 4 Spyros Angelopoulos and Shahin Kamali. Contract scheduling with predictions. In *35th AAAI Conference on Artificial Intelligence (AAAI), 33rd Conference on Innovative Applications of Artificial Intelligence (IAAI), 11th Symposium on Educational Advances in Artificial Intelligence (EAAI)*, pages 11726–11733. AAAI Press, 2021.
- 5 Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. *ArXiv*, 2021. [arXiv:2102.03311](https://arxiv.org/abs/2102.03311).
- 6 Spyros Angelopoulos, Shahin Kamali, and Dehou Zhang. Online search with best-price and query-based predictions. *ArXiv*, 2021. [arXiv:2112.01592](https://arxiv.org/abs/2112.01592).
- 7 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.
- 8 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 7933–7944. Curran Associates, Inc., 2020.
- 9 Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *33rd Annual conference on Neural Information Processing Systems (NeurIPS)*, pages 15350–15359. Curran Associates, Inc., 2020.
- 10 Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *33rd Annual conference on Neural Information Processing Systems (NeurIPS)*, pages 20083–20094. Curran Associates, Inc., 2020.
- 11 Siddhartha Banerjee, Vasilis Gkatzelis, Artur Gorokh, and Billy Jin. Online nash social welfare maximization with predictions. In *2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–19. SIAM, 2022.
- 12 Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-augmented weighted paging. In *2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 67–89. SIAM, 2022.

20:16 Online Unit Profit Knapsack with Untrusted Predictions

- 13 Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online learning with imperfect hints. In *37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 822–831. PMLR, 2020.
- 14 Hans-Joachim Böckenhauer, Dennis Komm, Richard Kráľovič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theoretical Computer Science*, 527:61–72, 2014.
- 15 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online Algorithms with Advice: A Survey. *ACM Computing Surveys*, 50(2):1–34, 2017. Article No. 19.
- 16 Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online unit profit knapsack with untrusted predictions. *ArXiv*, 2022. [arXiv:2203.00285](https://arxiv.org/abs/2203.00285).
- 17 Joan Boyar, Lene M. Favrholdt, Kim S. Larsen, and Morten N. Nielsen. The competitive ratio for on-line dual bin packing with restricted input sequences. *Nordic Journal of Computing*, 8:463–472, 2001.
- 18 Marek Cygan, Łukasz Jeż, and Jiri Sgall. Online knapsack revisited. *Theory of Computing Systems*, 58, 2016.
- 19 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019.
- 20 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 285–294. ACM, 2021.
- 21 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. In *Pre-Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- 22 Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrović, and Ronitt Rubinfeld. Online page migration with ML advice. *ArXiv*, 2020. [arXiv:2006.05028](https://arxiv.org/abs/2006.05028).
- 23 Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *LIPICs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 24 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.
- 25 Rohan Kodialam. Optimal algorithms for ski rental with soft machine-learned predictions. *ArXiv*, 2019. [arXiv:1903.00092](https://arxiv.org/abs/1903.00092).
- 26 Arvind Kumar and Bashir Alam. Task scheduling in real time systems with energy harvesting and energy minimization. *Journal of Computational Science*, 14(8):1126–1133, 2018.
- 27 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1859–1877. SIAM, 2020.
- 28 Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and Instance-Robust Predictions for Online Matching, Flows and Load Balancing. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPICs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 29 Russell Lee, Mohammad H. Hajiesmaili, and Jian Li. Learning-assisted competitive algorithms for peak-aware energy scheduling. *ArXiv*, 2020. [arXiv:1911.07972](https://arxiv.org/abs/1911.07972).
- 30 Russell Lee, Jessica Maghakian, Mohammad H. Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. Online peak-aware energy scheduling with untrusted advice. In *12th ACM International Conference on Future Energy Systems (e-Energy)*, pages 107–123. ACM, 2021.
- 31 Shi Li and Jiayi Xian. Online unrelated machine load balancing with predictions revisited. In *38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 6523–6532. PMLR, 2021.

- 32 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *35th International Conference on Machine Learning (ICML)*, volume 80, pages 3302–3311. PMLR, 2018.
- 33 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):24:1–24:25, 2021.
- 34 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995.
- 35 Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *30th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1858–1866. Curran Associates, Inc., 2017.
- 36 Michael Mitzenmacher. Scheduling with Predictions and the Price of Misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *LIPICs*, pages 14:1–14:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.
- 37 Michael Mitzenmacher. Queues with small advice. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 1–12, 2021.
- 38 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *ArXiv*, 2020. [arXiv:2006.09123](https://arxiv.org/abs/2006.09123).
- 39 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9661–9670. Curran Associates, Inc., 2018.
- 40 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1834–1845. SIAM, 2020.
- 41 Daan Rutten and Debankur Mukherjee. A new approach to capacity scaling augmented with unreliable machine learning predictions. *ArXiv*, 2021. [arXiv:2101.12160](https://arxiv.org/abs/2101.12160).
- 42 Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 2035–2037. International Foundation for Autonomous Agents and Multiagent Systems, 2020.
- 43 Alexander Wei. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176 of *LIPICs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 44 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. *ArXiv*, 2020. [arXiv:2010.11443](https://arxiv.org/abs/2010.11443).
- 45 Ali Zeynali, Bo Sun Mohammad Hajiesmaili, and Adam Wierman. Data-driven competitive algorithms for online knapsack and set cover. In *35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Nearest-Neighbor Decompositions of Drawings*

Jonas Cleve  

Institut für Informatik, Freie Universität Berlin, Germany

Nicolas Grelier 

Department of Computer Science, ETH Zürich, Switzerland

Kristin Knorr  

Institut für Informatik, Freie Universität Berlin, Germany

Maarten Löffler 

Utrecht University, The Netherlands

Wolfgang Mulzer  

Institut für Informatik, Freie Universität Berlin, Germany

Daniel Perz  

Technische Universität Graz, Austria

Abstract

Let \mathcal{D} be a set of straight-line segments in the plane, potentially crossing, and let c be a positive integer. We denote by P the union of the endpoints of the straight-line segments of \mathcal{D} and of the intersection points between pairs of segments. We say that \mathcal{D} has a *nearest-neighbor decomposition* into c parts if we can partition P into c point sets P_1, \dots, P_c such that \mathcal{D} is the union of the nearest neighbor graphs on P_1, \dots, P_c . We show that it is NP-complete to decide whether \mathcal{D} can be drawn as the union of $c \geq 3$ nearest-neighbor graphs, even when no two segments cross. We show that for $c = 2$, it is NP-complete in the general setting and polynomial-time solvable when no two segments cross. We show the existence of an $O(\log n)$ -approximation algorithm running in subexponential time for partitioning \mathcal{D} into a minimum number of nearest-neighbor graphs.

As a main tool in our analysis, we establish the notion of the *conflict graph* for a drawing \mathcal{D} . The vertices of the conflict graph are the connected components of \mathcal{D} , with the assumption that each connected component is the nearest neighbor graph of its vertices, and there is an edge between two components U and V if and only if the nearest neighbor graph of $U \cup V$ contains an edge between a vertex in U and a vertex in V . We show that string graphs are conflict graphs of certain planar drawings. For planar graphs and complete k -partite graphs, we give additional, more efficient constructions. We furthermore show that there are subdivisions of non-planar graphs that are not conflict graphs. Lastly, we show a separator lemma for conflict graphs.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases nearest-neighbors, decompositions, drawing

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.21

Funding *Jonas Cleve*: Supported in part by ERC StG 757609.

Nicolas Grelier: Supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

Kristin Knorr: Supported by the German Science Foundation within the research training group “Facets of Complexity” (GRK 2434).

Wolfgang Mulzer: Supported in part by ERC StG 757609 and by the German Research Foundation within the collaborative DACH project *Arrangements and Drawings* as DFG Project MU 3501/3-1.

Daniel Perz: Partially supported by FWF within the collaborative DACH project *Arrangements and Drawings* as FWF project I 3340-N35.

* This research was started at the 4th DACH Workshop on Arrangements and Drawings, February 24–28, 2020, in Malchow, Germany. We thank all participants of the workshops for valuable discussions and for creating a conducive research atmosphere.



© Jonas Cleve, Nicolas Grelier, Kristin Knorr, Maarten Löffler, Wolfgang Mulzer, and Daniel Perz; licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 21; pp. 21:1–21:16

Leibniz International Proceedings in Informatics

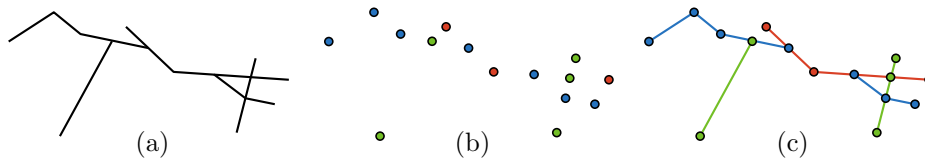


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

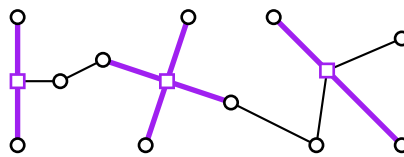
Let $P \subset \mathbb{R}^2$ be a finite planar point set, and let C be a finite set of colors. A coloring is a function $\sigma : P \rightarrow C$ that assigns a color to each point in P . For any color $c \in C$, we write $P_c = \{p \in P \mid \sigma(p) = c\}$ for the points in P that were colored with c .

In the following, we assume all pairwise distances in P are distinct. The nearest-neighbor graph for a color $c \in C$, \mathbb{N}_c , is the embedded graph with vertex set P_c and a straight-line edge between $p, q \in P_c$ if and only if p is the nearest neighbor of q among all points in P_c , or vice versa.¹ We will consider \mathbb{N}_c both as a combinatorial graph, consisting of vertices and edges, and as a subset of the plane, consisting of the points in P_c and the line segments that represent the edges. We write $\mathbb{N} = \bigcup_{c \in C} \mathbb{N}_c$ for the union of the nearest-neighbor graphs of all colors. Again, we consider \mathbb{N} both as a graph and as a set.



■ **Figure 1** (a) A drawing. (b) A 3-colored point set. (c) The nearest-neighbor graphs.

We are interested in the following problem: suppose we are given a drawing \mathcal{D} , i.e., a set of straight-line segments in the plane such that if two segments intersect, then their intersection is a point and the two segments are not parallel. Under this assumption, by considering a drawing as a set of points Q in the plane, the input segments of \mathcal{D} are the inclusion maximal segments in Q . The special points of \mathcal{D} are the endpoints of the segments in \mathcal{D} and the intersection points between pairs of segments in \mathcal{D} . We denote the set of special points by P . We require that the pairwise distances between the special points of \mathcal{D} are all distinct. Our general task is to find a set of colors C and a color assignment σ , such that the union \mathbb{N} of the nearest-neighbor graphs for P and C equals \mathcal{D} , interpreted as subsets of the plane. We call \mathbb{N} an NN-decomposition of \mathcal{D} with vertex set P , where NN stands for Nearest-Neighbor and we call $|C|$ the color-number of \mathbb{N} : see Figure 1.



■ **Figure 2** The possible violations that make a drawing non-plane.

A drawing \mathcal{D} is called plane if its segments meet only at their endpoints, i.e., no segment of \mathcal{D} contains a special point in its relative interior; see Figure 2 for an illustration where the bold edges contain an endpoint or crossing point (marked with a square) in their interior which is not allowed by the definition.

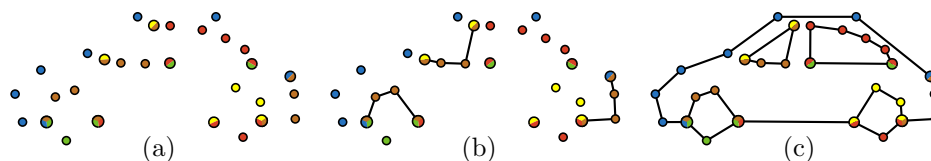
Let \mathcal{C} be a connected component in a plane drawing \mathcal{D} , and let p be a special point in \mathcal{C} . We denote by $a(p)$ the special point in $\mathcal{C} \setminus \{p\}$ that is closest to p (with distance d). Let $b(p)$ be the set of special points in \mathcal{D} whose distance to p is strictly less than d . By definition,

¹ Our notion of nearest-neighbor graph is undirected, but a directed version also exists.

$b(p) \subset \mathcal{D} \setminus \mathcal{C}$. Let \mathcal{C}_1 and \mathcal{C}_2 be two distinct connected components. We say that \mathcal{C}_1 and \mathcal{C}_2 are *conflicting* if there is a special point $p \in \mathcal{C}_1$ such that $b(p) \cap \mathcal{C}_2 \neq \emptyset$, or vice-versa. We denote by $V = \{\mathcal{C}_i\}_{1 \leq i \leq n}$ the connected components of \mathcal{D} . We define E as the set of pairs $\{\mathcal{C}_i, \mathcal{C}_j\}$ where \mathcal{C}_i and \mathcal{C}_j are conflicting. We say that the graph $G := (V, E)$ is the *conflict graph* of \mathcal{D} . We call the connected component \mathcal{C} *NN-representable* if \mathcal{C} is the nearest-neighbor graph of its special points. An abstract graph is a *conflict graph* if it is the conflict graph of some plane drawing.

Related work. The nearest-neighbor graph of a planar point set P is well understood [1, 10]. It is a subgraph of the relative neighborhood graph of P [7, 10], which in turn is a subgraph of the Delaunay triangulation. The problem of recognizing whether a given abstract graph can be realized as a nearest-neighbor graph of a planar point set is open and we conjecture it to be hard. In contrast, testing whether a given embedded graph is a (single) nearest-neighbor graph is easy, as it suffices to test if each vertex is indeed connected to its closest point.

Our problem also has applications in automated content generation for puzzle games: van Kapel introduces a version of *connect-the-dots* puzzles where the task is to connect dots based on colors rather than numbers [12]. In this puzzle, points may have multiple colors; see Figure 3. Van Kapel implemented a heuristic approach for generating such puzzles. The heuristic works well for small instances, but for larger instances, it generates too many colors to be practical [9].



■ **Figure 3** (a) Multi-colored points with 5 colors: blue, red, green, yellow, and orange: (b) the orange nearest-neighbor graph. (c) The union of all nearest-neighbor graphs. Figure taken from [9].

Our Results. First, we consider the problem of testing whether a given drawing \mathcal{D} can be decomposed into c nearest-neighbor graphs. We show that under the assumption that the drawing is plane, meaning that segments in \mathcal{D} may only meet at their endpoints, this problem is in P for $c \leq 2$, and NP-complete for $c \geq 3$. If we allow the segments of \mathcal{D} to cross the problem is already NP-complete for $c = 2$.

Inspired by our algorithms, we also introduce the new graph class of *conflict graphs* of drawings. We show that string graphs are conflict graphs and give additional, more efficient constructions in terms of size complexity for planar graphs and complete k -partite graphs. On the other hand, subdivisions of non-planar graphs are not conflict graphs.

We show a separator lemma for conflict graphs, which allows us to provide an algorithm for computing a maximum independent set in conflict graphs in subexponential time. Using it as a subroutine, we obtain an $O(\log n)$ -approximation algorithm for coloring conflict graphs that runs in subexponential time. This problem is of importance to us because we show that coloring conflict graphs is equivalent to partitioning a plane drawing into nearest-neighbor graphs.

2 Existence of NN-Decompositions on Special Points

2.1 The Plane Case

Let \mathcal{D} be a straight-line drawing. If s is a line segment with $s \subset \mathcal{D}$ such that s is not a segment of \mathcal{D} , we say that s is *covered* by \mathcal{D} . Recall that the vertex set of the NN-decomposition consists of the special points in \mathcal{D} . We investigate the question under which circumstances it is possible to find such a NN-decomposition of \mathcal{D} .

► **Lemma 2.1.** *Let \mathcal{D} be a plane drawing. Suppose there is a NN-decomposition \mathbb{N} of \mathcal{D} , and let σ be the underlying coloring of \mathbb{N} . Then, for any connected component \mathcal{C} of \mathcal{D} , the coloring σ assigns the same color to all special points in \mathcal{C} .*

Proof. Suppose \mathcal{D} has a connected component \mathcal{C} in which σ assigns two distinct colors. Then, \mathcal{C} has a segment $s = uv$ between two special points u and v such that $\sigma(u) \neq \sigma(v)$. However, the line segment uv must be covered by \mathbb{N} , and thus, there exists a segment t in \mathbb{N} that contains u, v , and another special point of \mathcal{D} (since the segments in \mathbb{N} are derived from nearest-neighbor relations between points of the same color). By our assumption that \mathcal{D} is a plane drawing, the segment t is not in \mathcal{D} , so \mathbb{N} is not an NN-decomposition of \mathcal{D} , a contradiction. ◀

► **Theorem 2.2.** *Let C be a set of colors with $|C| \leq 2$. There is a polynomial-time algorithm for the following task: given a plane drawing \mathcal{D} , is there a NN-decomposition of \mathcal{D} with color set C ?*

Proof. Let \mathcal{D} be a plane drawing. If there is a decomposition of \mathcal{D} with color set C , then, by Lemma 2.1, every connected component is colored with a single color of C , i.e., every connected component of \mathcal{D} is NN-representable. The latter necessary condition can be checked in polynomial time, as we only need to compute the nearest-neighbor graph of the special vertices in each component. If there is a connected component where this is not the case, the algorithm answers that there is no solution.

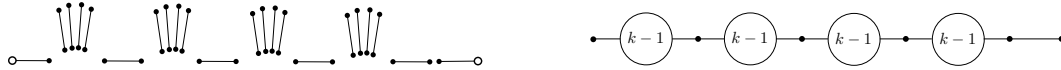
Otherwise, we construct the conflict graph G of \mathcal{D} , and we check if G can be colored with C . This takes polynomial time since $|C| \leq 2$ (for $|C| = 2$, check whether G is bipartite, for $|C| = 1$, check that G has no edges). Now, if G is C -colorable, we give all special points in a component \mathcal{C} the color assigned to the corresponding vertex in G . Since all connected components are NN-representable, this is also a NN-decomposition of \mathcal{D} with C . On the other hand, if \mathcal{D} has a NN-decomposition with color set C , then G must be C -colorable, by definition of G . ◀

► **Theorem 2.3.** *Let C be a set of colors with $|C| \geq 3$. The following task is NP-complete: given a plane drawing \mathcal{D} , is there a NN-decomposition of \mathcal{D} with color set C ?*

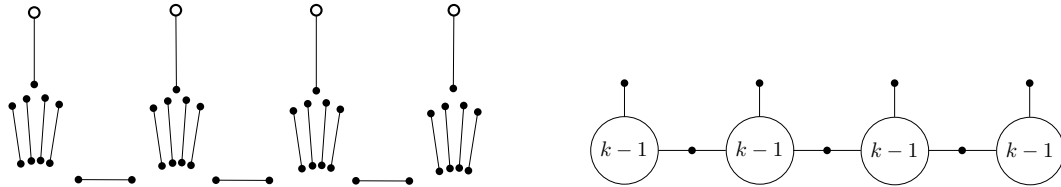
Proof. Gräf, Stumpf, and Weißenfels [6] showed how to reduce k -colorability to k -colorability of unit disk graphs. Our proof is inspired by theirs. Let $k = |C|$. We show the NP-hardness of coloring the special points of \mathcal{D} with $k \geq 3$ colors by means of a reduction from k -colorability. We make use of four types of gadgets: k -wires, k -chains, k -clones, and k -crossings. They are depicted in Figures 4–7, together with their conflict graphs. The symbol consisting of a number x in a circle denotes a clique of size x . A vertex v connected to such a symbol means that there is an edge between v and all the vertices of the clique. These conflict graphs are exactly the gadgets defined by Gräf, Stumpf, and Weißenfels. Note that each connected component in these gadgets is NN-representable. The gadgets shown are for $k = 5$.



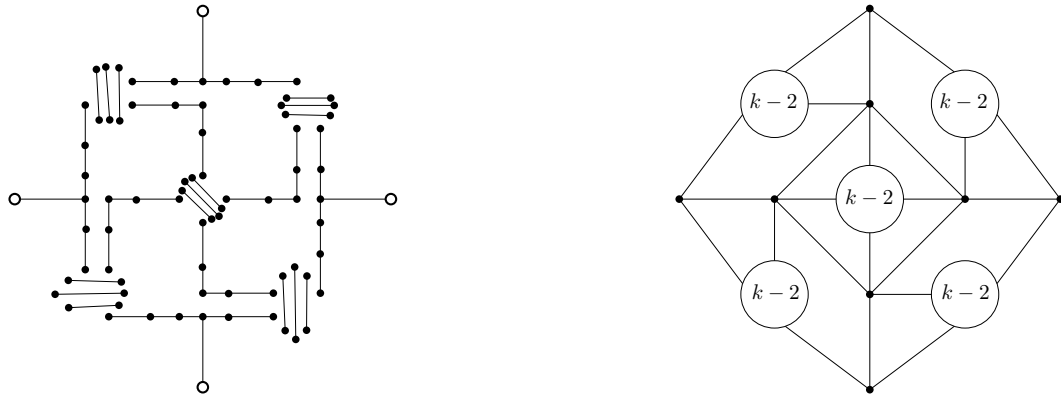
■ **Figure 4** A 5-wire of length 5 and the conflict graph of a k -wire of length 5. The symbol consisting of a number x in a circle denotes a clique of size x .



■ **Figure 5** A 5-chain of length 5 and the conflict graph of a k -chain of length 5.



■ **Figure 6** A 5-clone of length 4 and the conflict graph of a k -clone of length 4.

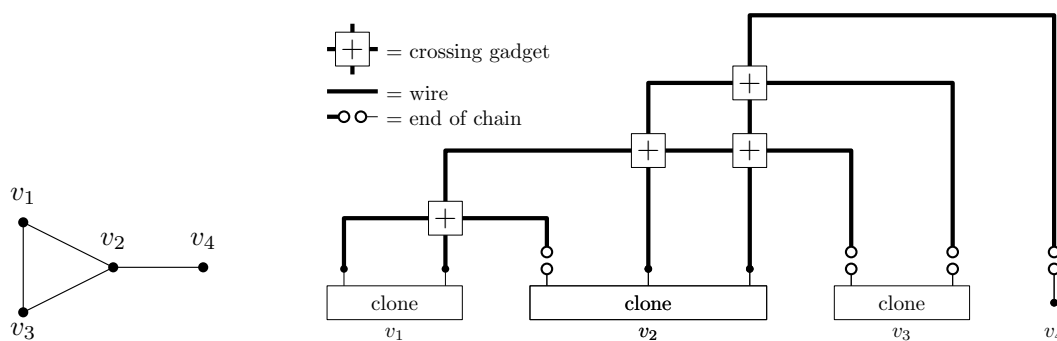


■ **Figure 7** The 5-crossing gadget and the conflict graph of the k -crossing gadget.

In Figures 4–6, there are several sets of four segments that are very close and nearly vertical. For other values of k , the gadgets are analogous, but with $k - 1$ almost vertical segments instead of four. Similarly, in Figure 7, there are five sets consisting of three close segments. For other values of k , there are five sets of $k - 2$ segments. In Figures 4 and 5, k -wires and k -chains are drawn as if they were on a line, but they may also bend with a right angle. Note that in Figures 4–7 some vertices are specially marked with larger empty circles. These vertices will be called *extreme vertices*.

In Figure 7, there seem to be points lying on a segment between two other points. Actually, these points are shifted by a sufficiently small $\varepsilon > 0$, to ensure that the resulting drawing is plane.

The main property of a k -wire is that in any coloring with k colors of its conflict graph, the extreme vertices are assigned the same color. In contrast, in a k -chain, the extreme vertices are assigned different colors. In a k -clone of length ℓ , there are ℓ extreme vertices. In any coloring with colors from C , all extreme vertices have the same color. Finally, for the k -crossing, opposite extreme vertices must have the same color; a pair of consecutive extreme vertices (e.g., top and left extreme vertices) may or may not be assigned the same color, as shown in [5].



■ **Figure 8** A graph with four vertices (left). Converting it to an NN-graph (right).

Now we follow the proof of Gräf, Stumpf, and Weißenfels. Suppose we are given a graph $G = (V, E)$. We describe a drawing \mathcal{D} whose conflict graph can be colored with color set C if and only if the vertices of G can be colored with C . Refer to Figure 8. For each vertex v of degree δ in G , we draw a k -clone of size δ . The clones are drawn so that they are arranged on a horizontal line and such that their upper points have the same y -coordinate. Then, for each edge $\{u, v\} \in E$, we draw it on the plane as two vertical segments, each incident to one k -clone, and one horizontal segment that connects the two upper points of the vertical segments. We do that such that for any pair of edges, their horizontal segments have distinct y -coordinates. Then we replace each crossing between a pair of edges by a k -crossing. Finally, let us consider one edge $\{u, v\} \in E$, and let us orient it arbitrarily, say toward v . We replace each part of the edge between two k -crossings by k -wires of sufficient length. If there are no crossings, we replace the edge by a k -chain. Otherwise, the part of the edge between u and the first k -crossing is replaced by a k -wire, and the part between the last k -crossing and v is replaced by a chain. As the points of distinct gadgets are sufficiently remote (except for pairs of gadgets that are connected on purpose), the conflict graph of this drawing is the union of the conflict graphs of the individual gadgets.

It is possible to find positions with a polynomial number of bits such that all pairwise distances are distinct but at the same time the positions are sufficiently close to the prescribed positions. This concludes the reduction.

It is straightforward to see that the problem is in NP with the certificate being a coloring of the vertices. For each point we can easily find its closest point with the same color (we compare squared distances to avoid taking square roots) and add the edges to the resulting graph. We can then compare the edges with the segments of the original drawing. ◀

2.2 The non-plane case

We show that if drawings are not required to be plane, the problem is hard for two colors.

► **Theorem 2.4.** *Let C be a set of colors with $|C| = 2$. The following problem is NP-complete: given a drawing \mathcal{D} , is there a NN-decomposition of \mathcal{D} with color set C ?*

Proof. We reduce from Not-All-Equal 3SAT (NAE-3SAT), where each clause has three variables and is satisfied if not all variables are equal. Let Φ be an NAE-3SAT formula with variable set X and clause set Y . Let G_Φ be the associated bipartite graph with vertex set $X \cup Y$, where two vertices x and y are adjacent if and only if x is a variable that appears in clause y . We draw G_Φ as follows: clauses are represented by vertical segments on the y -axis of length 3. Variables of degree δ are represented as horizontal segments on the x -axis of

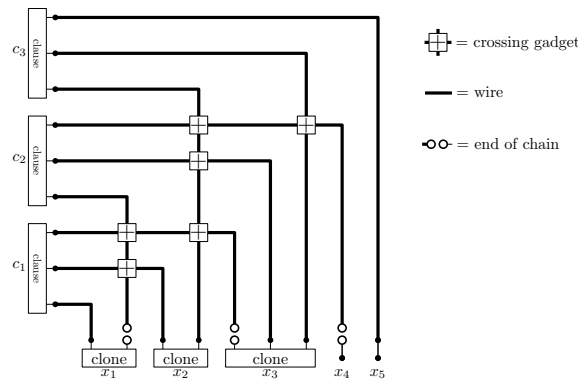


Figure 9 Structure of the conversion of the NAE-3SAT formula with clauses $c_1 = (x_1, x_2, \neg x_3)$, $c_2 = (\neg x_1, x_3, \neg x_4)$, and $c_3 = (x_2, x_3, x_5)$ into a 2-color \mathbb{N} graph.

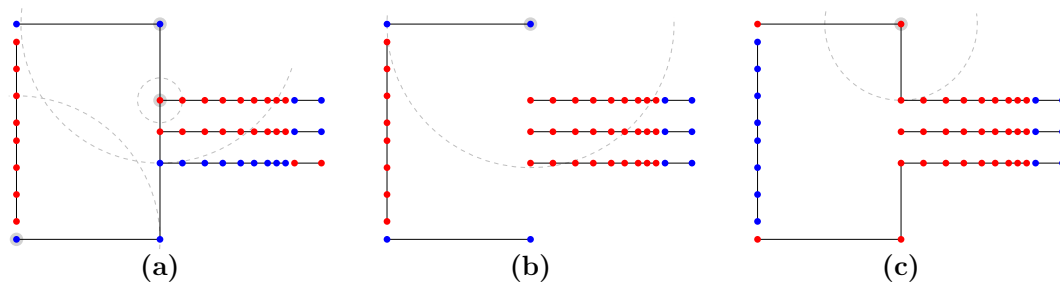


Figure 10 A clause gadget with (a) a valid assignment, (b)–(c) two invalid assignments. The dashed circles indicate distance to the nearest neighbor.

length δ . Each edge $\{x, y\}$ is drawn as the union of one vertical and one horizontal segment. The vertical segment is incident to the variable gadget for x . The horizontal segment is incident to a clause gadget for y . See Figure 9 for an example.

We use some gadgets from the proof of Theorem 2.3. We replace each variable by a 2-clone of length δ . We replace each clause by the gadget in Figure 10a (see Figure 10b-c for assignments where all literals have the same color). In Figure 10a, there seem to be points lying on a segment between two other points. Actually, these points are shifted by a sufficiently small $\varepsilon > 0$, to ensure that the resulting drawing is plane. We replace each crossing by the gadget in Figure 11. In Figure 11, some points have been colored. Note that this does not correspond to an assignment of truth values, but is supposed to provide

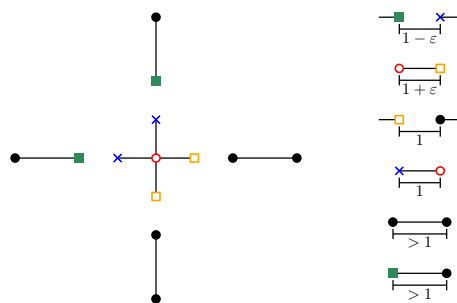
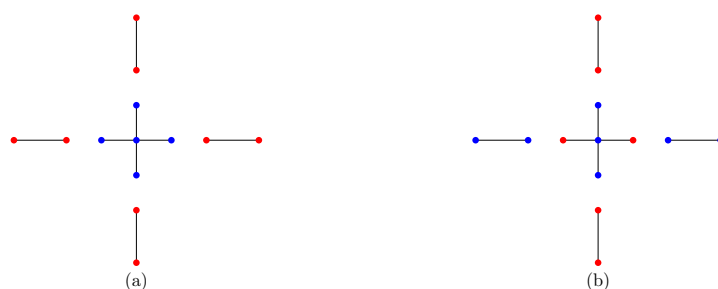


Figure 11 A non-plane crossing gadget.



■ **Figure 12** Two valid assignments of the non-plane crossing gadget. (a) All extreme segments have the same color. (b) Opposite segments have the same color.

visual information for the reader. The distance between a green point and a blue point is $1 - \varepsilon$, for a sufficiently small $\varepsilon > 0$. The distance between a blue point and the red point is 1. The distance between the red point and an orange point is $1 + \varepsilon$. The blue point on the left and the orange point on the right are finally shifted by a suitable $\eta > 0$ with $\eta \ll \varepsilon$, so that no two points are at the same distance from the red point. The points in the clause gadget that are on the vertical connected component on the left side are arranged so that this connected component is NN-representable. Finally, each part of an edge between two gadgets is replaced by a 2-wire of suitable length. We have thus obtained a drawing \mathcal{D} .

We claim that Φ is satisfiable if and only if there exists a special-point NN-decomposition of \mathcal{D} with two colors. First, notice a clause gadget has a special-point NN-decomposition if and only if two of the horizontal segments on the right side are assigned different colors. Indeed, we show in Figure 10b-c that if all literals have the same color, then the corresponding NN-graph is not the one that is required, the one shown in Figure 10a. It remains to show that if not all literals have the same color, then we obtain the correct NN-graph. By symmetry, if the top and bottom literals do not have the same color, then we are in the situation of Figure 10a. If the top and bottom literals have the same color, say red, then we keep the color of the remaining points of the clause gadget as in Figure 10a. Let us denote by p the point of the clause gadget incident to the middle literal. By assumption, p is blue. Let us denote by q the top right vertex of the clause gadget, which is also colored in blue. Therefore the closest neighbor of q which is also blue is p . Likewise, in the NN-graph, the closest neighbor colored in blue of the bottom right vertex is p . This shows that in this situation, the NN-graph is the same as in Figure 10a.

In the non-plane crossing gadget opposite segments are assigned the same color. All of them may be assigned the same color, as in Figure 12a, or consecutive segments might be assigned different colors, as in Figure 12b. Therefore, by associating the colors of \mathcal{C} with truth values, \mathcal{D} has a special-point NN-decomposition if and only if Φ is satisfiable.

That the problem is in NP can be seen the same way as in the proof for Theorem 2.3. ◀

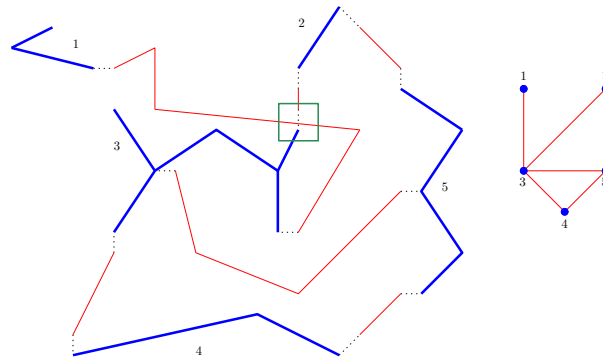
3 Conflict Graphs and Related Graph Classes

We show in Theorem 2.3 that k -coloring of conflict graphs is NP-complete, for any fixed $k \geq 3$. To put this result into context, we will show that there exist graphs that are not conflict graphs. Moreover, we will prove the inclusion of some well-known graph classes in the class of conflict graphs. The aim is to characterise the class of conflict graphs, as it gives some information about what kind of running time we can expect for the vertex coloring algorithms on conflict graphs.

Let G be a graph, and let us denote by G' the graph obtained from G by subdividing each edge of G once (i.e., for each edge $e = \{u, v\}$ in G , we add a vertex in G' whose neighbors are exactly u and v). In this section we will show that G' is a conflict graph if and only if G is planar. Sinden [11] showed the same statement for G' being a string graph, i.e., an intersection graph of continuous curves in the plane.

First, let us recall Sinden's proof for string graphs. Let G be a graph, and let G' be obtained from G as described above. Assume that G' is a string graph, and consider a representation R of G' as a string graph. Contract to a point each curve in R that corresponds to a vertex in G and extend the curves corresponding to the edges in G . In the process, one can maintain the property that the curves corresponding to the edges in G intersect only two other curves, which are now reduced to single points. Observe that the resulting drawing is a plane embedding of G . We adapt this method for conflict graphs.

Let G be a graph, and let G' be the edge-subdivision of G . Suppose that G' is a conflict graph, and let R be a representation of G' as a conflict graph. We still would like to contract each connected component of R that corresponds to a vertex in G to a single point. However, now it is not clear that we can extend the connected components corresponding to edges in G such that they only intersect two other connected components (now reduced to points). This is illustrated in Figure 13. The connected components in blue correspond to the vertices in G , those in red correspond to the edges in G . The dashed segments show a conflict between two connected components. Inside the green square, we have a connected component blocking another one. We therefore want to reroute the connected component so that there is no intersection. To show how we do it, we first need the following lemmas.

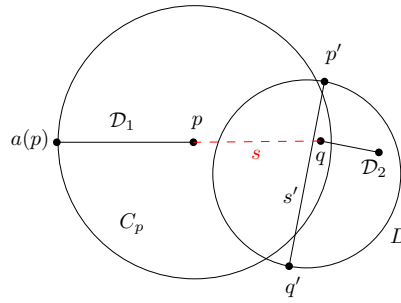


■ **Figure 13** A representation of the subdivision of G (G is shown on the right side) as a conflict graph. The connected components in blue correspond to vertices in G , the ones in red correspond to edges in G . The dashed segments show a conflict between two connected components. In the green zone, a conflict overlaps with a connected component.

Let \mathcal{D} be a plane drawing. Let $G_{\mathcal{D}}$ be its conflict graph. Let $\mathcal{D}_1, \mathcal{D}_2$ be a pair of connected components that are conflicting. Let us consider two points p and q that certify this conflict (for instance, assume without loss of generality $p \in \mathcal{D}_1, q \in \mathcal{D}_2, q \in b(p)$). We denote by s the segment with endpoints p and q .

► **Lemma 3.1.** *If a connected component intersects the line segment s , then this connected component is conflicting with \mathcal{D}_1 or \mathcal{D}_2 .*

Proof. The proof is illustrated in Figure 14. Let us denote by s' a segment in a component \mathcal{D}_3 which intersects s . Let us assume that \mathcal{D}_3 is not conflicting with \mathcal{D}_1 , and let us show that it is conflicting with \mathcal{D}_2 . Let us consider the circle C_p centered at p going through $a(p)$, the closest point to p among the ones that are in the same connected component. By assumption,



■ **Figure 14** Illustration of Lemma 3.1. The disk D contains q .

the endpoints of s' , denoted by p' and q' , are not inside C_p . In contrast, q is inside C_p . We consider the disk D with diameter s' . To show that \mathcal{D}_3 is conflicting with \mathcal{D}_2 , it is sufficient to show that q is contained in D , for then we have $q \in b(p')$ or $q \in b(q')$. We know that s' intersects C_p twice. By assumption, p is not contained in D . As two circles can intersect at most twice, D contains all of C_p on at least one side of s' . Therefore, D contains q . ◀

Let us consider another pair of conflicting components, denoted by \mathcal{D}_3 and \mathcal{D}_4 . Let $u \in \mathcal{D}_3$ and $v \in \mathcal{D}_4$ be two points that certify this conflict. Let us assume without loss of generality $v \in b(u)$. We denote by s' the segment with endpoints u and v .

► **Lemma 3.2.** *Assume that u and v are not in $b(p)$. If the segments s and s' intersect, then p or q is in $b(u)$. In particular, at least one of \mathcal{D}_3 and \mathcal{D}_4 is conflicting with \mathcal{D}_1 or \mathcal{D}_2 .*

Proof. The situation is similar to the one of Lemma 3.1. By assumption, the segment s' intersects twice the circle C_p centered at p with radius $a(p)$. Therefore the proof of Lemma 3.1 can be applied in this situation, too. ◀

We are now ready to prove the theorem. As we are considering conflict graphs, there might be obstacles when we try to follow Sinden's proof that were not there with string graphs. We use the two lemmas above to reroute these obstacles.

► **Theorem 3.3.** *Let G be a graph and let G' denote the subdivision of G . If G' is a conflict graph then G is planar.*

Proof. Let us assume we have a representation of G' as a conflict graph. We denote by V the vertices in G' corresponding to vertices in G and by E the vertices in G' corresponding to edges in G . For each vertex $v \in V$, we pick an arbitrary point p_v on the connected component that represents v in the conflict representation. We are going to reduce all connected components v to their corresponding point p_v . Let $e \in E$ be the vertex in G' corresponding to the edge $\{w, x\}$ in G . We want to extend e to a curve that contains v_x and v_w at each endpoint. We want to do that for all vertices in E , such that no two curves intersect, except maybe at endpoints. Therefore, we would obtain a plane representation of G .

Let us first consider all connected components in the representation as a conflict graph, before reducing some of them to points, and extending the rest to curves. For each pair of connected components $(\mathcal{D}_1, \mathcal{D}_2)$ that are conflicting, we find two points p and q that certify it, meaning that p is in $b(q)$ or vice versa. We now draw the segment with endpoint p and q , for each such pair of conflicting components. We denote by S the set of segments we have obtained. Let us consider a connected component \mathcal{D} corresponding to a vertex in E ,

that intersects with some segments in S . For one of these segments it intersects, say s , we name its endpoints p and q . Without loss of generality, we assume $p \in \mathcal{D}_1$ and $q \in \mathcal{D}_2$. By Lemma 3.1, \mathcal{D} is conflicting with \mathcal{D}_1 or \mathcal{D}_2 . By assumption, \mathcal{D}_1 and \mathcal{D}_2 are conflicting. Therefore one of them, say \mathcal{D}_1 , corresponds to a vertex in V , and the other corresponds to a vertex in E . By definition of G' , no two vertices in E are connected by an edge. This implies that \mathcal{D} is conflicting with \mathcal{D}_1 . This shows that when we want to extend a connected component e to a curve that contains v_x and v_w at each endpoint (see the notation above), we might be blocked by other curves, but these curves have to also contain v_x or v_w at an endpoint. This is the situation depicted in Figure 13. Here we simply have to reroute the edge going from vertex 1 to 3.

One issue that might still occur is that when trying to extend a connected component into a curve, we are not blocked by another connected component, but by the extension into a curve of a connected component. Namely, how do we do the rerouting when two segments in S intersect? Let s and s' be those two segments. By construction, s is a segment between two connected components \mathcal{D}_1 and \mathcal{D}_2 . Without loss of generality, we can assume that \mathcal{D}_1 corresponds to a vertex in V and \mathcal{D}_2 to a vertex in E . Likewise, s' is a segment between two connected components \mathcal{D}_3 and \mathcal{D}_4 , with \mathcal{D}_3 corresponding to a vertex in V and \mathcal{D}_4 to a vertex in E . Now we use Lemma 3.2, which states that one connected component from each pair are conflicting. By construction, this pair is $(\mathcal{D}_1, \mathcal{D}_4)$ or $(\mathcal{D}_2, \mathcal{D}_3)$. This shows that we are in the same situation as in the paragraph above. Thus it is possible to reduce each connected component corresponding to a vertex in V to a point, and then extend the connected components from E into curves. By doing that, we obtain a plane drawing of G . ◀

Theorem 3.3 and its analogy to the proof for string graphs in [11] may suggest that the class of conflict graph is equal to the class of string graphs. We show that, indeed, the string graphs are contained in the conflict graphs, but conjecture that the opposite is not true.

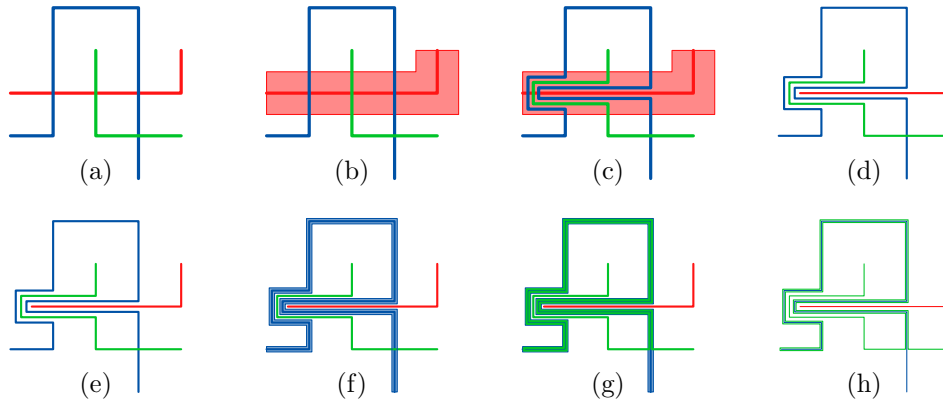
► **Theorem 3.4.** *All string graphs are conflict graphs.*

Proof. Let G be a string graph. We start by embedding G as a set of n strings (curves) in the plane. We may assume the strings are non-self-intersecting, but they could intersect other strings multiple times. For ease of exposition, we further assume that all curves are orthogonal polylines aligned to a unit grid. The proof steps are illustrated in Figure 15.

Fix a resolution $r_0 = \frac{1}{2}$ such that if we place points on each string at distance r_0 from each other, then these points will always be closer to each other than to points on other strings, except near crossings. Pick an arbitrary string s_0 . Consider the “tunnel” of width r_0 around s_0 ; by construction the other strings cross this tube in consecutive proper crossings. Consider the ordered list c_1, c_2, \dots, c_k of crossings of s_0 with other strings (note that k could be independent of n).

Set $r_1 = r_0/10k$. We increase the resolution of all strings except s_0 to r_1 . Now we reroute all other strings inside the tube of s_0 so that they keep distance r_1 from each other and from the tube boundary, but such that the crossings with s_0 are close to the start point of s_0 ; specifically, crossing c_i should be at distance $2ir_1$ from the start point of s_0 . Note that this rerouting is always possible.

Now, we shorten s_0 by deleting the first r_0 length of it. After this, s_0 does not intersect any other strings, but if we keep the resolution of s_0 at r_0 , it will have a conflict with exactly those strings that it originally intersected. Since the resolutions of the remaining strings were increased, they do not have any conflicts except near crossings with each other.



■ **Figure 15** Illustration of the proof of Theorem 3.4. (a) A string representation with 3 strings. (b) The tunnel around the red string. (c) Rerouting the blue and green strings inside the red tunnel. Note that the blue string crosses the red string twice. (d) The final set of strings after one iteration of the algorithm. The red string no longer intersects any other strings. (e-h) The second (and last) iteration of the algorithm.

We recursively apply this strategy: pick an arbitrary remaining string s_i , keep its resolution at r_i , determine its crossings and a finer resolution r_{i+1} , reroute all strings, and shorten s_i . In the end, we will have n strings s_0, \dots, s_{n-1} at increasingly fine resolutions r_0, \dots, r_{n-1} whose conflict graph is exactly the original string graph. ◀

Note that the resolution (and hence the size of the components) has a superexponential growth. Hence, this does not give us a polynomial-time reduction, and thus we cannot conclude that coloring embedded conflict graphs is NP-hard from the fact that coloring embedded string graphs is NP-hard.

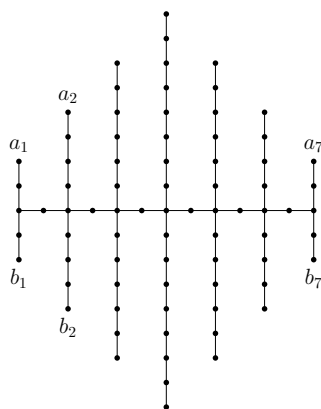
However, the following lemmas show that both planar graphs as well as complete k -partite graphs are conflict graphs and the proof gives a polynomial-time reduction (for a fixed k in the latter case). Note that Lemma 3.5 gives us an alternative proof that coloring embedded conflict graphs is NP-hard.

▶ **Lemma 3.5.** *Every planar graph with n vertices is a conflict graph of a set of components of complexity polynomial in n .*

Proof. Given a plane straight-line drawing D of a planar graph G (obtained by Fáry's theorem [2]) such that no two points are on a common vertical or horizontal line. We construct a drawing D' such that its conflict graph is G . The rough idea is to replace every vertex by the gadget in Figure 16, and replace each edge by two edges which enforce a conflict.

The gadget's drawing H (Figure 16) depends on the number of vertices n of G . We construct a horizontal line with $4n - 3$ points with small distance ℓ . To every odd vertex we add vertical lines, starting with 4 additional vertices on the left and right end, increasing by 4 vertices every step towards the center (with same distance ℓ). The topmost and bottommost vertex of these lines are denoted by a_k and b_k respectively, from left to right.

We replace every vertex v_i in D with a copy of H , called V_i . Let $a_{i,j}$ and $b_{i,j}$ denote a_j and b_j of V_i respectively. Consider an edge $\{v_i, v_j\}$ of D . Without loss of generality assume v_j is above and to the left of v_i (the other cases are symmetric). We consider all vertices in this quadrant of v_i in clockwise order and assume that v_j is the x -th such vertex. Accordingly, assume v_i is the y -th vertex in clockwise order below and to the right of v_j .



■ **Figure 16** Every vertex is replaced by this gadget. Here for $|V| = 4$.

We draw the edge $(a_{i,x}, b_{j,2n-y})$ and replace it by a path P of segments of length ℓ from $b_{j,2n-y}$ to a point p with $0 < d(p, a_{i,x}) < \ell$. The points p and $a_{i,x}$ enforce the conflict between V_i and V_j and hence the edge $\{v_i, v_j\}$ in G .

The conflict graph of D' obviously contains all the edges of G . It remains to show, that no additional edges are introduced by showing that any introduced path P only conflicts with the corresponding component V_i . Assume that the distance between every edge e and every vertex not incident to e is at least d . If ℓ is chosen small enough then the distance of P to every gadget V_k except V_i and V_j is larger than ℓ .

Assume Q is a path that conflicts with P . If Q does neither start at V_i nor at V_j , then Q cannot conflict with P since the distance between two independent edges is at least d . If Q starts at V_j we have two cases. If Q is below P , then the slope of Q is less than the slope of P . If Q is above P , then the slope of Q is greater than the slope of P . In both cases the smallest distance is between the starting points of P and Q which is at least 2ℓ by construction. Hence, there does not exist such a further conflict. ◀

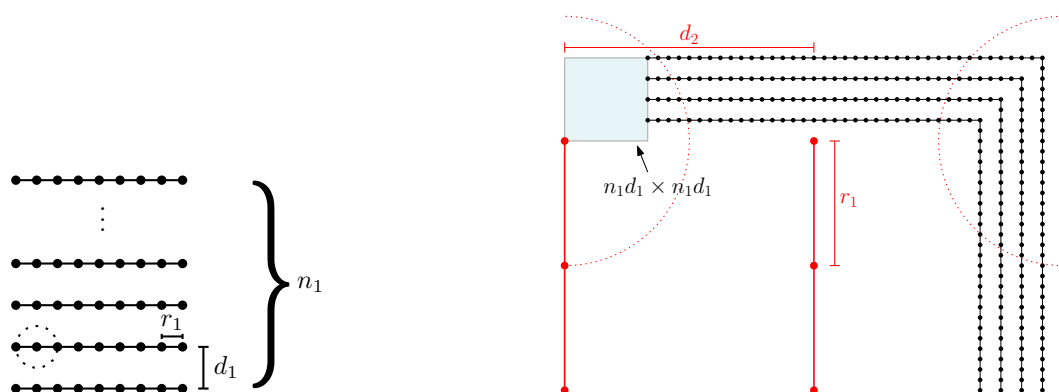
► **Lemma 3.6.** *Every complete k -partite graph with n vertices is a conflict graph of a set of components of complexity polynomial in n .*

Proof. Given a complete k -partite graph where each set with index i has n_i vertices. We will construct a graph with a drawing D such that G is the conflict graph of D , i.e., each set i will consist of n_i components which are not in conflict with each other but with all other components from the other sets.

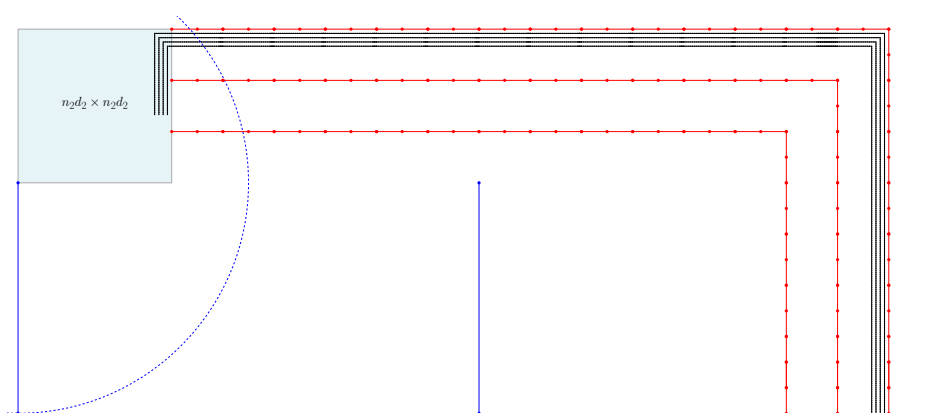
For each group i the drawing needs two properties: the components' resolution r_i and the components' minimum distance to each other d_i . The idea of the construction is that $d_i > r_i$ for all i but $r_j > d_i$ for all $j > i$ so that components in the same set do not conflict but all components in a set j conflict with all other components $i < j$ (and thus also with all $i > j$).

We start with n_1 parallel components where $r_1 = 1$ and $d_1 = 2r_1 = 2$, see Figure 17 (left). For all following sets $1 < i \leq k$ we set $r_i = \frac{3}{2}n_{i-1}d_{i-1} = 3n_{i-1}r_{i-1}$ and $d_i = 2r_i = 3n_{i-1}d_{i-1} = 6n_{i-1}r_{i-1}$. We place the components from the i th set orthogonal to the $(i-1)$ th, starting at the opposite corner of a square of size $n_{i-1}d_{i-1} \times n_{i-1}d_{i-1}$, as seen in Figure 17 (right). We finally route the $(i-1)$ th set between the last two components of the i th set with distance d_{i-1} to the last component. This requires that r_i and d_i are multiples of r_{i-1} and that d_i is a multiple of d_{i-1} which is ensured by our choices for r_i and d_i .

A depiction of the resulting drawing for $n_1 = 4$, $n_2 = 3$, and $n_3 = 3$ can be found in Figure 18. From the construction it is immediate that a component is not in conflict with a component of the same group but with all other components. Hence the initial graph G is a conflict graph of our constructed drawing D . ◀



■ **Figure 17** Left: The drawing for the first set. Right: The drawing for the third set with rerouted first set.



■ **Figure 18** A conflict graph representation of the complete 3-partite graph $K_{4,3,3}$.

4 Separators of conflict graphs and chromatic number

Let us recall the motivation for our problem. Starting from a drawing \mathcal{D} , we want to color the vertices of that drawing such that the nearest-neighbor graph on those colored vertices is the drawing \mathcal{D} . If the drawing is plane, we have shown that this problem boils down to coloring a conflict graph, where one vertex corresponds to a connected component of the drawing. We have shown in Theorem 2.3 that this problem is NP-hard, even for 3-coloring plane drawings. In this section, we show the following theorem:

► **Theorem 4.1.** *There exist an exact algorithm for maximum independent set, and an $O(\log n)$ -approximation algorithm for vertex coloring in conflict graphs with n vertices, running in $2^{n^{4/5}} \text{polylog } n$ -time.*

The exact algorithm for maximum independent set is used as a subroutine to obtain the $O(\log n)$ -approximation algorithm for coloring conflict graphs in subexponential time, where n denotes the number of vertices in the conflict graph. Indeed, coloring vertices can be seen as a covering problem, where there is a hyperedge for a set of vertices if and only if those are independent. As we have an exact algorithm for maximum independent set, we can use the greedy algorithm for covering to obtain the $O(\log n)$ approximation.

In [4], Fox and Pach present an algorithm running in $2^{n^{4/5}} \text{polylog } n$ -time for maximum independent set in string graphs. The input is an abstract graph, and it outputs a maximum independent set or a certificate that the input graph is not a string graph. As they observe,

the only property they use is the following separator lemma: Every string graph with m edges and maximum degree Δ contains a separator of order at most $c\Delta m^{1/2} \log m$ [4]. A separator in a graph $G = (V, E)$ with n vertices is a subset $V_0 \subset V$ such that there is partition of V into three sets $V = V_0 \cup V_1 \cup V_2$, with $|V_1| \leq 2n/3$ and $|V_2| \leq 2n/3$, such that there is no edge between a vertex in V_1 and a vertex in V_2 . We show that this lemma also holds for conflict graphs, which immediately implies that the algorithm by Fox and Pach also applies to our setting. The lemma was actually proven in another paper by the same authors, denoted there as Theorem 2.5 [3]. The proof uses several lemmas, but the assumption that the considered graph is a string graph appears only once. As defined in [3], the *pair-crossing number* $pcr(G)$ of a graph G is the minimum number of pairs of edges that intersect in a drawing of G . Fox and Pach showed that if G is a string graph, then $pcr(G)$ is at most the number of paths of length 2 or 3 in G , where the length of a path is the number of its edges. Following the proof of their Theorem 2.5 [3], it is sufficient for us to show the following:

► **Lemma 4.2.** *If G is a conflict graph, then $pcr(G)$ is at most the number of paths of length 2 or 3 in G .*

Proof. We use a similar notation to [3]. Let us consider a representation of a conflict graph. For each pair of conflicting connected components \mathcal{D}_i and \mathcal{D}_j , we consider two points $p \in \mathcal{D}_i$ and $q \in \mathcal{D}_j$, such that $q \in b(p)$. We now consider the drawing \mathcal{D} consisting of the union of the connected components and the line segments with endpoints p, q , for each pair of conflicting connected components. If three or more line segments intersect at the same point, we shift slightly the relative interior of one so that this is not the case anymore. It is not an issue that those are not segments anymore, we only want them to be Jordan curves. For simplicity, we keep referring to them as line segments with endpoints p, q . For each connected component \mathcal{D}_i , we consider an arbitrary point p_i on \mathcal{D}_i . Let $x = \{\mathcal{D}_i, \mathcal{D}_j\}$ denote a pair of conflicting connected components. We denote by $\alpha(x)$ a curve that starts at p_i , goes along \mathcal{D}_i until it reaches the line segment between \mathcal{D}_i and \mathcal{D}_j that we have added to the drawing, follows this line segment until it reaches \mathcal{D}_j , and finally goes along \mathcal{D}_j until it reaches p_j . Observe that this gives us a drawing of G in the plane. Suppose that two edges $\alpha(x)$ and $\alpha(y)$ in this drawing intersect. We claim that they determine a unique path of length 2 or 3 in G .

First if x and y share a connected component \mathcal{D}_i , then they determine a path of length 2 where \mathcal{D}_i is the middle vertex. Now let us assume without loss of generality that $x = \{\mathcal{D}_1, \mathcal{D}_2\}$ and $y = \{\mathcal{D}_3, \mathcal{D}_4\}$. If the curves $\alpha(x)$ and $\alpha(y)$ intersect inside one of the connected components, say \mathcal{D}_3 , then we can apply Lemma 3.1 to infer that \mathcal{D}_3 is in conflict with one of the components in x , say \mathcal{D}_1 . If $\alpha(x)$ and $\alpha(y)$ intersect outside of a connected component (i.e., the line segment between \mathcal{D}_1 and \mathcal{D}_2 intersects the line segment between \mathcal{D}_3 and \mathcal{D}_4) we can similarly apply Lemma 3.2 to show that one component from y , say \mathcal{D}_3 , is in conflict with one component in x , say \mathcal{D}_1 . In both cases the vertices $\mathcal{D}_2, \mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4$ form a path of length 3. We have shown that a pair of edges that intersect in the drawing determine a unique path of length 2 or 3 in G . ◀

Lemma 4.2 is sufficient to show that conflict graphs with m edges and maximum degree Δ contain a separator of order at most $c\Delta m^{1/2} \log m$, as shown by Fox and Pach [3]. Following their notation, the *bisection width* $b(G)$ of a graph is the least integer such that there is a partition $V = V_1 \cup V_2$ with $|V_1|, |V_2| \leq 2|V|/3$ and the number of edges between V_1 and V_2 is $b(G)$. As shown by Kolman and Matoušek [8], we have for every graph G on n vertices $b(G) \leq c \log(n)(\sqrt{pcr(G)} + \sqrt{ssqd(G)})$ where c is a constant and $ssqd(G)$ is twice the number of paths of length 1 or 2 in G . By denoting by p the number of paths of length at most 3 in G , we derive from Lemma 4.2 that if G is a conflict graph, then $b(G) = O(p^{1/2} \log n)$. A simple argument proven by Fox and Pach states that p is at most $m\Delta^2$ for a graph with m edges and maximum degree Δ [3], which concludes the proof.

5 Conclusion and open problems

In this work we studied the decomposition of a drawing into nearest-neighbor graphs. First, we studied the decision problem, whether for a given natural number $k \geq 2$ it is possible to decompose a drawing into k nearest-neighbor graphs. If we allow that segments of the drawing cross the problem is NP-complete. If we assume that the segments only meet at endpoints, it is NP-complete for $k \geq 3$ and polynomial-time solvable for $k = 2$. We provided an $O(\log n)$ -approximation algorithm running in subexponential time for coloring plane drawings with a minimum number of colors, which we showed to be equivalent to partitioning a plane drawing into a minimum number of nearest-neighbor graph. It would be interesting to find better approximation algorithms, with respect to the approximation ratio or the running time. Also, it would be interesting to study other variants of this problem; specifically, where points can have multiple colors.

We introduced so called conflict graphs and showed that not every graph is a conflict graph, but every string graph is a conflict graph. It is an open problem, whether there is a conflict graph, which is not a string graph. Further it would be interesting to know relations between other graph classes and conflict graphs.

References

- 1 D. Eppstein, M.S. Paterson, and F.F. Yao. On nearest-neighbor graphs. *Discrete Comput Geom*, 17:263–282, 1997. doi:10.1007/PL00009293.
- 2 István Fáry. On straight-line representation of planar graphs. *Acta scientiarum mathematicarum*, 11(229-233):2, 1948.
- 3 Jacob Fox and János Pach. A separator theorem for string graphs and its applications. *Combinatorics, Probability and Computing*, 19(3):371–390, 2010.
- 4 Jacob Fox and János Pach. Computing the independence number of intersection graphs. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 1161–1165. SIAM, 2011.
- 5 Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976. doi:10/dwvqpj.
- 6 Albert Gräf, Martin Stumpf, and Gerhard Weisfenfels. On coloring unit disk graphs. *Algorithmica*, 20(3):277–293, 1998.
- 7 J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. of the IEEE*, 80(9):1502–1517, 1992.
- 8 Petr Kolman and Jiří Matoušek. Crossing number, pair-crossing number, and expansion. *Journal of Combinatorial Theory, Series B*, 92(1):99–113, 2004.
- 9 Maarten Löffler, Mira Kaiser, Tim van Kapel, Gerwin Klappe, Marc van Kreveld, and Frank Staals. The connect-the-dots family of puzzles: Design and automatic generation. *ACM Transactions on Graphics*, 33(4):72, 2014. doi:10.1145/2601097.2601224.
- 10 Joseph S. B. Mitchell and Wolfgang Mulzer. Proximity algorithms. In Jacob E. Goodman, Joseph O’Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 32, pages 849–874. CRC Press, Boca Raton, 3rd edition, 2017. doi:10.1201/9781315119601.
- 11 F. W. Sinden. Topology of thin film RC circuits. *The Bell System Technical Journal*, 45(9):1639–1662, 1966. doi:10.1002/j.1538-7305.1966.tb01713.x.
- 12 Tim van Kapel. Connect the closest dot puzzles. Master’s thesis, Utrecht University, 2014. URL: <http://dspace.library.uu.nl/handle/1874/296600>.

Approximation Metatheorems for Classes with Bounded Expansion

Zdeněk Dvořák 

Computer Science Institute, Charles University, Prague, Czech Republic

Abstract

We give a number of approximation metatheorems for monotone maximization problems expressible in the first-order logic, in substantially more general settings than previously known. We obtain

- a constant-factor approximation algorithm in any class of graphs with bounded expansion,
- a QPTAS in any class with strongly sublinear separators, and
- a PTAS in any fractionally treewidth-fragile class (which includes all common classes with strongly sublinear separators).

Moreover, our tools also give an exact subexponential-time algorithm in any class with strongly sublinear separators.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases bounded expansion, approximation, meta-algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.22

Related Version *Full Version:* <https://arxiv.org/abs/2103.08698>

Funding Supported by the ERC-CZ project LL2005 (Algorithms and complexity within and beyond bounded expansion) of the Ministry of Education of Czech Republic.

1 Introduction

We are interested in approximation algorithms for problems such as the MAXIMUM INDEPENDENT SET and its variants (weighted, distance- d independent for a fixed parameter d , ...), MAXIMUM INDUCED MATCHING, MAXIMUM 3-COLORABLE INDUCED SUBGRAPH, and similar. There are many strong non-approximation results that preclude the existence of constant-factor approximation algorithms for these problems in general; for example, it is NP-hard to approximate the independence number [40] of an n -vertex graph up to the factor of $n^{1-\varepsilon}$ for every $\varepsilon > 0$. Hence, we need to consider more restricted settings.

There is a close connection between approximability and the existence of efficient algorithms parameterized by the solution size. Indeed, the existence of an EPTAS (arbitrarily precise polynomial-time approximation algorithm such that the degree of the polynomial bounding the complexity does not depend on the precision) directly implies fixed-parameter tractability, and a constant-factor approximation often forms a starting point for proving fixed-parameter tractability. A natural family of problems, namely those expressible in the first-order logic, is known to be fixed-parameter tractable in a subgraph-closed class of graphs if and (under standard complexity-theoretic assumptions) only if the class is nowhere-dense [26]. Moreover, in a slightly more restrictive setting of classes with bounded expansion, the parameterized algorithms have linear time complexity [17]. We refer the reader not familiar with the concept of bounded expansion to Section 1.1; here, let us just mention that examples of graph classes with this property are planar graphs and more generally all proper minor-closed classes, all graph classes with bounded maximum degree, and even more generally, graph classes closed under topological minors, as well as almost all Erdős-Rényi random graphs with bounded average degree.



© Zdeněk Dvořák;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 22; pp. 22:1–22:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Motivated by this connection, we explore the approximability of maximization problems expressible in the first-order logic when restricted to classes with bounded expansion. As our first main result, we show that every monotone maximization problem expressible in the first-order logic admits a constant-factor approximation algorithm in every class of graphs of bounded expansion, even in the weighted setting. We need a few definitions to formulate the precise statement.

Let I be a finite index set and let S be a set of vertices of a graph G . An I -tuple of subsets of S is a system $A_I = \{A_i : i \in I\}$, where $A_i \subseteq S$ for each i . We say that the I -tuple covers a vertex $v \in V(G)$ if $v \in \bigcup_{i \in I} A_i$. A property of I -tuples of subsets in G is a set π of I -tuples of subsets of $V(G)$, listing the I -tuples that satisfy the property π . As an example, suppose $I = \{1, 2, 3\}$ and π consists exactly of the I -tuples $\{A_1, A_2, A_3\}$ such that A_1, A_2 , and A_3 are disjoint independent sets in G ; then an induced subgraph of G is 3-colorable if and only if its vertex set is covered by some I -tuple satisfying the property π . For I -tuples A_I and A'_I , we write $A'_I \subseteq A_I$ if A'_i is a subset of A_i for each $i \in I$. Similarly, we define $A_I \cup A'_I$, $A_I \cap X$ and $A_I \setminus X$ for a set $X \subseteq V(G)$ by applying the operation in each index separately. We say that the property π is *monotone* if for all I -tuples $A'_I \subseteq A_I$, if A_I satisfies the property π , then so does A'_I .

Our goal will be to maximize the weight of an I -tuple satisfying the given property. The weight of the I -tuple is the sum of the weights of the covered vertices, where the weight of vertex can depend on its membership in the elements of the I -tuple. More precisely, for a vertex $v \in V(G)$, let $\chi_{A_I}(v) \in 2^I$ be the set of indices $i \in I$ such that $v \in A_i$. A function $w : V(G) \times 2^I \rightarrow \mathbb{Z}$ is a *weight assignment* if $w(v, \emptyset) = 0$ for each $v \in V(G)$; with a few exceptions, we will only consider assignments of non-negative weights. Let us define $w(A_I) = \sum_{v \in V(G)} w(v, \chi_{A_I}(v))$, and let $\text{MAX}(\pi, w)$ be the maximum of $w(A_I)$ over the systems A_I satisfying the property π .

Let $X_I = \{X_i : i \in I\}$ be a system of unary predicate symbols (to be interpreted as subsets of vertices of the input graph). A *first-order I -formula* is a formula φ using quantification over vertices, the predicates X_i for $i \in I$, equality, and the standard logic conjunctions. A *first-order graph I -formula* can additionally use a binary symmetric *adjacency predicate* E . A formula is a *sentence* if it has no free variables. For a graph G , an I -tuple A_I of subsets of $V(G)$, and a first-order I -sentence φ , we write $G, A_I \models \varphi$ if the sentence φ holds when the variables in its quantifiers take values from $V(G)$, the adjacency predicate is interpreted as the adjacency in G , and for $i \in I$, X_i is interpreted as the set A_i . The *property π expressed by φ* consists of all I -tuples A_I such that $G, A_I \models \varphi$. For example, the property “ X_1 is a distance-2 independent set” is expressed by the first-order graph $\{1\}$ -sentence

$$(\forall x, y) (X_1(x) \wedge X_1(y) \wedge x \neq y) \Rightarrow (\neg E(x, y) \wedge \neg(\exists z) E(x, z) \wedge E(y, z)).$$

► **Theorem 1.** *Let I be a finite index set and let φ be a first-order graph I -sentence expressing a monotone property π . For any graph class \mathcal{G} with bounded expansion, there exists a constant $c \geq 1$ and a linear-time algorithm that, given*

- a graph $G \in \mathcal{G}$ and
 - a weight assignment $w : V(G) \times 2^I \rightarrow \mathbb{Z}_0^+$,
- returns an I -tuple A_I of subsets of $V(G)$ satisfying the property π such that

$$w(A_I) \geq \frac{1}{c} \cdot \text{MAX}(\pi, w).$$

Actually, the result applies to even more general class of properties, expressible by the fragment of monadic second-order logic where we allow quantification only over the subsets of the vertices in the solution. For a finite index set I that does not contain the integers

$1, \dots, n$, a *solution-restricted MSOL I -sentence with first-order graph core* ψ is a formula of form $(Q_1 X_1 \subseteq \bigcup_{i \in I} X_i) \dots (Q_n X_n \subseteq \bigcup_{i \in I} X_i) \psi$, where Q_1, \dots, Q_n are quantifiers, X_1, \dots, X_n are unary predicate symbols (interpreted as subsets of vertices of the input graph), and ψ is a first-order graph $I \cup \{1, \dots, n\}$ -sentence. For example, the property “ $G[X]$ is a union of cycles, and the distance in G between the distinct cycles is at least three” (or more natural properties such as “ $G[X]$ is planar” or “ $G[X]$ is acyclic” that however do not use the fact that in the first-order core, we are allowed to quantify also over the vertices not in X) can be expressed in this way.

► **Theorem 2.** *Let I be a finite index set and let φ be a solution-restricted MSOL I -sentence with first-order graph core expressing a monotone property π . For any class \mathcal{G} with bounded expansion, there exists a constant $c \geq 1$ and a linear-time algorithm that, given*

■ a graph $G \in \mathcal{G}$ and

■ a weight assignment $w : V(G) \times 2^I \rightarrow \mathbb{Z}_0^+$,

returns an I -tuple A_I of subsets of $V(G)$ satisfying the property π such that

$$w(A_I) \geq \frac{1}{c} \cdot \text{MAX}(\pi, w).$$

We are also interested in the graph classes for which every monotone maximization problem expressible in the first-order logic admits a polynomial-time approximation scheme (PTAS), i.e., an arbitrarily precise polynomial-time approximation algorithm. Note that it is hard to approximate the maximum independent set size within the factor of 0.995 in graphs of maximum degree at most three [4], and thus we do not aim to obtain PTAS in all classes with bounded expansion. The class of graphs of maximum degree three has exponential expansion, motivating us to consider the classes with polynomial expansion. Dvořák and Norin [18] proved these are exactly the graph classes with strongly sublinear separators¹, and approximation questions have been intensively studied for various graph classes with this property (such as planar graphs or more generally for proper minor-closed classes); see Section 1.2 for an overview. While we were not able to obtain a PTAS for all classes with strongly sublinear separators, we were at least able to obtain a quasi-polynomial time approximation scheme (but only for properties expressed by first-order graph sentences, rather than solution-restricted MSOL sentences with first-order graph core).

► **Theorem 3.** *Let I be a finite index set and let φ be a first-order graph I -sentence expressing a monotone property π . For any class \mathcal{G} with strongly sublinear separators, there exists a polynomial p and an algorithm that, given*

■ a graph $G \in \mathcal{G}$,

■ a weight assignment $w : V(G) \times 2^I \rightarrow \mathbb{Z}_0^+$, and

■ a positive integer o ,

returns in time $\exp(p(o \cdot \log |V(G)|))$ an I -tuple A_I of subsets of $V(G)$ satisfying the property π such that

$$w(A_I) \geq \left(1 - \frac{1}{o}\right) \cdot \text{MAX}(\pi, w).$$

Interestingly, the ideas used to prove Theorem 3 also lead to exact subexponential-time algorithms for classes with strongly sublinear separators¹.

¹ For an n -vertex graph G , a set $X \subseteq V(G)$ is a *balanced separator* if each component of $G - X$ has at most $2n/3$ vertices. Let $s(G)$ denote the minimum size of a balanced separator in G , and for a class \mathcal{G} of graphs, let $s_{\mathcal{G}} : \mathbb{Z}^+ \rightarrow \mathbb{Z}_0^+$ be defined by $s_{\mathcal{G}}(n) = \max\{s(H) : H \subseteq G \in \mathcal{G}, |V(H)| \leq n\}$. The class \mathcal{G} has *strongly sublinear separators* if $s_{\mathcal{G}}(n) = O(n^{1-\beta})$ for some $\beta > 0$.

► **Theorem 4.** *Let I be a finite index set and let φ be a first-order graph I -sentence expressing a property π . Let \mathcal{G} be a class of graphs such that $s_{\mathcal{G}}(n) = O(n^{1-\beta})$ for some positive $\beta < 1$. There exists an algorithm that, given*

- a graph $G \in \mathcal{G}$ and
 - a weight assignment $w : V(G) \times 2^I \rightarrow \mathbb{Z}$ (of not necessarily non-negative weights),
- returns in time $\exp(O(|V(G)|^{1-\beta} \log^{1/2} |V(G)|))$ an I -tuple A_I of subsets of $V(G)$ satisfying the property π such that $w(A_I) = \text{MAX}(\pi, w)$.

We can obtain PTASes under a slightly stronger assumption on the considered class of graphs, efficient *fractional treewidth-fragility*. For a positive integer s and a positive real number $\delta \leq 1$, a multiset \mathcal{Z} of subsets of vertices of a graph G is an (s, δ) -generic cover of G if for every set $S \subseteq V(G)$ of size at most s , we have $S \subseteq Z$ for at least $\delta|\mathcal{Z}|$ sets $Z \in \mathcal{Z}$. The *treewidth* of the cover is the maximum of $\text{tw}(G[Z])$ over all $Z \in \mathcal{Z}$. We say that a class of graphs \mathcal{G} is *fractionally treewidth-fragile* if for some function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$, the following claim holds: for every $G \in \mathcal{G}$ and any positive integers s and o , there exists an $(s, 1 - 1/o)$ -generic cover of G of treewidth at most $f(os)$. The class is *efficiently fractionally treewidth-fragile* if such a cover can be found in time polynomial in $|V(G)|$, and in particular, \mathcal{Z} has polynomial size².

► **Theorem 5.** *Let I be a finite index set and let φ be a solution-restricted MSOL I -sentence with first-order graph core expressing a monotone property π . For any class \mathcal{G} that is efficiently fractionally treewidth-fragile, there exists a function f , a polynomial p , and an algorithm that, given*

- a graph $G \in \mathcal{G}$,
 - a weight assignment $w : V(G) \rightarrow \mathbb{Z}_0^+$, and
 - a positive integer o ,
- returns in time $f(o)p(|V(G)|)$ an I -tuple A_I of subsets of $V(G)$ satisfying the property π such that

$$w(A_I) \geq \left(1 - \frac{1}{o}\right) \cdot \text{MAX}(\pi, w).$$

Efficiently fractionally treewidth-fragile classes include many of the known graph classes with strongly sublinear separators, in particular

- all hereditary classes with sublinear separators and bounded maximum degree [13],
- all proper minor-closed classes, as an easy consequence of the result of DeVos et al. [10] (or [16] without using the Robertson-Seymour structure theorem), and
- many geometric graph classes, such as the intersection graphs of convex sets with bounded aspect ratio in a fixed Euclidean space that have bounded clique number (as can be seen using the idea of [22]).

Indeed, it is possible (and I have conjectured) that all classes with sublinear separators are fractionally treewidth-fragile.

² Note this is a somewhat non-standard formulation of fractional treewidth-fragility. In the usual definition [13, 14], one requires the existence of a system of sets whose deletion results in a graph of treewidth at most $f(o)$ and such that each vertex belongs to at most $1/o$ fraction of the sets, i.e., the complements of the sets of the system form a $(1, 1 - 1/o)$ -generic cover of treewidth at most $f(o)$. To match this with our definition, it suffices to observe that a $(1, 1 - \frac{1}{os})$ -generic cover is also $(s, 1 - 1/o)$ -generic.

Let us finish the introduction by giving two natural open questions. Our results only apply to maximization problems. More precisely, the technique we use can be applied to minimization problems (with *monotone* meaning the supersets of valid solutions are also valid solutions) as well, but the resulting algorithms have error bounded by a fraction of the total weight of all vertices, rather than a fraction of the optimal solution weight.

► **Problem 6.** *Do monotone minimization problems expressible in the first order logic admit constant factor approximation in all classes with bounded expansion? And PTASes in all efficiently fractionally treewidth-fragile graph classes?*

As a simplest example, we do not know whether there exists a PTAS for weighted vertex cover in fractionally treewidth-fragile graph classes.

Secondly, many results for classes of graphs with bounded expansion extend to nowhere-dense graph classes, up to replacement of some constants by terms of order $n^{o(1)}$. Our approach does not apply to this setting, since it is based on a quantifier elimination result specific to graph classes with bounded expansion.

► **Problem 7.** *Do monotone maximization problems expressible in the first order logic admit an $O(n^{o(1)})$ -factor approximation for n -vertex graphs from nowhere-dense classes?*

Let us remark that if a hereditary class \mathcal{G} of graphs is not nowhere-dense, then for some fixed integer r , it contains r -subdivisions of all graphs. Consequently, using the non-approximability results for the independence number [40], we conclude that for every $\varepsilon > 0$, MAXIMUM-WEIGHT DISTANCE- r INDEPENDENT SET cannot be approximated in polynomial time for n -vertex graphs from \mathcal{G} up to the factor of $O(n^{1/2-\varepsilon})$, unless $P = NP$.

1.1 Bounded expansion

The theory of bounded expansion and nowhere-density was developed chiefly by Nešetřil and Ossona de Mendez in a series of papers [32, 33, 34] to capture the notion of graph sparsity with respect to the expressive power of the first-order logic.

For a non-negative integer r , an r -shallow minor of a graph G is any graph obtained from a subgraph of G by contracting pairwise vertex-disjoint subgraphs of radius at most r . A class of graphs \mathcal{G} has *expansion bounded by a function* $f : \mathbb{Z}_0^+ \rightarrow \mathbb{Z}_0^+$ if for every $r \geq 0$, every r -shallow minor of a graph belonging to \mathcal{G} has average degree at most $f(r)$. We say that a class has *bounded expansion* if it has expansion bounded by some function, and *polynomial expansion* if it has expansion bounded by a polynomial.

Many natural graph classes have bounded expansion, thus making it possible to treat them uniformly within this framework. For example, Dvořák and Norin [18] proved that a class of graphs has polynomial expansion if and only if it has strongly sublinear separators. This includes

- planar graphs [29], and more generally all proper minor-closed classes [2];
- graphs drawn in the plane (or on a fixed surface) with a bounded number of crossings on each edge [36]; and
- many geometric graph classes, such as the intersection graphs of convex sets with bounded aspect ratio in a fixed Euclidean space that have bounded clique number, or nearest-neighbor graphs of point sets in a fixed Euclidean space [31].

Classes with bounded (but superpolynomial) expansion include

- Graph classes with bounded maximum degree, and more generally all graph classes closed under topological minors [32];
- graphs with bounded stack or queue number [36]; and,
- almost all Erdős-Rényi random graphs with linear number of edges [36].

For a more in-depth introduction to the topic, the reader is referred to the book of Nešetřil and Ossona de Mendez [35]. The classes with bounded expansion have found many applications in the design of parameterized algorithms [17, 11, 1, 28, 21]. Their applications in the context of approximation algorithms are discussed in the following section.

1.2 Related work

Distance versions of both minimum dominating set and the maximum independent set are known to admit constant-factor approximation algorithms in classes with bounded expansion [12, 15]. A constant-factor approximation algorithm for weighted and distance version of the minimum dominating set also follows from [6] combined with the bounds on the neighborhood complexity in classes with bounded expansion [38].

There are many known techniques to obtain approximation schemes for specific classes of graphs with strongly sublinear separators (polynomial expansion). We illustrate the power of these techniques on variants of the maximum INDEPENDENT SET problem: The DISTANCE- r INDEPENDENT SET problem (parameterized by a fixed positive integer r), where we require the distance between distinct vertices of the chosen set to be greater than r , and the WEIGHTED version of the problem, where the input contains an assignment of weights to vertices and we maximize the sum of weights of vertices in the set rather than the size of the set; see Table 1 for a summary.

- Let us start with the techniques that apply to all classes with strongly sublinear separators. Lipton and Tarjan [30] observed that for each $\varepsilon > 0$, one can split the input graph G by iteratively deleting sublinear separators into components of size $\text{poly}(1/\varepsilon)$, where the resulting set R of removed vertices has size at most $\varepsilon|V(G)|$. One can then solve the problem in each component separately by brute force and obtain an approximation with the additive error $\varepsilon|V(G)|$. In addition to only giving an additive approximation bound, this technique is limited to the problems for which a global solution can be obtained from the partial solutions in $G - R$; e.g., it does not apply to the DISTANCE-2 INDEPENDENT SET problem. It also does not apply in the WEIGHTED setting. Har-Peled and Quanrud [27] proved that in any hereditary class with sublinear separators, a simple local search approach (incrementally improving an initial solution by changes of bounded size) gives PTAS for a number of natural optimization problems, including the r -INDEPENDENT SET problem for any fixed $r \geq 1$ (this is not explicitly stated in their paper, but it is easy to work out the argument). On the other hand, it is not clear which problems are amenable to this approach, and it fails even for some very simple problems (e.g., finding the maximum monochromatic set in an edgeless graphs with vertices colored red and blue). The technique does not apply in the weighted setting.
- The property of *fractional treewidth-fragility*, which as noted above is satisfied by many classes with strongly sublinear separators, was developed as a way to extend Baker's technique (discussed below) to more general graph classes. While a direct application (solving the problem separately in each subgraph induced by the cover) fails for the distance versions of the problems, we have overcome this restriction in a joint work with Lahiri [20]. The approach presented in this paper can be seen as a substantial generalization in terms of the algorithmic problems to which it applies (Dvořák and Lahiri [20] only consider problems expressible in terms of distances between the solution vertices). The approach works for weighted problems. However, as a major restriction, it generally only applies to maximization problems.
- In [19], I made a rather technical attempt to improve upon the fractional treewidth-fragility, by introducing a more powerful notion of *thin systems of overlays*. All hereditary classes with sublinear separators and bounded maximum degree have this property, and

■ **Table 1** PTAS design techniques in hereditary classes with sublinear separators.

| Technique | Applies to | INDEP. SET | r -INDEP. S. | WEIGHTED I.S. | monot. FO |
|--------------------------------|---|------------|----------------|---------------|---------------|
| Iterated separators | all classes with sublinear separators | ✓ | | | |
| Local search | all classes with sublinear separators | ✓ | ✓ | | |
| Fractional treewidth-fragility | bounded max. degree, proper minor closed, ...; maybe all? | ✓ | ✓ | ✓ | ✓ (this work) |
| Thin systems of overlays | bounded max. degree, proper minor closed, ... | ✓ | ✓ | ✓ | |
| Baker's technique | proper minor closed, some geometric settings | ✓ | ✓ | ✓ | ✓ |

so do all proper minor-closed classes. Thin systems of overlays make it possible to design PTAS for the r -INDEPENDENT SET problem for any fixed $r \geq 1$, as well as for other problems defined in terms of distances between the vertices in the solution (including minimization problems such as the distance version of the minimum dominating set). However, the notion is not suitable for the problems where more complex relationships need to be considered.

- Baker [3] designed a very powerful technique for planar graphs based on finding a partition of the graph into layers (where the edges are allowed only within the layers and between consecutive layers) such that the union of a bounded number of these layers induces a subgraph of bounded treewidth. This is a substantially more restrictive condition than fractional treewidth-fragility (not even all proper minor-closed classes have this property). In a trade-off, the range of problems for which it applies is much wider. In particular, it can deal with all (maximization or minimization) problems expressible in monotone first-order logic [8], which includes all the discussed variants of the INDEPENDENT SET problem. A modified version of this technique (where the layering step is iterated and combined with removal of a bounded number of vertices) also can be used in less restricted settings [16], including for example all proper minor-closed classes (but not all classes with sublinear separators).
- The arguments based on bidimensionality [9] are rather powerful, but limited in scope to (subclasses of) the proper minor-closed classes. With regards to the PTAS design, they essentially build on the Baker's technique framework.

1.3 Proof outline

The proofs of all our results are based on three ingredients:

(1) A strong locality result for first-order properties in graphs from classes with bounded expansion, proved using a modification of the quantifier elimination procedure of [17]. To state the result, we need a few more definitions. A *simple signature* σ is a set of unary predicate and function symbols. For a finite index set I and a system X_I of unary predicate symbols disjoint from σ , a *first-order graph (I, σ) -formula* is a formula φ using all the ingredients from the definition of a first-order graph I -formula and additionally the predicates and unary functions from σ . For a graph G , a unary function $f : V(G) \rightarrow V(G)$ is *guarded by G* if for each $v \in V(G)$, either $f(v) = v$ or v is adjacent to $f(v)$ in G . A *G -interpretation* \mathcal{I} of σ assigns to each unary predicate symbol P a subset $P_{\mathcal{I}}$ of vertices of G and to each

unary function symbol f a unary function $f_{\mathcal{I}}$ guarded by G . For a positive integer s and a graph G , a function $h : V(G) \rightarrow 2^{V(G)}$ is an s -shroud if for each $v \in V(G)$, $|h(v)| \leq s$ and $v \in h(v)$. The h -center of a set $Y \subseteq V(G)$ is the set $\{v \in Y : h(v) \subseteq Y\}$.

► **Theorem 8.** *Let I be a finite index set, let φ be a first-order graph I -sentence, and let \mathcal{G} be a class of graphs with bounded expansion. There exists a constant s , a simple signature σ disjoint from all symbols appearing in φ , and a first-order graph (I, σ) -sentence φ' such that the following claim holds.*

Given a graph $G \in \mathcal{G}$, we can in time $O(|V(G)|)$ find an s -shroud h with the following property: For any $Y \subseteq V(G)$, we can in linear time find a $G[Y]$ -interpretation \mathcal{I}_Y of σ for which every I -tuple A_I of subsets of the h -center of Y satisfies

$$G, A_I \models \varphi \text{ if and only if } G[Y], \mathcal{I}_Y, A_I \models \varphi'.$$

That is, for the I -tuples of subsets of the h -center of Y , we can evaluate whether they have the property π (in the whole graph G) just by looking at the induced subgraph $G[Y]$ enhanced by \mathcal{I}_Y . Note that Theorem 8 straightforwardly extends to solution-restricted MSOL I -sentences $(Q_1 X_1 \subseteq \bigcup_{i \in I} X_i) \dots (Q_n X_n \subseteq \bigcup_{i \in I} X_i) \psi$ with first-order graph core, since if X_I is interpreted as an I -tuple A_I of subsets of the h -center of Y , then X_1, \dots, X_n also correspond to subsets of the h -center of Y .

► **Corollary 9.** *Let I be a finite index set, let φ be a solution-restricted MSOL I -sentence with first-order graph core, and let \mathcal{G} be a class of graphs with bounded expansion. There exists a constant s , a simple signature σ disjoint from all symbols appearing in φ , and a solution-restricted MSOL (I, σ) -sentence φ' with first-order graph core such that the following claim holds.*

Given a graph $G \in \mathcal{G}$, we can in linear time find an s -shroud h with the following property: For any $Y \subseteq V(G)$, we can in linear time find a $G[Y]$ -interpretation \mathcal{I}_Y of σ for which every I -tuple A_I of subsets of the h -center of Y satisfies

$$G, A_I \models \varphi \text{ if and only if } G[Y], \mathcal{I}_Y, A_I \models \varphi'.$$

(2) The existence of sufficiently generic covers. For efficiently fractionally treewidth-fragile classes, we have them by definition. For classes with bounded expansion, we use covers obtained from *low-treedepth colorings*. A *rooted forest* F is an acyclic graph with a specified *root* vertex in each component. The *depth* of F is the number of vertices on the longest path from a root to a leaf. If the path in F from a root to a vertex v contains a vertex u , we say that u is an *ancestor* of v and v is a *descendant* of u . The *closure* of F is the graph with the vertex set $V(F)$ where each vertex is adjacent exactly to its ancestors and descendants in F . The *treedepth* of a graph H is the minimum d such that H is a subgraph of the closure of a rooted forest of depth d . A graph of treedepth d is known to have treewidth (in fact, even pathwidth) smaller than d [35]. For a positive integer s , a *treedepth- s coloring* of a graph G is a coloring such that the union of every s color classes induces a subgraph of treewidth at most s . Nešetřil and Ossona de Mendez [33] proved the following claim.

► **Theorem 10** (Nešetřil and Ossona de Mendez [33]). *For every class \mathcal{G} of graphs with bounded expansion and every positive integer s , there exists an integer a and a linear-time algorithm that given a graph $G \in \mathcal{G}$ returns a treedepth- s coloring of G using at most a colors.*

By considering the cover consisting of all $c = \binom{a}{s}$ unions of s -tuples of color classes, we obtain the following claim.

► **Corollary 11.** *For every class \mathcal{G} of graphs with bounded expansion and every positive integer s , there exists a positive integer c and a linear-time algorithm that given a graph $G \in \mathcal{G}$ returns an $(s, 1/c)$ -generic cover of size c and treedepth at most s .*

For classes with strongly sublinear separators, in [14] I proved they are “almost” fractionally treewidth-fragile, in the following sense (again, with a somewhat different notation, see the footnote at the definition of fractional treewidth-fragility).

► **Theorem 12** (Dvořák [14]). *For every class \mathcal{G} with strongly sublinear separators, there exists a polynomial $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ and a polynomial-time algorithm that, for every $G \in \mathcal{G}$ and positive integers s and o , returns an $(s, 1 - 1/o)$ -generic cover of G of treewidth at most $f(os \log |V(G)|)$. Moreover, the algorithm also returns the corresponding tree decomposition for each element of the cover.*

(3) The means to solve the problem on graphs of bounded treewidth. For Theorems 2 and 5, we use the well-known result of Courcelle [7], in the following optimization version (note that we do not need to be given a tree decomposition, as for graphs of bounded treewidth, we can find an optimal tree decomposition in linear time [5]).

► **Theorem 13.** *Let I be a finite index set and let φ be a MSOL graph formula with free variables X_I , expressing a property π . For any positive integer t , there exists a linear-time algorithm that, given*

- a graph G of treewidth at most t ,
- a set $X \subseteq V(G)$, and
- a weight assignment $w : V(G) \times 2^I \rightarrow \mathbb{Z}$,

returns an I -tuple A_I of subsets of X satisfying the property π such that $w(A_I)$ is maximum among all such I -tuples, or decides no such I -tuple of subsets of X exists.

This is not sufficient for the proof of Theorem 3, where we work with covers of polylogarithmic treewidth, and thus we need a better control over the dependence of the time complexity on the treewidth. We use the following result proved using the locality property underlying Theorem 8; we believe this result to be of independent interest.

► **Theorem 14.** *Let I be a finite index set and let φ be a first-order graph I -sentence expressing a property π . For any class \mathcal{G} with bounded expansion, there exists a constant $c > 0$ and an algorithm that, given*

- a graph $G \in \mathcal{G}$,
- a tree decomposition τ of G with at most $|V(G)|$ nodes,
- a set $X \subseteq V(G)$, and
- a weight assignment $w : V(G) \times 2^I \rightarrow \mathbb{Z}_0$,

returns in time $O(\exp(ct)|V(G)|)$, where t is the width of the decomposition τ , an I -tuple A_I of subsets of X satisfying the property π such that $w(A_I)$ is maximum among all such I -tuples, or decides no such I -tuple of subsets of X exists.

Let us remark that the reason we are not able to generalize Theorem 3 to solution-restricted MSOL sentences with first-order graph core is that we cannot prove the analogue of Theorem 14 in that setting.

We are now ready to prove our results.

Proof of Theorems 2, 3 and 5. Note that \mathcal{G} has bounded expansion:

- In the situation of Theorem 2, this is an assumption.
- In the situation of Theorem 3, this is the case since classes with strongly sublinear separators have polynomial expansion [18].

22:10 Approximation Metatheorems for Classes with Bounded Expansion

- In the situation of Theorem 5, this is the case since every fractionally treewidth-fragile class has bounded expansion [13].

Let s , σ , and φ' be obtained by applying Theorem 8 or Corollary 9 to φ and \mathcal{G} . Now, for the input graph G and the weight assignment w (and the precision o in the case of Theorems 3 and 5), we apply the following algorithm:

- Let h be the s -shroud obtained using Theorem 8 or Corollary 9.
- Let \mathcal{Z} be an (s, δ) -generic cover of treewidth at most t , where
 - in the situation of Theorem 2, \mathcal{Z} is obtained using Corollary 11, $\delta = 1/c$, and $t = s$;
 - in the situation of Theorem 3, \mathcal{Z} is obtained using Theorem 12 for the given o and s , $\delta = 1 - 1/o$, and $t = \text{poly}(os \log |V(G)|)$; and
 - in the situation of Theorem 5, \mathcal{Z} is obtained using the definition of efficient fractional treewidth-fragility for the given o and s , $\delta = 1 - 1/o$, and $t = f(os)$.
- For each $Y \in \mathcal{Z}$:
 - Let \mathcal{I}_Y be the $G[Y]$ -interpretation of σ obtained using Theorem 8 or Corollary 9.
 - Let Y' be the h -center of Y .
 - Using Theorem 13 or 14 in the bounded-treewidth graph $G[Y]$, find an I -tuple A_I^Y of subsets of Y' satisfying $G[Y], \mathcal{I}_Y, A_I^Y \models \varphi'$ such that $w(A_I^Y)$ is maximum possible.
- Return the I -tuple A_I^Y such that $w(A_I^Y)$ is maximum over all $Y \in \mathcal{Z}$.

Note that Theorem 8 or Corollary 9 implies that $G, A_I^Y \models \varphi$, and thus it suffices to bound the approximation ratio of the algorithm.

Let A_I be an I -tuple of subsets of vertices of G satisfying the property π such that $w(A_I)$ is maximum. Choose $Y \in \mathcal{Z}$ uniformly at random. Since h is an s -shroud and \mathcal{Z} is an (s, δ) -cover, for each $v \in V(G)$, the probability that $h(v) \subseteq Y$ is at least δ . Hence, letting $A_I' = A_I \cap Y'$ (where Y' is the h -center of Y), the expected value of $w(A_I')$ is at least $\delta \cdot w(A_I)$. Since π is monotone, we have $G, A_I' \models \varphi$, and by Theorem 8, $G, \mathcal{I}_Y, A_I' \models \varphi'$. This implies $w(A_I^Y) \geq w(A_I')$, and thus the expected value of $w(A_I^Y)$ is also at least $\delta \cdot w(A_I)$. Since we return the maximum over all elements of \mathcal{Z} , this implies the weight of the returned set is at least $\delta \cdot w(A_I)$. ◀

Proof of Theorem 4. Note that since \mathcal{G} has strongly sublinear separators, it has bounded (in fact, polynomial) expansion [18].

Using the algorithm of [23], we can for any n -vertex subgraph of G in polynomial time find a balanced separator of size $O(n^{1-\beta} \log^{1/2} n)$. Using this algorithm, let us construct a tree decomposition τ of G as follows:

- Find a balanced separator S in G .
- Recursively find a tree decomposition τ_C of each component C of $G - S$.
- Add a new root vertex adjacent to the root of τ_C for each component C of $G - S$, and add S to all bags (including the bag of the new root vertex).

This tree decomposition has width $t = O(|V(G)|^{1-\beta} \log^{1/2} |V(G)|)$. We then apply the algorithm from Theorem 14 for this tree decomposition and $X = V(G)$. ◀

Theorem 8 is a consequence of a quantifier-elimination result whose proof is inspired by the approach of Dvořák, Král', and Thomas [17] (but likely could also be proved using the alternative approaches to quantifier elimination in bounded expansion classes, such as [24, 25, 37, 39]); we state the result and derive Theorem 8 in Section 2. Theorem 14 follows by a standard dynamic programming approach, with Theorem 8 used to bound the number of states; we give the proof in Section 3.

2 The quantifier elimination result and its applications

In [13], it has been shown that any first-order formula on a graph from a class with bounded expansion can be transformed into an equivalent quantifier-free first-order formula on a graph from a (different) class with bounded expansion, by introducing unary functions guarded by the resulting graph and new unary predicates, both computable in linear time. Essentially the same procedure can be applied to first-order formulas describing set properties; however, for our application, we need to be more explicit in terms of how the newly introduced functions and predicates are defined and in particular, how they depend on the set system whose weight we are maximizing.

To this end, in addition to unary functions and predicates, we introduce *counters*. Semantically, a counter γ assigns a non-negative integer $\gamma(v)$ to each vertex v . For each counter symbol γ , our formulas can use expressions of form $\gamma(x) \geq m$, where x is a term and m is a positive integer, with the natural interpretation. The symbols for counters are linearly ordered, and we say that a formula is γ -dominated if all counter symbols appearing in the formula are strictly smaller than γ . For a variable x , we say that a formula θ is x -local if θ is quantifier-free, does not use unary functions, and x is the only variable appearing in θ .

The counters are used to keep track of the numbers of vertices that satisfy a prescribed property. Formally, the values of each counter γ are determined by an associated *trigger* (f, θ) , where f is a unary function symbol and θ is an x -local γ -dominated first-order formula. For each vertex v , the value $\gamma(v)$ is equal to the number of vertices $u \in V(G) \setminus \{v\}$ such that $f(u) = v$ and $\theta(u)$ holds.

A *global formula* is a formula that can additionally use elementary formulas of form $\#\theta \geq m$, where θ is an x -local formula and m is a positive integer; this elementary formula is true if there are at least m vertices $v \in V(G)$ such that $\theta(v)$ holds.

Let I be a finite index set. A *counter I -signature* σ is a set of unary predicate and function symbols and linearly ordered counter symbols, together with the triggers associated with these counter symbols, where the triggers are allowed to refer to the unary predicates from X_I and σ . For a graph G , a *G -interpretation* of σ is a G -interpretation of the simple signature consisting of the unary predicate and function symbols from σ . We can now state the quantifier elimination result.

► **Theorem 15.** *Let I be a finite index set, let φ be a first-order graph I -sentence, and let \mathcal{G} be a class of graphs with bounded expansion. There exists a counter I -signature σ and a global quantifier-free first-order (I, σ) -sentence φ' such that the following claim holds. Given a graph $G \in \mathcal{G}$, we can in linear time compute a G -interpretation \mathcal{I} of σ such that*

$$G, A_I \models \varphi \text{ if and only if } G, \mathcal{I}, A_I \models \varphi'$$

for every I -tuple A_I of subsets of $V(G)$.

As we mentioned before, Theorem 8 can be proved by a straightforward modification of known arguments [17, 24, 25, 37, 39] and we omit the proof in this extended abstract; it can be found in the full version of the paper at <https://arxiv.org/abs/2103.08698>.

Let us remark that since φ' is a quantifier-free sentence, it is a Boolean combination of formulas of form $\#\theta \geq m$, where θ is an x -local formula, and in particular φ' does not refer to any function symbols.

Importantly, the interpretation \mathcal{I} of the predicate and function symbols from σ is independent on A_I ; the choice of A_I only affects the values of counters. Let $\ell(\sigma)$ denote the number of counter symbols in σ , and for $i = 1, \dots, \ell(\sigma)$, let $f_{i, \mathcal{I}}$ be the interpretation

22:12 Approximation Metatheorems for Classes with Bounded Expansion

of the unary function from the trigger of the i -th counter symbol in σ , in their fixed linear ordering. For $v \in V(G)$, let us define $h_{0,\mathcal{I}}(v) = \{v\}$ and for $i = 1, \dots, \ell(\sigma)$, let $h_{i,\mathcal{I}}(v) = h_{i-1,\mathcal{I}}(v) \cup \{f_{i,\mathcal{I}}(u) : u \in h_{i-1,\mathcal{I}}(v)\}$. We let $h_{\mathcal{I}} = h_{\ell(\sigma),\mathcal{I}}$; note that $h_{\mathcal{I}}$ is a $2^{\ell(\sigma)}$ -shroud.

For a counter γ of a counter I -signature σ , a G -interpretation \mathcal{I} , and an I -tuple A_I of subsets of $V(G)$, let $\gamma(\mathcal{I}, A_I, v)$ denote the value of the counter γ when the symbols of σ are interpreted according to \mathcal{I} and the predicates X_I are interpreted as A_I . The applications of Theorem 15 use the simple fact that the membership of a vertex v in the sets of an I -tuple A_I only affects the values of the counters in $h_{\mathcal{I}}(v)$.

► **Lemma 16.** *Let I be a finite index set, let σ be a counter I -signature, let G be a graph and let \mathcal{I} be a G -interpretation of σ . For any $X \subseteq V(G)$ and I -tuples A_I and A'_I of subsets of vertices of G , if $A_I \setminus X = A'_I \setminus X$, then $\gamma(\mathcal{I}, A_I, v) = \gamma(\mathcal{I}, A'_I, v)$ for every $v \in V(G) \setminus h_{\mathcal{I}}(X)$.*

Proof. For $i = 1, \dots, \ell(\sigma)$, let (θ_i, f_i) be the trigger of the i -th counter symbol γ_i in σ , in their fixed linear ordering, and let $f_{i,\mathcal{I}}$ be the interpretation of f_i in \mathcal{I} . By induction on i , we show that $\gamma_i(\mathcal{I}, A_I, v) = \gamma_i(\mathcal{I}, A'_I, v)$ for each $v \in V(G) \setminus h_{i,\mathcal{I}}(X)$. Recall that $\gamma_i(\mathcal{I}, A_I, v)$ is the number of vertices $u \in V(G) \setminus \{v\}$ such that $f_{i,\mathcal{I}}(u) = v$ and $G, \mathcal{I}, A_I \models \theta_i(u)$. Note that $u \in V(G) \setminus h_{i-1,\mathcal{I}}(X) \subseteq V(G) \setminus h_{j,\mathcal{I}}(X) \subseteq V(G) \setminus X$ for each $j \leq i-1$, and thus $\gamma_j(\mathcal{I}, A_I, u) = \gamma_j(\mathcal{I}, A'_I, v)$ for each such j by the induction hypothesis. Since θ_i is x -local, the value of $\theta_i(u)$ only depends on these counters, \mathcal{I} , and the interpretation of X_I on $u \notin X$, and thus $G, \mathcal{I}, A_I \models \theta_i(u)$ if and only if $G, \mathcal{I}, A'_I \models \theta_i(u)$. Consequently, $\gamma_i(\mathcal{I}, A_I, v) = \gamma_i(\mathcal{I}, A'_I, v)$, as required. ◀

This is useful in combination with the fact that vertices with fixed values of the counters can be deleted. Let I be a finite index set, let σ be a counter I -signature, and let φ be a global quantifier-free first-order (I, σ) -sentence. For a graph G and $Y \subseteq V(G)$, a (G, Y, σ, φ) -census is a function n that, letting M be the largest integer appearing in the formula φ and the triggers of σ , assigns an element of $\{0, \dots, M\}$ to

- each x -local formula θ appearing in an elementary formula $\#\theta \geq m$ in φ , and
- each pair (γ, v) , where γ is a counter symbol of σ and $v \in Y$ is a vertex with at least one neighbor in $V(G) \setminus Y$.

Given a G -interpretation \mathcal{I} of σ and an I -tuple A_I of subsets of vertices of G , we say that n is a $(G, Y, \mathcal{I}, \sigma, \varphi)$ -shadow of A_I if

- for each x -local formula θ appearing in an elementary formula $\#\theta \geq m$ in φ , $n(\theta)$ is the minimum of M and the number of vertices $u \in V(G) \setminus Y$ such that $G, \mathcal{I}, A_I \models \theta(u)$; and,
- for each counter γ with trigger (θ, f) and each vertex $v \in Y$ with at least one neighbor in $V(G) \setminus Y$, $n(\gamma, v)$ is the minimum of M and the number of vertices $u \in V(G) \setminus Y$ such that $f_{\mathcal{I}}(u) = v$ and $G, \mathcal{I}, A_I \models \theta(u)$.

► **Lemma 17.** *Let I be a finite index set, let σ be a counter I -signature, and let φ be a global quantifier-free first-order (I, σ) -sentence. There exists a signature σ' obtained from σ by adding unary predicate symbols and changing the triggers on the counter symbols, and a global quantifier-free first-order (I, σ') -sentence φ' such that the following claim holds. For any graph G , a G -interpretation \mathcal{I} of σ , a set $Y \subseteq V(G)$ and a (G, Y, σ, φ) -census n , we can in linear time find a $G[Y]$ -interpretation \mathcal{I}' of σ' such that*

$$G, \mathcal{I}, A_I \models \varphi \text{ if and only if } G[Y], \mathcal{I}', A_I \models \varphi'$$

for every I -tuple A_I of subsets of Y with $(G, Y, \mathcal{I}, \sigma, \varphi)$ -shadow n .

Proof. Let M be the largest integer appearing in the formula φ or the triggers of σ . For each counter symbol γ and each positive integer $m \leq M$, we add to σ' a unary predicate $P_{\gamma,m}$, interpreted in \mathcal{I}' as the set of vertices $v \in Y$ such that $n(\gamma, v) \geq m$. Each function symbol g is interpreted in \mathcal{I}' by setting $g_{\mathcal{I}'}(v) = g_{\mathcal{I}}(v)$ if $g_{\mathcal{I}}(v) \in Y$ and $g_{\mathcal{I}}(v) = v$ otherwise. The predicate symbols are interpreted in \mathcal{I}' as the restrictions of their interpretations in \mathcal{I} to Y .

For a counter symbol γ with trigger (θ, f) , we set the trigger of γ in σ' to be (θ', f) , where θ' is obtained from θ by replacing each formula of form $\gamma'(x) \geq m$ by the formula

$$(\gamma'(x) \geq m) \vee (P_{\gamma',1}(x) \wedge \gamma'(x) \geq m-1) \vee \dots \vee P_{\gamma',m}(x).$$

This ensures that for each $v \in Y$ and each I -tuple A_I of subsets of Y with $(G, Y, \mathcal{I}, \sigma, \varphi)$ -shadow n , we have

$$\min(M, \gamma(\mathcal{I}, A_I, v)) = \min(M, \gamma(\mathcal{I}', A_I, v) + n(\gamma, v)),$$

where $n(\gamma, v) = 0$ if all neighbors of v belong to Y . For each subformula of φ of form $\#\theta \geq m$, let θ' be obtained by performing the same replacements. The formula φ' is obtained from φ by replacing each such subformula by the formula $\#\theta' \geq \max(0, m - n(\theta))$. The fact that $G, \mathcal{I}, A_I \models \varphi$ if and only if $G[Y], \mathcal{I}', A_I \models \varphi'$ holds for every I -tuple A_I of subsets of Y with $(G, Y, \mathcal{I}, \sigma, \varphi)$ -shadow n is clear from the construction. \blacktriangleleft

We also need the fact that counters can be eliminated at the expense of re-introducing quantifiers.

► Lemma 18. *Let I be a finite index set, let σ be a counter I -signature, and let σ' be the simple signature consisting of the predicate and function symbols from σ . For any counter symbol $\gamma \in \sigma$ and a positive integer m , there exists a first-order (I, σ') -formula $\psi_{\gamma,m}$ with one free variable x such that the following claim holds: Let G be a graph and let \mathcal{I} be a G -interpretation of σ . For any I -tuple A_I of subsets of vertices of G , we have $\gamma(\mathcal{I}, A_I, v) \geq m$ if and only if $G, \mathcal{I}, A_I \models \psi_{\gamma,m}(v)$.*

Proof. We prove the claim by induction along the linear ordering of the counter symbols in σ , and thus we can assume that the claim holds for all counter symbols appearing in the trigger (θ, f) of γ . Let θ' be the (I, σ') -formula obtained from θ by replacing each formula of form $\gamma'(x) \geq m'$ by the formula $\psi_{\gamma',m'}(x)$. We let $\psi_{\gamma,m}$ be the formula

$$(\exists x_1) \dots (\exists x_m) \left(\bigwedge_{i < j} x_i \neq x_j \right) \wedge \bigwedge_{i=1}^m (x_i \neq x \wedge f(x_i) = x \wedge \theta'(x_i)). \quad \blacktriangleleft$$

We now straightforwardly compose the results.

Proof of Theorem 8. Let σ_1 be the counter I -signature and φ_1 the global quantifier-free first-order (I, σ_1) -sentence obtained using Theorem 15. Let $s = 2^{\ell(\sigma_1)}$. Let σ_2 and φ_2 be the counter I -signature and the global quantifier-free first-order (I, σ_2) -sentence obtained using Lemma 17 for σ_1 and φ_1 . We let σ be the simple signature consisting of the predicate and function symbols from σ_2 , and for each counter symbol γ of σ_2 and each positive integer m , we let $\psi_{\gamma,m}$ be the formula constructed in Lemma 18. We let φ' be the formula obtained from φ_2 by replacing every subformula of form $\gamma(x) \geq m$ by the formula $\psi_{\gamma,m}(x)$, and every subformula of form $\#\theta \geq m$ by the formula

$$(\exists x_1) \dots (\exists x_m) \left(\bigwedge_{i < j} x_i \neq x_j \right) \wedge \bigwedge_{i=1}^m \theta(x_i).$$

22:14 Approximation Metatheorems for Classes with Bounded Expansion

Now, given a graph G , we first use the algorithm from Theorem 15 to compute a G -interpretation \mathcal{I}_1 of σ_1 such that

$$G, A_I \models \varphi \text{ if and only if } G, \mathcal{I}_1, A_I \models \varphi_1$$

for every I -tuple A_I of subsets of $V(G)$. We let h be the s -shroud $h_{\mathcal{I}_1}$ and let X be the h -center of Y . Let n be the $(G, Y, \mathcal{I}_1, \sigma_1, \varphi_1)$ -shadow of the I -tuple of empty sets. Note that by Lemma 16, every I -tuple A_I of subsets of X has $(G, Y, \mathcal{I}_1, \sigma_1, \varphi_1)$ -shadow n . Using the algorithm from Lemma 17, we compute a $G[Y]$ -interpretation \mathcal{I}_2 of σ_2 such that

$$G, \mathcal{I}_1, A_I \models \varphi_1 \text{ if and only if } G[Y], \mathcal{I}_2, A_I \models \varphi_2$$

for every I -tuple A_I of subsets of X . By construction, φ' is a first-order (I, σ) -sentence such that

$$G[Y], \mathcal{I}_2, A_I \models \varphi_2 \text{ if and only if } G[Y], \mathcal{I}_2, A_I \models \varphi'$$

for every I -tuple A_I of subsets of Y . Therefore,

$$G, A_I \models \varphi \text{ if and only if } G[Y], \mathcal{I}_2, A_I \models \varphi'$$

for every I -tuple A_I of subsets of the h -center X of Y , and we can set $\mathcal{I}_Y = \mathcal{I}_2$. ◀

3 Deciding first-order properties in time exponential in treewidth

In this section, we give the algorithm from the statement of Theorem 14. Let I be a finite index set, let σ be a counter I -signature, let φ be a global quantifier-free first-order (I, σ) -sentence, let G be a graph, and let \mathcal{I} be a G -interpretation of σ . Suppose $G = L \cup R$ for some subgraphs L and R of G . In this context, for I -tuples A_I and A'_I of subsets of vertices of L , we write $A_I \equiv_{(L,R)} A'_I$ if for every I -tuple B_I of subsets of $V(R) \setminus V(L)$, we have

$$G, \mathcal{I}, A_I \cup B_I \models \varphi \text{ if and only if } G, \mathcal{I}, A'_I \cup B_I \models \varphi.$$

The algorithm is based on the following key fact.

► **Lemma 19.** *Let I be a finite index set, let σ be a counter I -signature, let φ be a global quantifier-free first-order (I, σ) -sentence, let $G = L \cup R$ be a graph and let \mathcal{I} be a G -interpretation of σ . Let $h_{\mathcal{I}}$ be the corresponding $2^{\ell(\sigma)}$ -shroud and let $S = h_{\mathcal{I}}(V(L \cap R)) \cap V(L)$. If A_I and A'_I are I -tuples of subsets of $V(L)$ such that*

- $A_I \cap S = A'_I \cap S$ and
- A_I and A'_I have the same $(G, V(L) \setminus S, \mathcal{I}, \sigma, \varphi)$ -shadow n ,

then $A_I \equiv_{(L,R)} A'_I$.

Proof. By the definition of $h_{\mathcal{I}}$, we have $S = h_{\mathcal{I}}(V(R)) \cap V(L)$. Consider any I -tuple B_I of subsets of $V(R) \setminus V(L)$. By Lemma 16 applied with $X = V(R) \setminus V(L)$, we have $\gamma(\mathcal{I}, A_I \cup B_I, v) = \gamma(\mathcal{I}, A_I, v)$ and $\gamma(\mathcal{I}, A'_I \cup B_I, v) = \gamma(\mathcal{I}, A'_I, v)$ for each $v \in V(L) \setminus S$. Consequently, the $(G, V(L) \setminus S, \mathcal{I}, \sigma, \varphi)$ -shadow of both $A_I \cup B_I$ and $A'_I \cup B_I$ is equal to n . Let σ' , φ' , and \mathcal{I}' be obtained using Lemma 17 for the census n and $Y = V(R) \cup S$. By the assumptions, we have $(A_I \cup B_I) \cap (V(R) \cup S) = (A'_I \cup B_I) \cap (V(R) \cup S)$, and thus

$$\begin{aligned} G, \mathcal{I}, A_I \cup B_I \models \varphi & \text{ if and only if} \\ G[V(R) \cup S], \mathcal{I}', (A_I \cup B_I) \cap (V(R) \cup S) \models \varphi' & \text{ if and only if} \\ G[V(R) \cup S], \mathcal{I}', (A'_I \cup B_I) \cap (V(R) \cup S) \models \varphi' & \text{ if and only if} \\ G, \mathcal{I}, A'_I \cup B_I \models \varphi, & \end{aligned}$$

as required. ◀

► **Corollary 20.** *Let I be a finite index set, let σ be a counter I -signature, let φ be a global quantifier-free first-order (I, σ) -sentence, let $G = L \cup R$ be a graph and let \mathcal{I} be a G -interpretation of σ . Then $\equiv_{(L,R)}$ has $\exp(O(|V(L \cap R)|))$ equivalence classes.*

Proof. Let M be the largest integer appearing in the formula φ and the triggers of σ and let a be the number of x -local formulas appearing in φ . Let

$$c = (M + 1)^{\ell(\sigma)2^{\ell(\sigma)}} 2^{|I|2^{\ell(\sigma)}}.$$

Let $h_{\mathcal{I}}$ be the $2^{\ell(\sigma)}$ -shroud corresponding to \mathcal{I} and let $S = h_{\mathcal{I}}(V(L \cap R)) \cap V(L)$.

By Lemma 19, each equivalence class of $\equiv_{(L,R)}$ is determined by

- the $(G, V(L) \setminus S, \sigma, \varphi)$ -census n : since only the vertices of S can have neighbors in $V(L) \setminus S$, the number of such censuses is at most $(M + 1)^{a + \ell(\sigma)|S|} \leq (M + 1)^{a + \ell(\sigma)2^{\ell(\sigma)} \cdot |V(L \cap R)|}$, and
- the restriction of the I -tuple to S : there are $2^{|I| \cdot |S|} \leq 2^{|I|2^{\ell(\sigma)} \cdot |V(L \cap R)|}$ options.

Hence, the number of equivalence classes of $\equiv_{(L,R)}$ is at most $(M + 1)^a c^{|V(L \cap R)|}$. ◀

Proof of Theorem 14. Let T be the tree of the tree decomposition τ , rooted arbitrarily. For each node $x \in V(T)$, let T_x denote the subtree of T induced by x and its descendants. Let $L_x = G[\bigcup_{y \in V(T_x)} \tau(y)]$ and $R_x = G[\bigcup_{y \in (V(T) \setminus V(T_x)) \cup \{x\}} \tau(y)]$, so that $L_x \cup R_x = G$ and $|V(L_x \cap R_x)| = |\tau(x)| \leq t + 1$.

Let σ, φ' and \mathcal{I} be obtained using Theorem 15. We use the standard dynamic programming approach, computing for each $x \in V(T)$ a table assigning to each equivalence class C of $\equiv_{(L_x, R_x)}$ an I -tuple $A_{I,C} \in C$ of subsets of $X \cap V(L_x)$ such that $w(A_{I,C})$ is maximized. Since $\equiv_{(L_x, R_x)}$ has $\exp(O(t))$ equivalence classes by Corollary 20, this can be done in total time $\exp(O(t))|V(G)|$ for all nodes of τ .

Let r be the root of T , and note that $L_r = G$ and $R_r = G[\tau(r)]$. We go over the equivalence classes C of $\equiv_{(L_r, R_r)}$ corresponding to I -tuples satisfying the property expressed by φ' (and thus also by φ), and return the I -tuple $A_{I,C}$ maximizing $w(A_{I,C})$. ◀

References

- 1 Saeed Akhoondian Amiri, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz. Distributed domination on graph classes of bounded expansion. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, pages 143–151, New York, NY, USA, 2018. Association for Computing Machinery.
- 2 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 293–299. ACM, 1990.
- 3 B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.
- 4 Piotr Berman and Marek Karpinski. On some tighter inapproximability results. In *International Colloquium on Automata, Languages, and Programming*, pages 200–209. Springer, 1999.
- 5 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.
- 6 Timothy M Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1576–1585. SIAM, 2012.
- 7 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

- 8 Anuj Dawar, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Approximation schemes for first-order definable optimisation problems. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 411–420. IEEE, 2006.
- 9 Erik D. Demaine and MohammadTaghi Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 590–601. Society for Industrial and Applied Mathematics, 2005.
- 10 Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel Sanders, Bruce Reed, Paul Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *J. Comb. Theory, Ser. B*, 91:25–41, 2004.
- 11 Pål G. 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 *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- 12 Z. Dvořák. Constant-factor approximation of domination number in sparse graphs. *European Journal of Combinatorics*, 34:833–840, 2013.
- 13 Z. Dvořák. Sublinear separators, fragility and subexponential expansion. *European Journal of Combinatorics*, 52:103–119, 2016.
- 14 Z. Dvořák. On classes of graphs with strongly sublinear separators. *European Journal of Combinatorics*, 71:1–11, 2018.
- 15 Z. Dvořák. On distance r -dominating and $2r$ -independent sets in sparse graphs. *J. Graph Theory*, 91(2):162–173, 2019.
- 16 Z. Dvořák. Baker game and polynomial-time approximation schemes. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 2227–2240. Society for Industrial and Applied Mathematics, 2020.
- 17 Z. Dvořák, Daniel Král', and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM (JACM)*, 60(5):36, 2013.
- 18 Z. Dvořák and Sergey Norin. Strongly sublinear separators and polynomial expansion. *SIAM Journal on Discrete Mathematics*, 30:1095–1101, 2016.
- 19 Zdeněk Dvořák. Thin graph classes and polynomial-time approximation schemes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 1685–1701. ACM, 2018.
- 20 Zdeněk Dvořák and Abhiruk Lahiri. Approximation schemes for bounded distance problems on fractionally treewidth-fragile graphs. *arXiv*, 2105.01780, 2021. [arXiv:2105.01780](https://arxiv.org/abs/2105.01780).
- 21 Carl Einarson and Felix Reidl. A General Kernelization Technique for Domination and Independence Problems in Sparse Classes. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- 22 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34:1302–1323, 2005.
- 23 Uriel Feige, MohammadTaghi Hajiaghayi, and James R Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.
- 24 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona De Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–41, 2020.
- 25 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. *Model Theoretic Methods in Finite Combinatorics*, 558:181–206, 2011.

- 26 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 89–98. ACM, 2014.
- 27 Sarel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In *Algorithms-ESA 2015*, pages 717–728. Springer, 2015.
- 28 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Transactions on Algorithms (TALG)*, 15(2):24, 2018.
- 29 R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.
- 30 R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.
- 31 Gary L Miller, Shang-Hua Teng, William Thurston, and Stephen A Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM (JACM)*, 44(1):1–29, 1997.
- 32 J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European J. Combin.*, 29:760–776, 2008.
- 33 J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion II. Algorithmic aspects. *European J. Combin.*, 29:777–791, 2008.
- 34 J. Nešetřil and P. Ossona de Mendez. First order properties on nowhere dense structures. *J. Symbolic Logic*, 75:868–887, 2010.
- 35 J. Nešetřil and P. Ossona de Mendez. *Sparsity (Graphs, Structures, and Algorithms)*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.
- 36 J. Nešetřil, P. Ossona de Mendez, and D. Wood. Characterisations and examples of graph classes with bounded expansion. *Eur. J. Comb.*, 33:350–373, 2012.
- 37 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 789–798, 2018.
- 38 Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *European Journal of Combinatorics*, 75:152–168, 2019.
- 39 Szymon Toruńczyk. Aggregate queries on sparse databases. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 427–443, 2020.
- 40 David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3:103–128, 2007.

Almost Shortest Paths with Near-Additive Error in Weighted Graphs

Michael Elkin ✉

Ben-Gurion University of the Negev, Beer-Sheva, Israel

Yuval Gitlitz ✉

Ben-Gurion University of the Negev, Beer-Sheva, Israel

Ofer Neiman ✉

Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Let $G = (V, E, w)$ be a weighted undirected graph with n vertices and m edges, and fix a set of s sources $S \subseteq V$. We study the problem of computing *almost shortest paths* (ASP) for all pairs in $S \times V$ in both classical centralized and parallel (PRAM) models of computation. Consider the regime of multiplicative approximation of $1 + \varepsilon$, for an arbitrarily small constant $\varepsilon > 0$ (henceforth $(1 + \varepsilon)$ -ASP for $S \times V$). In this regime existing centralized algorithms require $\Omega(\min\{|E|s, n^\omega\})$ time, where $\omega < 2.372$ is the matrix multiplication exponent. Existing PRAM algorithms with polylogarithmic depth (aka time) require work $\Omega(\min\{|E|s, n^\omega\})$.

In a bold attempt to achieve centralized time close to the lower bound of $m + ns$, Cohen [10] devised an algorithm which, in addition to the multiplicative stretch of $1 + \varepsilon$, allows also *additive error* of $\beta \cdot W_{\max}$, where W_{\max} is the maximum edge weight in G (assuming that the minimum edge weight is 1), and $\beta = (\log n)^{O(\frac{\log 1/\rho}{\rho})}$ is polylogarithmic in n . It also depends on the (possibly) arbitrarily small parameter $\rho > 0$ that determines the running time $O((m + ns)n^\rho)$ of the algorithm.

The tradeoff of [10] was improved in [15], whose algorithm has similar approximation guarantee and running time, but its β is $(1/\rho)^{O(\frac{\log 1/\rho}{\rho})}$. However, the latter algorithm produces distance estimates rather than actual approximate shortest paths. Also, the additive terms in [10, 15] depend linearly on a possibly quite large *global* maximum edge weight W_{\max} .

In the current paper we significantly improve this state of affairs. Our centralized algorithm has running time $O((m + ns)n^\rho)$, and its PRAM counterpart has polylogarithmic depth and work $O((m + ns)n^\rho)$, for an arbitrarily small constant $\rho > 0$. For a pair $(s, v) \in S \times V$, it provides a path of length $\hat{d}(s, v)$ that satisfies $\hat{d}(s, v) \leq (1 + \varepsilon)d_G(s, v) + \beta \cdot W(s, v)$, where $W(s, v)$ is the weight of the heaviest edge on some shortest $s - v$ path. Hence our additive term depends linearly on a *local* maximum edge weight, as opposed to the *global* maximum edge weight in [10, 15]. Finally, our $\beta = (1/\rho)^{O(1/\rho)}$, i.e., it is significantly smaller than in [10, 15].

We also extend a centralized algorithm of Dor et al. [14]. For a parameter $\kappa = 1, 2, \dots$, this algorithm provides for *unweighted* graphs a purely additive approximation of $2(\kappa - 1)$ for *all pairs shortest paths* (APASP) in time $\tilde{O}(n^{2+1/\kappa})$. Within the same running time, our algorithm for *weighted* graphs provides a purely additive error of $2(\kappa - 1)W(u, v)$, for every vertex pair $(u, v) \in \binom{V}{2}$, with $W(u, v)$ defined as above.

On the way to these results we devise a suite of novel constructions of spanners, emulators and hopsets.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases spanners, hopset, shortest paths, PRAM, distance oracles

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.23

Related Version *Full Version*: <https://arxiv.org/abs/1907.11422>

Funding *Michael Elkin*: ISF grant 2344/19

Yuval Gitlitz: Partially supported by the Lynn and William Frankel Center for Computer Sciences and ISF grant 970/21.

Ofer Neiman: ISF grant 970/21



© Michael Elkin, Yuval Gitlitz, and Ofer Neiman;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 23; pp. 23:1–23:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We study the problem of computing *almost shortest paths* from a set $S \subseteq V$ of designated vertices, called *sources*, to all other vertices in an n -vertex m -edge weighted undirected graph $G = (V, E, w)$, with non-negative edge weights. We aim at approximation guarantee of the form $(1 + \varepsilon, \beta \cdot W)$, meaning that for every pair $(s, v) \in S \times V$, our algorithm will return a path of length $\hat{d}(s, v)$ (called *distance estimate*) that satisfies

$$d_G(s, v) \leq \hat{d}(s, v) \leq (1 + \varepsilon)d_G(s, v) + \beta \cdot W(s, v), \quad (1)$$

where $W(s, v)$ is the weight of the heaviest edge on some shortest $s - v$ path in G (If there are multiple shortest paths, pick the one with minimal $W(s, v)$).

Here $\varepsilon > 0$ is an arbitrarily small positive constant, and β is typically a large constant that depends on ε , and on the running time of the algorithm.¹ We call this problem $(1 + \varepsilon, \beta \cdot W)$ -ASP for $S \times V$.

The problem of computing approximate shortest paths is one of the most central, fundamental and well-researched problems in Graph Algorithms. We study this problem in both the classical centralized and in the parallel (PRAM CRCW) models of computation. Next, we overview the main previous results for this problem in these two settings, and describe our new results. We start with the centralized model, and then turn our attention to the PRAM model.

1.1 Centralized Setting

The classical algorithm of Dijkstra for the exact *single-source shortest path* (SSSP) problem provides running time of $O(m + n \log n)$ [24]. Thorup [42] devised an algorithm with running time $O(m + n \log \log n)$ for the case when all edge weights are integer. Using this algorithm separately from each of the $s = |S|$ sources results in running time of at least $(m \cdot s)$.

On the opposite end of the spectrum, one can compute $(1 + \varepsilon)$ -APASP (All Pair Approximate Shortest Paths) using matrix multiplication in $\tilde{O}(n^\omega)$ time [26, 4, 45], where $\omega < 2.372$ is the matrix multiplication exponent [13, 44, 27]. There are also a number of combinatorial (i.e., not exploiting fast matrix multiplication) APASP algorithms. In particular, Cohen and Zwick [11] showed that 3-APASP can be computed in $\tilde{O}(n^2)$ time. (They also provided a few additional algorithms with approximation ratio between 2 and 3 and running time greater than n^2 .) Baswana and Kavitha [6] improved their approximation guarantee to $(2, W)$ (with the same running time of $\tilde{O}(n^2)$), and with W defined as in (1).

Finally, Cohen [10] devised an algorithm that for an arbitrarily small parameter $\rho > 0$, solves $(1 + \varepsilon, \beta \cdot W_{\max})$ -ASP for $S \times V$ in $\tilde{O}((m + s \cdot n) \cdot n^\rho)$ time², with $\beta = \beta_{Coh} = (\log n)^{O(\frac{\log(1/\rho)}{\rho})}$. Here W_{\max} is the maximum edge weight in the entire graph (assuming the minimum edge weight is 1).

This result was improved by Elkin [15]. The algorithm of Elkin [15] also solves $(1 + \varepsilon, \beta \cdot W_{\max})$ -ASP for $S \times V$ in $\tilde{O}((m + sn)n^\rho)$ time, with $\beta = \beta_{Elk} = (1/\rho)^{O(\frac{\log(1/\rho)}{\rho})}$. This algorithm reports distance estimates, rather than actual paths.³

¹ In the introduction we will mostly suppress the dependence on ε . It can however be found in the technical part of the paper.

² The running time of the algorithm of [10], as well as of the algorithm of [15] and of our algorithm, is actually slightly better than $O((m + sn)n^\rho)$. Specifically, it is roughly $O(mn^\rho + sn^{1+1/2^{1/\rho}})$. We use the simpler expression of $O((m + sn)n^\rho)$ to simplify presentation. More precise and general bounds can be found in the technical part of the paper.

³ There is a variant of the algorithm of [15] which reports actual paths, but requires time $\tilde{O}((m + sn)n^\rho \cdot W_{\max})$. This running time is typically prohibitively large as it depends linearly on W_{\max} .

The running time of [10, 15] is close (up to n^ρ , for an arbitrarily small constant $\rho > 0$) to the lower bound of $\Omega(m + n \cdot s)$. This is unlike other aforementioned algorithms, whose running time is much larger, i.e., $\Omega(\min\{m \cdot s, n^2\})$. However, the algorithms of [10, 15] suffer from a number of drawbacks. First, their additive term is linear in the maximum edge weight W_{\max} . Second, the coefficient β in them is quite large, even for a relatively large values of the parameter ρ . Third, as was mentioned above, the algorithm of [15] returns distance estimates, as opposed to actual paths that implement these estimates, and in the algorithm of [10] the coefficient β of the additive term is super-constant (specifically, polylogarithmic in n).

In the current paper we address these issues, and devise an algorithm for $(1 + \varepsilon, \beta \cdot W)$ -ASP for $S \times V$ with running time $O((m + sn)n^\rho)$, for an arbitrarily small parameter $\rho > 0$, with $\beta = (1/\rho)^{O(1/\rho)}$. This algorithm does report paths, rather than just distance estimates. Note also that the additive term grows linearly with the *local* maximum edge weight, i.e., with the weight of heaviest edge on each particular source-destination shortest path, as opposed to the *global* maximum edge weight W_{\max} . Finally, its coefficient β is significantly smaller than $\beta_{Elk} = (1/\rho)^{O(\frac{\log(1/\rho)}{\rho})}$, though it is admittedly still quite large⁴. (The coefficient β_{Coh} of [10] depends polylogarithmically on n , while the coefficient β in [15] and here are independent of n .)

We also extend an algorithm of Dor et al. [14] to weighted graphs. Specifically, the algorithm of [14] works for *unweighted* undirected graphs. For any parameter $\kappa = 1, 2, \dots$, it provides an additive $2(\kappa - 1)$ -APASP in $\tilde{O}(n^{2+1/\kappa})$ time. Our extension applies to *weighted* undirected graphs. It computes additive $2(\kappa - 1)W$ -approximation for all pairs shortest paths within the same time $\tilde{O}(n^{2+1/\kappa})$, i.e., for any vertex pair $u, v \in V$, it produces a path of length at most $d_G(u, v) + 2(\kappa - 1) \cdot W(u, v)$, where $W(u, v)$ is as in (1).

Note that the linear dependence of additive error on W is unavoidable, as an algorithm with stretch $(1 + \varepsilon, o(W))$ can be translated into an algorithm with the same running time and with a purely multiplicative stretch of $1 + \varepsilon$.

1.2 Parallel Setting

In the PRAM model, multiple processors are connected to a single memory block, and the operations are performed in parallel by these processors. We will mostly be concerned with the Concurrent Read Concurrent Write (CRCW) PRAM model, that allows multiple processors to access any memory cell at any given round. The running time is measured by the number of rounds, and the work by the number of processors multiplied by the number of rounds.

Early algorithms for these problems [43, 31, 38, 39] require $\Omega(\sqrt{n})$ parallel time. Algorithms of [26, 4, 45], that were discussed in Section 1.1, can also be applied in PRAM. They provide $(1 + \varepsilon)$ -APASP in polylogarithmic time and $\tilde{O}(n^\omega)$ work. The algorithm of Cohen [10], for a parameter $\rho > 0$, solves $(1 + \varepsilon)$ -ASP for $S \times V$ in polylogarithmic time $(\log n)^{O(\frac{\log(1/\rho)}{\rho})}$ and work $\tilde{O}((m + n^{1+\rho})s + m \cdot n^\rho)$.

This tradeoff was then improved in [18, 19], where the running time is $(\log n)^{O(1/\rho)}$, and the work is the same as in [10]. Further spectacular progress was recently achieved by [33, 5], who devised $(1 + \varepsilon)$ -SSSP algorithms with time $(\log n)^{O(1)}$ and work $\tilde{O}(m)$. Nevertheless, for $(1 + \varepsilon)$ -ASP problem from the set $S \subseteq V$ of sources, one needs to run these algorithms in parallel from all the s sources. As a result, their work complexity becomes $\Theta(ms)$.

⁴ We are able to further decrease β to $2^{O(1/\rho)}$, at the expense of increasing the multiplicative stretch from $1 + \varepsilon$ to $3 + \varepsilon$.

To summarize, all existing solutions for the problem with polylogarithmic time have work complexity $\Omega(\min\{m \cdot s, n^\omega\})$. We devise the first algorithm with polylogarithmic time $(\log n)^{O(1/\rho)}$ and work complexity $\tilde{O}((m + ns)n^\rho)$, for an arbitrarily small constant $\rho > 0$. In other words, our work complexity is within n^ρ , for an arbitrarily small constant $\rho > 0$, close to the lower bound of $\Omega(m + ns)$. On the other hand, unlike algorithms of [45, 10, 18, 19, 33, 5], whose approximation guarantee is a purely multiplicative $1 + \varepsilon$, for an arbitrarily small $\varepsilon > 0$, our approximation guarantee is $(1 + \varepsilon, \beta \cdot W)$, with $\beta = (1/\rho)^{O(1/\rho)}$, and W as in (1).

Moreover, our result can, in fact, be viewed as a *PRAM distance oracle*. Specifically, following the preprocessing that requires time $(\log n)^{O(1/\rho)}$ and work $\tilde{O}(mn^\rho)$, our algorithm stores a compact data structure of size $\tilde{O}(n^{1+\rho})$. Given a query vertex s , this data structure provides distance estimates $\hat{d}(s, v)$ for all $v \in V$, which satisfies (1) in *constant time* and using work $\tilde{O}(n^{1+\rho})$, where $\beta = (1/\rho)^{O(1/\rho)}$. Note that the distance oracle has size arbitrarily close to linear in n , its preprocessing time is polylogarithmic and preprocessing work is arbitrarily close to linear in m , and the query time is constant and the query work is arbitrarily close to linear in n . (Note that as the query provides distance estimates for n vertex pairs $\{(s, v) \mid v \in V\}$, its query work complexity must be $\Omega(n)$.)

1.3 Hopsets, Spanners and Emulators

From the technical viewpoint, these results are achieved via a combination of our new constructions of emulators, spanners and hopsets. For parameters $\alpha, \beta \geq 1$, we say that a graph $H = (V, E', w)$ is an (α, β) -hopset for a (weighted) graph $G = (V, E, w)$, if by adding E' to the graph, every pair $x, y \in V$ has an α -approximate shortest path consisting of at most β hops; Formally,

$$d_G(x, y) \leq d_{G \cup H}^{(\beta)}(x, y) \leq \alpha \cdot d_G(x, y) ,$$

where $d_{G \cup H}^{(\beta)}$ is the shortest path in $G \cup H$ containing at most β edges. The parameter α is called the *stretch*, and β is the *hopbound*.

We say that H is an (α, β) -emulator if for every $x, y \in V$,

$$d_G(x, y) \leq d_H(x, y) \leq \alpha \cdot d_G(x, y) + \beta ,$$

and H is a *spanner* if it is an emulator and a subgraph of G .

Hopsets and near-additive spanners are fundamental combinatorial constructs, and play a major role in efficient approximation of shortest paths in various computational models. These objects have been extensively investigated in recent years [21, 16, 41, 23, 36, 37, 7, 17, 32, 10, 9, 35, 28, 34, 29, 25, 18, 2, 19, 30]. The main interest is to understand the triple tradeoff between the size of the hopset (respectively, spanner), to the stretch α , and to the hopbound (resp., additive stretch) β . For algorithmic applications, it is also crucial to bound the construction time of the hopset/spanner/emulator.

We show near-additive spanners for *weighted* graphs, where the additive stretch for the pair x, y may depend also on the largest edge weight on the corresponding shortest path from x to y , $W(x, y)$. For a parameter $0 < \rho < 1$, we devise an algorithm that constructs a $(1 + \varepsilon, \beta \cdot W)$ -spanner of size $O(n^{1+1/2^{1/\rho}} + n/\rho)$ with $\beta \leq \left(\frac{1/\rho}{\varepsilon}\right)^{O(1/\rho)}$. We also show how to analyze the construction so that it yields smaller additive stretch, while increasing the multiplicative one. Specifically, we get a $(c, \beta \cdot W)$ -spanner of same size as above, for every constant $c > 3$, and with $\beta = \left(\frac{1}{\varepsilon}\right)^{O(1/\rho)}$. Our emulators have a somewhat improved β . All of our results admit near-linear time algorithms, i.e., their running time is $O(|E|n^\rho)$, for an arbitrarily small constant $\rho > 0$.

1.4 Technical Overview

We adapt the constructions of [41, 36, 17, 19, 30] of hopsets, spanners and emulators so that they are suitable for weighted graphs, and provide an improved additive stretch (or hopbound) β . The basic idea in all these constructions is to generate a random hierarchy of vertex sets $V = A_0 \supseteq A_1 \supseteq \dots \supseteq A_k = \emptyset$, where for each $0 \leq i < k - 1$, each element in A_i is included in A_{i+1} with probability $\approx n^{-2^{i-k}}$ (one should think of $k = 1/\rho$). We also refer to vertices at A_i as vertices at level i . For each $v \in V$, the pivot $p_i(v)$ is the closest vertex in A_i to v . Then the set of edges H is created by connecting, for every $0 \leq i \leq k - 1$, every vertex $v \in A_i \setminus A_{i+1}$ to its *bunch*: all other vertices in A_i that are closer to it than $p_{i+1}(v)$. One difference in our construction is that we also connect each vertex to all its (at most k) pivots. The main technical innovation in this work is the new analysis of this construction, yielding various hopsets, emulators and spanners that apply for *weighted* graphs, and admit improved parameters. (Previous constructions of spanners and emulators [21, 41, 36, 17, 30] applied only for unweighted graphs.)

The previous analysis of the stretch for some pair $x, y \in V$ goes roughly as follows. Divide the $x - y$ path into intervals, and try to connect these intervals using low stretch paths in H (and for hopsets, also some of the graph edges, but with few hops). Each interval can either have a low stretch path; or fail, in which case that interval admits a nearby pivot (of some level i). Then, consider two failed intervals (the leftmost and rightmost ones), and try to find an $x - y$ path via the pivots of these intervals.

We note that this partitioning of the $x - y$ path to equal size intervals (used in previous works), cannot work directly for weighted graphs, since it may not be possible to divide the path to intervals of equal (or even near-equal) size. We provide a subtle adaptation of this technique so that it can handle weights. In particular, we distinguish between short and long distances: The sufficiently long distances may suffer a partition to un-equal intervals, as the induced error is dominated by the multiplicative stretch of $1 + \varepsilon$. For the short distances, we stop this partitioning when it becomes too “expensive”, and resort to an argument similar to the one in [40], which has large multiplicative stretch (of roughly $2^{1/\rho}$). However, at the point where we stop, that stretch can be accounted for by the additive stretch $\beta \cdot W$.

An additional ingredient in our new analysis of H as an emulator/spanner for weighted graphs, given a pair $x, y \in V$, is to iteratively find a vertex z on the $x - y$ shortest path (sufficiently far from x) that admits in H a path with low multiplicative stretch from x . When there is no such z , we show that we can reach y with small additive stretch. This technique can be used to obtain the improved dependence of β on the parameter $k = 1/\rho$.

Recall that emulators are insufficient for reporting paths. In particular, the approach of [10, 15] was based on emulators, rather than on spanners, and it is not clear if these algorithms can be adapted to build spanners (for weighted graphs). On the other hand, with our approach we can convert our constructions of emulators into constructions of spanners. Specifically, to build a spanner, we must use graph edges. So in order to connect vertices $v \in A_i \setminus A_{i+1}$ to the vertices in their bunch $B(v)$, we need to add paths of possibly many edges in the graph (rather than a single edge, as for emulators/hopsets). To do this we connect every vertex v to all vertices in its *half-bunch* (see Section 4 for its definition), as opposed to connecting it to all vertices in its full bunch. (The latter is the case in the construction of emulators.) This turns out to be sufficient to ensure that the union of all these paths does not contain too many edges. For this analysis we employ ideas of counting pairwise path intersections, developed in the context of distance preservers [12] and near-additive spanners and distance oracles for unweighted graphs [36, 22]. We simplify this approach, and extend it to weighted graphs.

1.5 Organization

In Section 2 we describe the construction of our emulators and hopsets. In Section 3 we analyze this construction, showing it provides emulators for weighted graphs. In Section 4 we describe the construction of our spanners for weighted graphs (proofs are deferred to Appendix D) and use them for our centralized algorithms for the ASP problem. In Section 5 we provide efficient implementations of our constructions, and use them in Section 6 for solving ASP in PRAM, and for PRAM distance oracles.

Our hopsets and emulators with improved β appear in Appendices A and B.

Our algorithm for pure additive APASP for weighted graphs is available in the full version of the paper.

Bibliographical note

Related results about (α, β) -spanners for *unweighted* graphs and (α, β) -hopsets with $\alpha \geq 3 + \varepsilon$ were achieved independently of us and simultaneously by [8]. In another submission [20], an $(1 + \varepsilon)$ -ASP algorithm for $S \times V$ with $|S| = n^r$, for some $0 < r \leq 1$, was devised. The running time of this algorithm in the centralized setting is $\tilde{O}(n^{\omega(r)})$, where $\omega(r)$ is the rectangular matrix multiplication exponent. ($n^{\omega(r)}$ is the time required to multiply an $n^r \times n$ matrix by an $n \times n$ one.) In PRAM setting that algorithm runs in polylogarithmic time and $\tilde{O}(n^{\omega(r)})$ work. For graphs $G = (V, E, w)$ and sets of sources $S \subseteq V$ of size s such that $m + ns = o(n^2)$, the algorithms that we devise in the current submission are more efficient than in [20].

Following our work, [3] devised algorithms for purely additive spanners (e.g., spanners with multiplicative stretch exactly 1) for weighted graphs. In their results, the additive term depends on the global maximal edge weight W_{\max} , and the size of their spanners is always $\Omega(n^{4/3})$ (the latter is unavoidable for purely additive spanners, due to a lower bound of [1]).

2 Construction

We use a similar construction to that of [41, 19, 30]. One difference is that every vertex connects to pivots in all levels. (Recall that the main difference is in the analysis.) Let $G = (V, E, w)$ be a weighted graph with n vertices, and fix an integer parameter $k \geq 1$. Let $\nu = 1/(2^k - 1)$. Let $A_0 \dots A_k$ be sets of vertices such that $A_0 = V$, $A_k = \emptyset$, and for $0 \leq i \leq k - 2$, A_{i+1} is created by sampling independently every element from A_i with probability $q_i = n^{-2^i \nu} \cdot 2^{-2^i - 1}$. For every $0 \leq i \leq k - 1$, the expected size of A_i is:

$$N_i = E[|A_i|] = n \prod_{j=0}^{i-1} q_j = n^{1 - (2^i - 1)\nu} \cdot 2^{-2^i - i + 1}$$

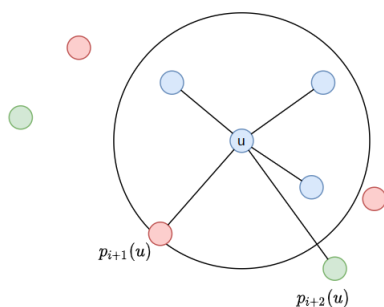
For every $i \in [k - 1]$, define the pivot $p_i(v)$ to be the closest vertex in A_i to v , breaking ties by lexicographic order. For every $u \in A_i \setminus A_{i+1}$ define the bunch (see Figure 1)

$$B(u) = \{v \in A_i \mid d_G(u, v) < d_G(u, A_{i+1})\} \cup \{p_j(u) \mid i < j < k\}. \quad (2)$$

That is, the bunch $B(u)$ contains all the vertices which are in A_i and closer to u than $p_{i+1}(u)$, and at most k pivots. We then define $H = \{(u, v) : u \in V, v \in B(u)\}$, where the weight of the edge (u, v) is set as the weight of the shortest path between u, v in G .

► **Lemma 1.** *The size of H is*

$$H = O(kn + n^{1+\nu})$$



■ **Figure 1** The bunch of u . Here vertices in $A_i \setminus A_{i+1}$ are colored in blue, in $A_{i+1} \setminus A_{i+2}$ are colored in red and in A_{i+2} are colored in green.

The proof of Lemma 1 is similar to previous works, we include it for completeness in appendix C.

3 Near-Additive Emulators for Weighted Graph

In this section we will show that H as defined in Section 2 is a $(1 + \varepsilon, \beta \cdot W)$ -emulator for weighted graphs. Note that the construction there does not depend on ε , and indeed, H will be an emulator for all values of $0 < \varepsilon < 1$ simultaneously (with $\beta = \beta(\varepsilon)$ depending on ε).

Let $G = (V, E)$ be a weighted graph with non-negative weights $w : E \rightarrow \mathbb{R}_+$, recall that for $x, y \in V$ we have $W(x, y) = \max\{w(e) : e \in P_{xy}\}$ (where P_{xy} is a shortest path from x to y in G). Let $k \geq 1$ and $\Delta > 3$ be given parameters (think of $\Delta = 3 + O(k/\varepsilon)$). We begin by proving two lemmas, handling long and short distances, respectively. The first lemma asserts that pairs which are sufficiently far apart admit either a low stretch path, or a nearby pivot of a higher level.

► **Lemma 2.** Fix $\Delta > 3$. Let $0 \leq i < k$ and let $x, y \in V$ such that $d_G(x, y) \geq (3\Delta)^i W(x, y)$. Then at least one of the following holds:

1. $d_H(x, y) \leq (1 + \frac{4i}{\Delta-3})d_G(x, y)$
 2. $d_H(x, p_{i+1}(x)) \leq \frac{\Delta}{\Delta-3}d_G(x, y)$
- (For $i = k - 1$, the first item must hold since p_{i+1} doesn't exist).

Proof. Denote $W = W(x, y)$. The proof is by induction on i . For the base case $i = 0$, if $y \in B(x)$ then $d_H(x, y) = d_G(x, y)$ and the first item holds. Otherwise, if $x \in A_1$ then $d_G(x, p_1(x)) = 0$, so the second item holds. The last case is that $x \in A_0 \setminus A_1$ and $y \notin B(x)$, then by (2) we have $d_H(x, p_1(x)) = d_G(x, p_1(x)) \leq d_G(x, y) \leq \frac{\Delta}{\Delta-3}d_G(x, y)$, thus the second item holds. Assume the lemma holds for i and prove for $i + 1$. Let $x, y \in V$ be a pair of vertices such that $d_G(x, y) \geq (3\Delta)^{i+1}W(x, y)$

Divide the shortest path between x and y into J segments $\{L_j = [u_j, u_{j+1}]\}_{j \in [J]}$ of length at least $(3\Delta)^i W$ and at most $d_G(x, y)/\Delta$. It can be done as follows: define $u_1 = x, j = 2$ and walk on the shortest path from x to y . Define u_j as the first vertex which $d_G(u_{j-1}, u_j) \geq (3\Delta)^i W$ or define $u_j = y$ if $d_G(u_{j-1}, y) < (3\Delta)^i W$. Increase j by 1 and repeat. Note that each segment has length at most $(3\Delta)^i W + W$. Finally, we join the last two segments. The length of the last segment is at most $(3\Delta)^i W + W + (3\Delta)^i W \leq 3^{i+1} \Delta^i W \leq d_G(x, y)/\Delta$. The length of any segment except the last is also at most $(3\Delta)^i W + W \leq d_G(x, y)/\Delta$.

23:8 Almost Shortest Paths with Near-Additive Error in Weighted Graphs

Apply the induction hypothesis for every segment L_j with parameter i . If for all the segments the first item holds, then first item holds for x, y and $i + 1$, since

$$\begin{aligned} d_H(x, y) &\leq \sum_{j \in J} d_H(u_j, u_{j+1}) \leq \sum_{j \in J} \left(1 + \frac{4i}{\Delta - 3}\right) d_G(u_j, u_{j+1}) \\ &\leq \left(1 + \frac{4i}{\Delta - 3}\right) d_G(x, y). \end{aligned}$$

Otherwise, for at least one segment the second item holds. Let L_l (resp., L_{r-1}) be the leftmost (resp., rightmost) segment for which the second item holds. We have that the second item holds for the pair u_l, u_{l+1} , and by symmetry of the first item, the second item also holds for the pair u_r, u_{r-1} with parameter i . Hence

$$\begin{aligned} d_H(u_r, p_{i+1}(u_r)) &\leq \frac{\Delta}{\Delta - 3} d_G(u_{r-1}, u_r) \leq \frac{d_G(x, y)}{\Delta - 3}, \\ d_H(u_l, p_{i+1}(u_l)) &\leq \frac{\Delta}{\Delta - 3} d_G(u_l, u_{l+1}) \leq \frac{d_G(x, y)}{\Delta - 3}. \end{aligned} \quad (3)$$

Consider first the case that $p_{i+1}(u_r) \in B(p_{i+1}(u_l))$. In this case H contains the edge $\{p_{i+1}(u_l), p_{i+1}(u_r)\}$, so we have

$$d_H(p_{i+1}(u_l), p_{i+1}(u_r)) \leq d_G(p_{i+1}(u_l), u_l) + d_G(u_l, u_r) + d_G(u_r, p_{i+1}(u_r)). \quad (4)$$

By the triangle inequality,

$$d_H(u_l, u_r) \quad (5)$$

$$\leq d_H(u_l, p_{i+1}(u_l)) + d_H(p_{i+1}(u_l), p_{i+1}(u_r)) + d_H(p_{i+1}(u_r), u_r)$$

$$\stackrel{(4)}{\leq} 2d_H(u_l, p_{i+1}(u_l)) + d_G(u_l, u_r) + 2d_H(p_{i+1}(u_r), u_r)$$

$$\stackrel{(3)}{\leq} \frac{4d_G(x, y)}{\Delta - 3} + d_G(u_l, u_r). \quad (6)$$

Thus, the distance between x and y in H ,

$$\begin{aligned} d_H(x, y) &\leq \sum_{j \in [J]} d_H(u_j, u_{j+1}) \\ &\leq \sum_{j=1}^{l-1} \left(1 + \frac{4i}{\Delta - 3}\right) d_G(u_j, u_{j+1}) + d_H(u_l, u_r) + \sum_{j=r}^J \left(1 + \frac{4i}{\Delta - 3}\right) d_G(u_j, u_{j+1}) \\ &\leq \left(1 + \frac{4i}{\Delta - 3}\right) d_G(x, u_l) + d_H(u_l, u_r) + \left(1 + \frac{4i}{\Delta - 3}\right) d_G(u_r, y) \\ &\stackrel{(6)}{\leq} \left(1 + \frac{4i}{\Delta - 3}\right) d_G(x, u_l) + \frac{4d_G(x, y)}{\Delta - 3} + d_G(u_l, u_r) + \left(1 + \frac{4i}{\Delta - 3}\right) d_G(u_r, y) \\ &\leq \left(1 + \frac{4(i+1)}{\Delta - 3}\right) d_G(x, y), \end{aligned}$$

therefore the first item holds.

The other case is that $p_{i+1}(u_r) \notin B(p_{i+1}(u_l))$. Then

$$d_H(p_{i+1}(u_l), p_{i+2}(p_{i+1}(u_l))) \leq d_G(p_{i+1}(u_l), p_{i+1}(u_r)). \quad (7)$$

We can bound the distance $d_H(x, p_{i+2}(x))$ by

$$\begin{aligned}
d_H(x, p_{i+2}(x)) &\leq d_G(x, u_l) + d_G(u_l, p_{i+1}(u_l)) + d_G(p_{i+1}(u_l), p_{i+2}(p_{i+1}(u_l))) \\
&\leq d_G(x, u_l) + d_G(u_l, p_{i+1}(u_l)) + d_G(p_{i+1}(u_l), p_{i+1}(u_r)) \\
&\stackrel{(4)}{\leq} d_G(x, u_l) + d_G(u_l, p_{i+1}(u_l)) + d_G(u_l, p_{i+1}(u_l)) + d_G(u_l, u_r) + d_G(p_{i+1}(u_r), u_r) \\
&\stackrel{(3)}{\leq} d_G(x, y) + \frac{3d_G(x, y)}{\Delta - 3} = \frac{\Delta}{\Delta - 3}d_G(x, y).
\end{aligned}$$

Hence the second item holds. ◀

The previous lemma is useful for vertices which are very far from each other, since for $i = k - 1$ the first item must hold. For vertices which are close to each other, we will need the following lemma.

► **Lemma 3.** *Let $0 \leq i < k$ and fix $x, y \in V$. Let $m = \max\{d_G(x, p_i(x)), d_G(y, p_i(y)), d_G(x, y)\}$. Then at least one of the following holds:*

1. $d_H(x, y) \leq 5m$
2. $d_H(x, p_{i+1}(x)) \leq 4m$ and $d_H(y, p_{i+1}(y)) \leq 4m$

(For $i = k - 1$, the first item must hold since p_{i+1} doesn't exist).

Proof. If $p_i(y) \in B(p_i(x))$ the first item holds, since

$$\begin{aligned}
d_H(x, y) &\leq d_H(x, p_i(x)) + d_H(p_i(x), p_i(y)) + d_H(p_i(y), y) \\
&\leq d_G(x, p_i(x)) + d_G(p_i(x), x) + d_G(x, y) + d_G(y, p_i(y)) + d_G(p_i(y), y) \\
&\leq 5m.
\end{aligned}$$

If $p_i(y) \notin B(p_i(x))$, then $d_G(p_i(x), p_{i+1}(p_i(x))) \leq d_G(p_i(x), p_i(y))$, in this case the second item holds, as

$$\begin{aligned}
d_H(x, p_{i+1}(x)) &\leq d_H(x, p_i(x)) + d_H(p_i(x), p_{i+1}(p_i(x))) \\
&\leq d_G(x, p_i(x)) + d_G(p_i(x), x) + d_G(x, y) + d_G(y, p_i(y)) \leq 4m.
\end{aligned}$$

The bound on $d_H(y, p_{i+1}(y))$ is symmetric. ◀

We are now ready to prove the following theorem.

► **Theorem 4.** *For any weighted graph $G = (V, E)$ on n vertices, and any integer $k > 1$, there exists H of size at most $O(kn + n^{1+1/(2^k-1)})$, which is a $(1 + \varepsilon, \beta \cdot W)$ -emulator for any $0 < \varepsilon < 1$, where $\beta = O(\frac{k}{\varepsilon})^{k-1}$.*

Proof. Fix $\Delta = 3 + \frac{4(k-1)}{\varepsilon}$, $\beta = 10(3\Delta)^{k-1}$. Let $x, y \in V$, and $W = W(x, y)$. If $d_G(x, y) \geq (3\Delta)^{k-1}W$, we can apply Lemma 2 for x, y and $i = k - 1$. Since $p_k(x)$ does not exist, the first item must hold. Thus,

$$d_H(x, y) \leq \left(1 + \frac{4(k-1)}{\Delta - 3}\right) d_G(x, y) = (1 + \varepsilon)d_G(x, y).$$

Otherwise, take the integer $0 \leq i < k - 1$ satisfying $(3\Delta)^i W \leq d_G(x, y) < (3\Delta)^{i+1}W$ (note that there must be such an i , since $d_G(x, y) \geq W$). Apply Lemma 2 for x, y and i . If the first item holds, we will get $1 + \varepsilon$ stretch as before.

23:10 Almost Shortest Paths with Near-Additive Error in Weighted Graphs

Otherwise, the second item holds, and we know that $d_G(x, p_{i+1}(x)) \leq \frac{\Delta}{\Delta-3} d_G(x, y) \leq 2d_G(x, y)$ (using that $k \geq 2$). By symmetry of x, y in the first item of Lemma 2, we have $d_G(y, p_{i+1}(y)) \leq 2d_G(x, y)$ as well. Set $j = i + 1$ and apply Lemma 3 with x, y, j , noting that $m \leq 2d_G(x, y)$. If the first item holds, we found a path in H from x to y of length at most $5m \leq 10d_G(x, y) \leq 10(3\Delta)^{k-1}W = \beta \cdot W$.

If the second item holds, we increase j by one and apply Lemma 3 again. We continue this procedure until the first item holds. The fact that the second item holds implies that the bound m (the maximal distance of x, y to the level j pivots) increases every iteration by a factor of at most 4. Since the first item must hold for $j = k - 1$, the path we found is of length at most $10 \cdot 4^{k-i-2} d_G(x, y) \leq 10 \cdot 4^{k-i-2} \cdot (3\Delta)^{i+1}W$, which is maximized for $i = k - 2$. Hence the additive stretch is at most $10 \cdot (3\Delta)^{k-1}W = O((9 + \frac{12(k-1)}{\varepsilon})^{k-1}W)$, as required. ◀

► **Remark 5.** We note that the analysis did not use the fact that the path from x to y is a shortest path. In particular, for every path P from x to y of length d , we can obtain a path in H of length at most $(1 + \varepsilon) \cdot d + \beta \cdot W(P)$, where $W(P)$ is the largest edge weight in P .

4 Near-Additive Spanners for Weighted Graphs

In this section we devise our spanners for weighted graphs. We first describe the new construction, that differs from that of Section 2 in several aspects, which are required in order to keep the size of the spanner under control (and independent of W_{\max}). In particular, since we add paths, rather than edges, between each vertex to other vertices in its bunch, we need to ensure the size of H is small enough. To do that, we use half-bunches rather than bunches to define H (see (8) below), and show that there are few intersections between the aforementioned paths (see Lemma 17). One last ingredient is altering the sampling probabilities, so that the argument on intersection goes through. This approach refines and improves ideas from [36, 22].

Construction

Let $G = (V, E)$ be a weighted graph with n vertices, and fix an integer parameter $k \geq 3$. Let $\nu = \frac{1}{(4/3)^{k-1}}$. Let $A_0 \dots A_k$ be sets of vertices such that $A_0 = V$, $A_k = \emptyset$, and for $0 \leq i \leq k-2$, A_{i+1} is created by sampling every element from A_i with probability $q_i = n^{-4^i \nu / 3^{i+1}}$. For every $0 \leq i \leq k-1$, the expected size of A_i is:

$$N_i = E[|A_i|] = n \prod_{j=0}^{i-1} q_j = n^{1 - \frac{\nu}{3} \sum_{j=0}^{i-1} (4/3)^j} = n^{1 - ((4/3)^i - 1)\nu}.$$

For every $i \in [k-1]$, define the pivot $p_i(v)$ to be the closest vertex in A_i to v , breaking ties by lexicographic order. For every $u \in A_i \setminus A_{i+1}$ define the *half bunch*

$$B_{1/2}(u) = \{v \in A_i : d_G(u, v) < d_G(u, A_{i+1})/2\}. \quad (8)$$

Let $H = \{P_{uv} : u \in V, v \in B_{1/2}(u) \cup \{p_j(u) | i < j < k\}\}$, where P_{uv} is the shortest path between u, v in G (if there is more than one, break ties consistently, by vertex id, say).

► **Theorem 6.** *For any weighted graph $G = (V, E)$ on n vertices, and any integer $k > 2$, there exists H of size at most $O(kn + n^{1+1/((4/3)^k - 1)})$, which is a $(1 + \varepsilon, \beta \cdot W)$ -spanner for any $0 < \varepsilon < 1$, with $\beta = O(k/\varepsilon)^{k-1}$.*

► **Theorem 7.** *For any weighted graph $G = (V, E)$ on n vertices, and any integer $k > 2$, there exists H of size at most $O(kn + n^{1+1/((4/3)^k-1)})$, which is a $(3 + \varepsilon, \beta \cdot W)$ -spanner for any $\varepsilon > 0$ with $\beta = O(1 + 1/\varepsilon)^{k-1}$.*

Full proofs for both theorems are given in appendix D.

5 Efficient Implementation

Since we use very similar constructions to the ones in [19], we can use their efficient implementations (connecting to all pivots, which is the difference between constructions, can be done efficiently in their framework as well). We consider here the standard model of computation, and the PRAM (CRCW) model. Given a parameter $1/k < \rho < 1/2$, we will want poly-logarithmic parallel time and $\tilde{O}(|E| \cdot n^\rho)$ work / centralized time. This is achieved by adding additional $\lceil 1/\rho \rceil$ sets A_i , that are sampled with uniform probability $n^{-\rho}$, which in turn increases the exponent of β by an additive $1/\rho + 1$. (In the case of multiplicative stretch $1 + \varepsilon$, it also increases the base of the exponent in β .)

We summarize the efficient implementation results for hopsets and emulators in the following theorem.

► **Theorem 8.** *For any weighted graph $G = (V, E)$ on n vertices, parameters $k > 2$ and $1/k < \rho < 1/2$, there is a randomized algorithm running in time $\tilde{O}(|E| \cdot n^\rho)$, that w.h.p. computes H of size at most $O(kn + n^{1+1/(2^k-1)})$, such that for any $0 < \varepsilon < 1$ this H is:*

1. A $(1 + \varepsilon, \beta \cdot W)$ -emulator with $\beta = O\left(\frac{k+1/\rho}{\varepsilon}\right)^{k+1/\rho}$.
2. A $(3 + \varepsilon, \beta \cdot W)$ -emulator with $\beta = O(1/\varepsilon)^{k+1/\rho}$.
3. A $(3 + \varepsilon, \beta)$ -hopset with $\beta = O(1/\varepsilon)^{k+1/\rho}$.

Given ε in advance, the algorithm can also be implemented in the PRAM (CRCW) model, in parallel time $\left(\frac{\log n}{\varepsilon}\right)^{O(k+1/\rho)}$ and work $\tilde{O}(|E| \cdot n^\rho)$, while increasing the size of H by a factor of $O(\log^* n)$.

For spanners, recall that in Section 4 we have a somewhat different construction, and in the analysis we enforce a stricter requirement on the sampling probabilities q_i . To handle this, we start sampling with the uniform probability $n^{-\rho}$ only when $N_i \leq n^{1-3\rho}$ (and not when $N_i \leq n^{1-\rho}$ like before). Now the bound of Claim 18 still holds, as $N_i/q_i^3 \leq n$ even for these latter sets. The “price” we pay for waiting until $N_i \leq n^{1-3\rho}$ is that the work will now be $|E| \cdot n^{3\rho}$. Rescaling ρ by 3, we get the following.

► **Theorem 9.** *For any weighted graph $G = (V, E)$ on n vertices, parameters $k > 6$ and $1/k < \rho < 1/6$, there is a randomized algorithm running in time $\tilde{O}(|E| \cdot n^\rho)$, that w.h.p. computes H of size at most $O(kn + n^{1+1/(2^k-1)})$, such that for any $0 < \varepsilon < 1$ this H is:*

1. A $(1 + \varepsilon, \beta \cdot W)$ -spanner with $\beta = O\left(\frac{k+1/\rho}{\varepsilon}\right)^{k+3/\rho}$.
2. A $(3 + \varepsilon, \beta \cdot W)$ -spanner with $\beta = O(1/\varepsilon)^{k+3/\rho}$.

Given ε in advance, the algorithm can also be implemented in the PRAM (CRCW) model, in parallel time $\left(\frac{\log n}{\varepsilon}\right)^{O(k+1/\rho)}$ and work $\tilde{O}(|E| \cdot n^\rho)$, while increasing the size of H by a factor of $O(\log^* n)$.

6 Almost Shortest Paths in Weighted Graphs

Given a weighted graph $G = (V, E, w)$ with n vertices and a set $S \subseteq V$ of s sources, fix parameters $k > 6$, $0 < \varepsilon < 1$ and $0 < \rho < 1/6$. Here we show that our hopsets, emulators and spanners can be used for a $(1 + \varepsilon, \beta \cdot W)$ -approximate shortest paths for pairs in $S \times V$, in various settings.

For the standard centralized setting, we first compute a $(1 + \varepsilon, \beta \cdot W)$ -spanner H of size $O(kn + n^{1+1/(2^k-1)})$ with $\beta = O(\frac{k+3/\rho}{\varepsilon})^{k+3/\rho}$, in time $\tilde{O}(|E| \cdot n^\rho)$ as in Theorem 9. Next, for every $u \in S$ run Dijkstra's shortest path algorithm in H , which takes time $O(s \cdot (|E(H)| + n \log n)) = \tilde{O}(s \cdot n^{1+1/(2^k-1)})$.

The total running time for computing $(1 + \varepsilon, \beta \cdot W)$ -approximate shortest path for all $S \times V$, is $\tilde{O}(|E| \cdot n^\rho + s \cdot n^{1+1/(2^k-1)})$. One may choose $\rho = 1/k$ and obtain the following.

► **Theorem 10.** *For any weighted graph $G = (V, E)$ on n vertices, a set $S \subseteq V$ of s sources, and parameters $k > 6$ and $0 < \varepsilon < 1$, there is a randomized algorithm running in time $\tilde{O}(|E| \cdot n^{1/k} + s \cdot n^{1+1/(2^k-1)})$, that computes $(1 + \varepsilon, \beta \cdot W)$ -approximate shortest paths for all pairs in $S \times V$, where $\beta = (\frac{k}{\varepsilon})^{O(k)}$.*

We remark that there is a more general tradeoff by choosing ρ as a free parameter. In addition, if one desires improved additive stretch, simply use the $(3 + \varepsilon, \beta \cdot W)$ -spanner with $\beta = \varepsilon^{-O(k)}$.

6.1 PRAM Shortest Paths and Distance Oracles

Given a weighted graph $G = (V, E, w)$ with n vertices, fix parameters $k \geq 1$, $0 < \varepsilon < 1$ and $0 < \rho < 1/6$. The first step in both settings (computing approximate shortest paths and distance oracles) is the same. We construct a $(1 + \varepsilon, \beta \cdot W)$ -emulator G' of size $O(kn + n^{1+1/(2^k-1)}) \cdot \log^* n$ with $\beta = O(\frac{k+1/\rho}{\varepsilon})^{k+1/\rho}$, in parallel time $(\frac{\log n}{\varepsilon})^{O(k+1/\rho)}$ and work $\tilde{O}(|E| \cdot n^\rho)$ as in Theorem 8. Next, compute a $(1 + \varepsilon, \beta)$ -hopset H size $O(n^{1+1/(2^k-1)}) \cdot \log^* n$ for G' with the same $\beta = O(\frac{k+1/\rho}{\varepsilon})^{k+1/\rho}$ within the same parallel time and work [19].⁵ We store both G' and H .

Approximate Shortest Paths

For the sake of simplicity we will choose $\rho = 1/k$. Given a set S of s sources, for each source $u \in S$ run β rounds of the Bellman-Ford algorithm in the graph $G' \cup H$ starting at u . In each round of Bellman-Ford, every vertex sends its neighbors the current distance estimate to u that it has, and they update their distance estimate if needed. Since $G' \cup H$ is a sparse graph with $\tilde{O}(n^{1+1/(2^k-1)})$ edges, with $\tilde{O}(n^{1+1/(2^k-1)})$ processors one can implement each iteration in PRAM (CRCW) in $O(2^k)$ time (see [19] for more details). As we have only β rounds, the total parallel time for all the Bellman-Ford rounds from all vertices in S is $(k/\varepsilon)^{O(k)}$, and the total work is $\tilde{O}(s \cdot n^{1+1/(2^k-1)})$.

As the error of the emulator is $(1 + \varepsilon, \beta \cdot W)$, and the hopset has only multiplicative $1 + \varepsilon$ stretch, the total error is only $(1 + O(\varepsilon), O(\beta \cdot W))$. We thus have the following result.

⁵ We remark that even though the emulator and hopset have exactly the same construction, we run the hopset algorithm on G' and not on G , thus we get a different set of edges.

► **Theorem 11.** For any weighted graph $G = (V, E)$ on n vertices, a set $S \subseteq V$ of s sources, and parameters $k > 2$ and $0 < \varepsilon < 1$, there is a PRAM randomized algorithm running in $\left(\frac{\log n}{\varepsilon}\right)^{O(k)}$ parallel time and using $\tilde{O}(|E| \cdot n^{1/k} + s \cdot n^{1+1/(2^k-1)})$ work, that computes $(1 + \varepsilon, \beta \cdot W)$ -approximate shortest paths for all pairs in $S \times V$, where $\beta = \left(\frac{k}{\varepsilon}\right)^{O(k)}$.

As above, we can get a more general tradeoff with the parameter ρ , and improve the additive stretch by using the hopsets and emulators from Sections A,B, albeit the multiplicative stretch will increase.

Distance Oracles

Recall that we store the emulator G' and a hopset H for G' . Whenever a query $u \in V$ arrives, we run β rounds of Bellman-Ford algorithm in the graph $G' \cup H$. As noted above, each round of Bellman-Ford can be implemented in PRAM (CRCW) in $O(2^k)$ time using $\tilde{O}(n^{1+1/(2^k-1)})$ processors. So the total parallel time for the query is $O\left(\frac{k+1/\rho}{\varepsilon}\right)^{k+1/\rho}$. Rescaling ε , we get a $\left(1 + \varepsilon, O\left(\frac{k+1/\rho}{\varepsilon}\right)^{k+1/\rho} \cdot W\right)$ -approximation. We conclude that the properties of the distance oracle we devise are:

- Has size $O(kn + n^{1+1/(2^k-1)}) \cdot \log^* n$.
- Given query $u \in V$, can report $(1 + \varepsilon, \beta \cdot W)$ -approximation to *all* distances in $\{u\} \times V$, with $\beta = O\left(\frac{k+1/\rho}{\varepsilon}\right)^{k+1/\rho}$.
- Has query time $O\left(\frac{k+1/\rho}{\varepsilon}\right)^{k+1/\rho}$ and $\tilde{O}(n^{1+1/(2^k-1)})$ work.
- The preprocessing time is $\left(\frac{\log n}{\varepsilon}\right)^{O(k+1/\rho)}$ and work $\tilde{O}(|E| \cdot n^\rho)$.

References

- 1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017. doi:10.1145/3088511.
- 2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 568–576, 2017. doi:10.1137/1.9781611974782.36.
- 3 Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Stephen Kobourov, and Richard Spence. Weighted additive spanners, 2020. arXiv:2002.07152.
- 4 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.
- 5 Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 322–335. ACM, 2020. doi:10.1145/3357713.3384321.
- 6 Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS*, pages 591–602, 2006. doi:10.1109/FOCS.2006.29.
- 7 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (α, β) -spanners. *ACM Transactions on Algorithms*, 7(1):5, 2010. doi:10.1145/1868237.1868242.

- 8 Uri Ben-Levy and Merav Parter. New (α, β) spanners and hopsets. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1695–1714. SIAM, 2020. doi:10.1137/1.9781611975994.104.
- 9 Aaron Bernstein. Fully dynamic $(2 + \epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 693–702, 2009. doi:10.1109/FOCS.2009.16.
- 10 Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000. doi:10.1145/331605.331610.
- 11 Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001. doi:10.1006/jagm.2000.1117.
- 12 D. Coppersmith and M. Elkin. Sparse source-wise and pair-wise distance preservers. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 660–669, 2005.
- 13 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.
- 14 D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29:1740–1759, 2000.
- 15 M. Elkin. Computing almost shortest paths. In *Proc. 20th ACM Symp. on Principles of Distributed Computing*, pages 53–62, 2001.
- 16 M. Elkin. An unconditional lower bound on the time-approximation tradeoff of the minimum spanning tree problem. In *Proc. of the 36th ACM Symp. on Theory of Comput. (STOC 2004)*, pages 331–340, 2004.
- 17 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2019. doi:10.1145/3274651.
- 18 Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM J. Comput.*, 48(4):1436–1480, 2019. doi:10.1137/18M1166791.
- 19 Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019.*, pages 333–341, 2019. doi:10.1145/3323165.3323177.
- 20 Michael Elkin and Ofer Neiman. Centralized, parallel, and distributed multi-source shortest paths via hopsets and rectangular matrix multiplication. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPICs*, pages 27:1–27:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.STACS.2022.27.
- 21 Michael Elkin and David Peleg. $(1+\epsilon, \beta)$ -spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004. doi:10.1137/S0097539701393384.
- 22 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 805–821, 2015. doi:10.1137/1.9781611973730.55.
- 23 Michael Elkin and Jian Zhang. Efficient algorithms for constructing $(1+\epsilon, \beta)$ -spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006. doi:10.1007/s00446-005-0147-2.
- 24 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.
- 25 Stephan Friedrichs and Christoph Lenzen. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '16*, pages 455–466, New York, NY, USA, 2016. ACM. doi:10.1145/2935764.2935777.

- 26 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Inf. Comput.*, 134(2):103–139, 1997. doi:10.1006/inco.1997.2620.
- 27 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.
- 28 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 146–155, 2014. doi:10.1109/FOCS.2014.24.
- 29 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, STOC '16*, pages 489–498, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897638.
- 30 Shang-En Huang and Seth Pettie. Thorup-zwick emulators are universally optimal hopsets. *Inf. Process. Lett.*, 142:9–13, 2019. doi:10.1016/j.ipl.2018.10.001.
- 31 Philip N. Klein and Sairam Subramanian. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 259–270, 1993. doi:10.1109/SFCS.1993.366861.
- 32 Philip N. Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *J. Algorithms*, 25(2):205–220, 1997. doi:10.1006/jagm.1997.0888.
- 33 Jason Li. Faster parallel algorithm for approximate shortest path. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 308–321. ACM, 2020. doi:10.1145/3357713.3384268.
- 34 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '15*, pages 192–201, New York, NY, USA, 2015. ACM. doi:10.1145/2755573.2755574.
- 35 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573, 2014. doi:10.1145/2591796.2591850.
- 36 Seth Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1), 2009. doi:10.1145/1644015.1644022.
- 37 Seth Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3):147–166, 2010. doi:10.1007/s00446-009-0091-7.
- 38 Hanmao Shi and Thomas H. Spencer. Time-work tradeoffs of the single-source shortest paths problem. *J. Algorithms*, 30(1):19–32, 1999. doi:10.1006/jagm.1998.0968.
- 39 Thomas H. Spencer. Time-work tradeoffs for parallel algorithms. *J. ACM*, 44(5):742–778, 1997. doi:10.1145/265910.265923.
- 40 M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of the 33rd ACM Symp. on Theory of Computing*, pages 183–192, 2001.
- 41 M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. of Symp. on Discr. Algorithms*, pages 802–809, 2006.
- 42 Mikkel Thorup. Undirected single source shortest path in linear time. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 12–21. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646088.
- 43 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.

- 44 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.
- 45 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A $(3 + \varepsilon, \beta)$ -Hopset

Here we show that the set H of Section 2 serves as a $(3 + \varepsilon, \beta)$ -hopset, for all $0 < \varepsilon < 12$ simultaneously, with $\beta = 2^{O(k \cdot \log(1/\varepsilon))}$.

Denote by $d_G^{(t)}(u, v)$ the length of the shortest path between u, v in G that contains at most t edges. The following lemma bounds the number of hops and the stretch of the constructed hopset:

► **Lemma 12.** *Fix any $0 < \delta \leq 1/4$ and any $x, y \in V$. Then for every $0 \leq i \leq k - 1$, at least one of the following holds:*

1. $d_{G \cup H}^{(2 \cdot (1/\delta)^i - 1)}(x, y) \leq (3 + \frac{12\delta}{1-3\delta})d_G(x, y)$
2. $d_{G \cup H}^{(1)}(x, p_{i+1}(x)) \leq \frac{3}{1-3\delta}d_G(x, y)$

Proof. The proof is by induction on i . For the base case $i = 0$, if $y \in B(x)$, then the edge (x, y) was added to the hopset and the first item holds. If not, it means that $d_G(x, y) \geq d_G(x, p_1(x))$. Because every vertex is connected by a direct edge to all its pivots, the second item holds (since the coefficient of the right hand side is between 3 and 12 for $0 < \delta \leq 1/4$).

Assume the claim holds for i , and we will prove it holds for $i + 1$. Partition the shortest path between x and y into $J \leq 1/\delta$ segments $\{L_j = [u_j, v_j]\}_{j \in [J]}$ each of length at most $\delta \cdot d_G(x, y)$, and at most $(1/\delta - 1)$ edges $\{(v_j, u_{j+1})\}_{j \in [J]}$ between consecutive segments. We can use the following: setting $u_1 = x$, and for each $j \in [J]$, set v_j as the vertex in the shortest path between u_j and y which is farthest from u_j , but still $d_G(u_j, v_j) \leq \delta \cdot d_G(x, y)$. If $v_j \neq y$, set u_{j+1} as the vertex which follows v_j in the shortest path between x and y . Otherwise set $u_{j+1} = y$. This partition satisfies our requirement $J \leq 1/\delta$ because for every $j \in [J - 1]$, $d_G(u_j, u_{j+1}) > \delta \cdot d_G(x, y)$ (otherwise, we could have chosen v_j as u_{j+1}).

Next, apply the induction hypothesis for all the pairs (u_j, v_j) with parameter i . If for all the pairs (u_j, v_j) the first item holds, we can show that the first item holds for (x, y) with parameter $i + 1$. Consider the path from x to y which uses the guaranteed path in $G \cup H$ of the first item for all the pairs (u_j, v_j) , and the edges (v_j, u_{j+1}) . The number of hops in this path is bounded by $(1/\delta) \cdot (2(1/\delta)^i - 1) + (1/\delta - 1) \leq 2 \cdot (1/\delta)^{i+1} - 1$. The length of the path is bounded by (using the induction hypothesis on each pair):

$$\begin{aligned} d_{G \cup H}^{(2(1/\delta)^{i+1} - 1)}(x, y) &\leq \sum_{j \in [J]} (d_{G \cup H}^{(2(1/\delta)^i - 1)}(u_j, v_j) + d_G^{(1)}(v_j, u_{j+1})) \\ &\leq (3 + \frac{12\delta}{1-3\delta})d_G(x, y) . \end{aligned}$$

Otherwise, there exist at least one segment for which the first item doesn't hold. Let $l \in [J]$ be the smallest index so that only the second item holds for the pair (u_l, v_l) . By the induction hypothesis

$$d_{G \cup H}^{(1)}(u_l, p_{i+1}(u_l)) \leq \frac{3}{1-3\delta}d_G(u_l, v_l) \leq \frac{3\delta}{1-3\delta}d_G(x, y).$$

Since we added the edges $(x, p_{i+1}(x))$, $(y, p_{i+1}(y))$ to the hopset H , by the triangle inequality,

$$\begin{aligned} d_{G \cup H}^{(1)}(x, p_{i+1}(x)) &\leq d_G(x, u_i) + d_G(u_i, p_{i+1}(u_i)) \\ &\leq d_G(x, u_i) + \frac{3\delta}{1-3\delta} d_G(x, y), \end{aligned} \quad (9)$$

$$\begin{aligned} d_{G \cup H}^{(1)}(y, p_{i+1}(y)) &\leq d_G(y, u_i) + d_G(u_i, p_{i+1}(u_i)) \\ &\leq d_G(y, u_i) + \frac{3\delta}{1-3\delta} d_G(x, y). \end{aligned} \quad (10)$$

If the edge $(p_{i+1}(x), p_{i+1}(y))$ exists in H , its length can be bounded by

$$\begin{aligned} d_{G \cup H}^{(1)}(p_{i+1}(x), p_{i+1}(y)) \\ &\leq d_{G \cup H}^{(1)}(p_{i+1}(x), x) + d_G(x, y) + d_{G \cup H}^{(1)}(y, p_{i+1}(y)). \end{aligned} \quad (11)$$

Thus, the distance between x and y using 3 hops is

$$\begin{aligned} d_{G \cup H}^{(3)}(x, y) \\ &\leq d_{G \cup H}^{(1)}(x, p_{i+1}(x)) + d_{G \cup H}^{(1)}(p_{i+1}(x), p_{i+1}(y)) + d_{G \cup H}^{(1)}(p_{i+1}(y), y) \\ &\stackrel{(11)}{\leq} 2d_{G \cup H}^{(1)}(x, p_{i+1}(x)) + d_G(x, y) + 2d_{G \cup H}^{(1)}(p_{i+1}(y), y) \\ &\stackrel{(9)+(10)}{\leq} 2\left(d_G(x, u_i) + \frac{3\delta}{1-3\delta} d_G(x, y)\right) + d_G(x, y) + 2\left(d_G(y, u_i) \right. \\ &\quad \left. + \frac{3\delta}{1-3\delta} d_G(x, y)\right) \\ &\leq \left(3 + \frac{12\delta}{1-3\delta}\right) d_G(x, y), \end{aligned}$$

therefore the first item holds.

If $(p_{i+1}(x), p_{i+1}(y)) \notin H$, then we know that $d_{G \cup H}^{(1)}(p_{i+1}(x), p_{i+2}(p_{i+1}(x))) \leq d_G(p_{i+1}(x), p_{i+1}(y))$. We can bound the distance $d_{G \cup H}^{(1)}(x, p_{i+2}(x))$ using the triangle inequality:

$$\begin{aligned} d_{G \cup H}^{(1)}(x, p_{i+2}(x)) \\ &\leq d_G(x, p_{i+1}(x)) + d_G(p_{i+1}(x), p_{i+2}(p_{i+1}(x))) \\ &\stackrel{(11)}{\leq} 2d_G(p_{i+1}(x), x) + d_G(x, y) + d_G(y, p_{i+1}(y)) \\ &\leq 2\left(d_G(x, u_i) + \frac{3\delta}{1-3\delta} d_G(x, y)\right) + d_G(x, y) + d_G(y, u_i) + \frac{3\delta}{1-3\delta} d_G(x, y) \\ &\leq 3d_G(x, y) + \frac{9\delta}{1-3\delta} d_G(x, y) = \frac{3}{1-3\delta} d_G(x, y). \end{aligned}$$

Thus the second item holds. \blacktriangleleft

We conclude by summarizing the main result of this section.

► **Theorem 13.** *For any weighted graph $G = (V, E)$ on n vertices, and any $k \geq 1$, there exists H of size at most $O(kn + n^{1+1/(2^k-1)})$, which is a $(3 + \varepsilon, \beta)$ -hopset for any $0 < \varepsilon \leq 12$, with $\beta = 2(3 + 12/\varepsilon)^{k-1}$.*

Proof. Let $x, y \in V$. Apply lemma 12 for x, y with $\delta = \frac{\varepsilon}{12+3\varepsilon}$ and $i = k - 1$. Since $A_k = \emptyset$, the first item must hold:

$$d_{G \cup H}^{(2(3+12/\varepsilon)^{k-1})}(x, y) \leq (3 + \varepsilon) d_G(x, y). \quad \blacktriangleleft$$

► **Remark 14.** Note that at its lowest, the hopbound is $O(4^k)$, achieved with stretch 15.

B A $(3 + \varepsilon, \beta \cdot W)$ -Emulator

In this section we show that the same H constructed in Section 2 can also serve as a $(3 + \varepsilon, \beta \cdot W)$ emulator for weighted graphs, for all values of $0 < \varepsilon < 1$ simultaneously.

Let $G = (V, E)$ be a weighted graph and let $k \geq 1$ and $\Delta > 3$ be given parameters (think of $\Delta = 3 + O(1/\varepsilon)$). Fix a pair $x, y \in V$. Define $D_{-1} = 0$ and for any integer $i \geq 0$, let $D_i = W(x, y) \cdot \sum_{j=0}^i \Delta^j$. We can easily verify that

$$D_{i+1} = \Delta \cdot D_i + W(x, y). \quad (12)$$

► **Lemma 15.** *Let $0 \leq i \leq k$ and let $x, y \in V$ such that $d_G(x, y) \leq D_i$ and $d_H(x, p_i(x)) \leq \frac{2\Delta}{\Delta-3} D_{i-1}$. Define $m = \max\{\Delta D_{i-1}, d_G(x, y)\}$. Then at least one of the following holds:*

1. $d_H(x, y) \leq (3 + \frac{8}{\Delta-3})m$.
2. $d_H(x, p_{i+1}(x)) \leq \frac{2\Delta}{\Delta-3} D_i$.

Proof. The proof is by induction on i . For the base case $i = 0$, if $y \in B(x)$, the first item holds. Otherwise $d_H(x, p_1(x)) \leq d_G(x, y) \leq W(x, y) = D_0$, thus the second item holds.

Assume the claim holds for i and prove for $i + 1$. By the triangle inequality:

$$d_H(y, p_{i+1}(y)) \leq d_G(y, x) + d_G(x, p_{i+1}(x)). \quad (13)$$

If $p_{i+1}(y) \in B(p_{i+1}(x))$, we have

$$d_H(p_{i+1}(x), p_{i+1}(y)) \leq d_G(p_{i+1}(x), x) + d_G(x, y) + d_G(y, p_{i+1}(y)). \quad (14)$$

Thus, the distance between x and y is

$$\begin{aligned} d_H(x, y) &\leq d_H(x, p_{i+1}(x)) + d_H(p_{i+1}(x), p_{i+1}(y)) + d_H(p_{i+1}(y), y) \\ &\stackrel{(14)}{\leq} 2d_H(x, p_{i+1}(x)) + d_G(x, y) + 2d_H(p_{i+1}(y), y) \\ &\stackrel{(13)}{\leq} 2d_H(x, p_{i+1}(x)) + d_G(x, y) + 2(d_G(y, x) + d_G(x, p_{i+1}(x))) \\ &\leq 3d_G(x, y) + \frac{4 \cdot 2\Delta}{\Delta - 3} D_i \\ &\leq \left(3 + \frac{8}{\Delta - 3}\right) m, \end{aligned}$$

therefore the first item holds.

If $p_{i+1}(y) \notin B(p_{i+1}(x))$, then we know that

$$d_H(p_{i+1}(x), p_{i+2}(p_{i+1}(x))) \leq d_G(p_{i+1}(x), p_{i+1}(y)). \quad (15)$$

We can bound the distance $d_H(x, p_{i+2}(x))$ as follows.

$$\begin{aligned}
d_H(x, p_{i+2}(x)) &\leq d_G(x, p_{i+1}(x)) + d_G(p_{i+1}(x), p_{i+2}(p_{i+1}(x))) \\
&\stackrel{(15)}{\leq} 2d_G(p_{i+1}(x), x) + d_G(x, y) + d_G(y, p_{i+1}(y)) \\
&\stackrel{(13)}{\leq} 2d_H(x, p_{i+1}(x)) + d_G(x, y) + d_G(y, x) + d_G(x, p_{i+1}(x)) \\
&\leq 2d_G(x, y) + \frac{3 \cdot 2\Delta}{\Delta - 3} D_i \\
&\stackrel{(12)}{\leq} 2D_{i+1} + \frac{6}{\Delta - 3} D_{i+1} \\
&= \frac{2\Delta}{\Delta - 3} D_{i+1}.
\end{aligned}$$

Hence the second item holds. ◀

We are now ready to state the result of this section.

► **Theorem 16.** *For any weighted graph $G = (V, E)$ on n vertices, and any $k \geq 1$, there exists H of size at most $O(kn + n^{1+1/(2^k-1)})$, which is a $(3 + \varepsilon, \beta \cdot W)$ -emulator for any $\varepsilon > 0$ with $\beta = O(1 + 1/\varepsilon)^{k-1}$.*

Proof. Let $x, y \in V$. Recall that P_{xy} is the shortest path between x and y in G , and fix $\Delta = 3 + 8/\varepsilon$. Initialize $i = 0$. Let z be the farthest vertex from x in P_{xy} satisfying $d_G(x, z) \leq D_i$. Note the requirement $d_H(x, p_0(x)) \leq \frac{2\Delta}{\Delta-3} D_{-1} = 0$ holds since $p_0(x) = x$. Apply lemma 15 on x, z and i . If the second item holds, we increase i by one, update z to be the last vertex in $P(x, y)$ satisfying $d_G(x, z) \leq D_i$, and apply the lemma again for x, z and i (since the second item held for $i - 1$, we have that $d_H(x, p_i(x)) \leq \frac{2\Delta}{\Delta-3} D_{i-1}$ indeed holds).

Consider now the index i such that the first item holds (we must find such an index, since at $i = k - 1$ there is no pivot in level k). If it is the case that $d_G(x, y) \geq D_i$ then since $D_i - \Delta D_{i-1} = W(x, y)$, it must be that $d_G(x, z) \geq \Delta D_{i-1}$, as otherwise we could have taken a further away z (recall that every edge on this path has weight at most $W(x, y)$). Therefore $m = d_G(x, z)$ and we found a path in H from x to z with stretch at most $3 + \frac{8}{\Delta-3}$. Next we update $x = z$, $i = 0$ and repeat the same procedure all over again.

The last remaining case is that we found an index i such that the first item holds but $d_G(x, y) < D_i$. Note that in such a case it must be that $z = y$. The path in H we have from x to y is of length at most $(3 + \frac{8}{\Delta-3}) \cdot D_i = (3 + \varepsilon) \cdot D_i$. As $i \leq k - 1$ and $D_{k-1} \leq 2\Delta^{k-1} \cdot W(x, y)$, we have that

$$d_H(x, y) \leq 2(3 + \varepsilon) \cdot (3 + 8/\varepsilon)^{k-1} \cdot W(x, y),$$

which is our additive stretch β . ◀

C Full proof of Lemma 1

If we order the vertices in A_i by their distance to u , it is easy to see that the number of vertices which are in A_i and closer than $p_{i+1}(u)$ is bounded by a random variable sampled from a geometric distribution with parameter q_i . Hence $E[|B(u)|] \leq k + 1/q_i = k + n^{2^i} 2^{2^i+1}$. For $u \in A_{k-1}$, since $p_k(u)$ doesn't exist, $B(u)$ contains all the vertices in A_{k-1} . The number of vertices in A_{k-1} is a random variable sampled from binomial Distribution with parameters $(n, \prod_{j=0}^{k-2} q_j) = (n, n^{-(2^{k-1}-1)\nu} \cdot 2^{-2^{k-1}-k+2})$. Hence, the expected number of edges added by bunches of vertices in A_{k-1} is

$$\begin{aligned}
 E \left[\binom{|A_{k-1}|}{2} \right] &\leq E[|A_{k-1}|^2] = E[|A_{k-1}|]^2 + \text{Var}(|A_{k-1}|) \\
 &= n^2 \prod_{j=0}^{k-2} q_j^2 + n(1 - \prod_{j=0}^{k-2} q_j) \prod_{j=0}^{k-2} q_j \\
 &\leq n^{2-2(2^{k-1}-1)\nu} \cdot 2^{2(-2^{k-1}-k+2)} + n \cdot 2^{-2^{k-1}-k+2} \\
 &\leq (n^{1+\nu} + n)2^{3-k}.
 \end{aligned}$$

Hence, the total expected number of edges in H is:

$$\begin{aligned}
 &\sum_{i=0}^{k-2} (N_i \cdot n^{2^i\nu} \cdot 2^{2^i+1}) + E[|A_{k-1}|^2] + kn \\
 &= \sum_{i=0}^{k-2} (n^{1+\nu} \cdot 2^{-i+2}) + E[|A_{k-1}|^2] + kn \\
 &= O(kn + n^{1+\nu})
 \end{aligned}$$

D Proofs of Theorems 6,7

D.1 Size analysis

Recall the construction of H described in Section 4. Define the bunch $B(u)$ as in (2). The following lemma will be useful to bound the size of the spanner H .

► **Lemma 17.** *Fix $0 \leq i \leq k-1$. Let $u, v, x, y \in A_i$ be such that $v \in B_{1/2}(u)$ and $y \in B_{1/2}(x)$, and $P_{uv} \cap P_{xy} \neq \emptyset$, then all four points are in $B(u)$, or all four are in $B(x)$.*

Proof. Assume w.l.o.g. that P_{uv} is not shorter than P_{xy} . Let $z \in V$ be a point in the intersection of the two shortest paths, then

$$\begin{aligned}
 d_G(u, x) &\leq d_G(u, z) + d_G(z, x) \leq d_G(u, v) + d_G(y, x) \\
 &\leq 2d_G(u, v) < d_G(u, A_{i+1}),
 \end{aligned}$$

so $x \in B(u)$. The calculation showing $y \in B(u)$ is essentially the same. ◀

Define the shortest path between two vertices consistently, s.t. each subpath is also a shortest path. Therefore two shortest paths can have at most one common subpath.

Fix $0 \leq i \leq k-2$, and consider the graph G_i containing all the shortest paths P_{uv} with $u \in A_i$ and $v \in B_{1/2}(u)$. We claim that the number of edges in G_i is at most $O(n + C_i)$, where C_i is the number of pairwise intersections between these shortest paths. This is because vertices participating in at most 1 path have degree at most 2, and each intersection increases the degree of one vertex by at most 2 (recall that shortest paths can meet at most once).

▷ **Claim 18.** $\mathbb{E}[|C_i|] \leq O(n^{1+\nu})$.

Proof. By Lemma 17 each intersecting pair of paths P_{uv} and P_{xy} we have that all four points belong to the same bunch. Thus, each $u \in A_i$ can introduce at most $|B(u)|^3$ pairwise intersecting paths. Recall that $|B(u)|$ is a random variable distributed geometrically with parameter q_i , so

$$\begin{aligned} \mathbb{E}[|B(u)|^3] &= \sum_{j=1}^{\infty} j^3 \cdot q_i \cdot (1 - q_i)^{j-1} \\ &\leq q_i \cdot \sum_{j=1}^{\infty} (1 - q_i)^{j-1} \cdot j(j+1)(j+2) \leq \frac{6}{q_i^3}. \end{aligned}$$

Thus the expected number of intersections at level i is at most

$$\begin{aligned} \mathbb{E} \left[\sum_{u \in A_i} |B(u)|^3 \right] &\leq O(N_i/q_i^3) = O(n^{1-((4/3)^i-1)\nu} \cdot (n^{4^i\nu/3^{i+1}})^3) \\ &= O(n^{1+\nu}). \end{aligned}$$

It remains to bound path intersections in the last level $k - 1$. Recall that

$$N_{k-1} = n^{1-((4/3)^{k-1}-1)\nu} = n^{1-((4/3)^{k-1}-1)/((4/3)^k-1)} \leq n^{(1+\nu)/4}.$$

Since the random choices for each point are independent, we have by Chernoff bound that $\Pr[N_{k-1} > 2n^{(1+\nu)/4}] \leq e^{-\Omega(n^{(1+\nu)/4})}$, so with very high probability the last set A_{k-1} contains $O(n^{(1+\nu)/4})$ points. It means that we have $O(\sqrt{n^{1+\nu}})$ paths connecting these points, and even if they all intersect, they can yield at most $O(n^{1+\nu})$ intersections. \triangleleft

It remains to bound the number of edges to pivots. For every $v \in V$, $p_i(v)$ is the closest vertex to v in A_i breaking ties by id. Therefore all the vertices in $P_{v,p_i(v)}$ share the same pivot at level i . Thus we add at most one edge for each vertex at every level, and $O(nk)$ edges overall.

We conclude that the size of H is at most $O(k \cdot n^{1+\nu})$ (we can slightly change the probabilities by introducing a factor of $2^{-4^i/3^{i+1}-1}$ to obtain size $O(kn + n^{1+\nu})$, as we did before.

D.2 Proof of Theorem 6

We use the corresponding analysis of the emulator from Section 3. The use of half-bunches instead of bunches creates the following version of Lemma 2.

► **Lemma 19.** *Fix $\Delta > 5$. Let $0 \leq i < k$ and let $x, y \in V$ such that $d_G(x, y) \geq (3\Delta)^i W(x, y)$. Then at least one of the following holds:*

1. $d_H(x, y) \leq (1 + \frac{8i}{\Delta-5})d_G(x, y)$
2. $d_H(x, p_{i+1}(x)) \leq \frac{2\Delta}{\Delta-5}d_G(x, y)$

The main difference in the proof is in (7), which is replaced by

$$d_H(p_{i+1}(u_l), p_{i+2}(p_{i+1}(u_l))) \leq 2d_G(p_{i+1}(u_l), p_{i+1}(u_r)).$$

The new bounds in the Lemma guarantee the calculations still go through. For Lemma 3 which takes care of small distances, we have the following change, with a very similar proof.

► **Lemma 20.** *Let $0 \leq i < k$ and fix $x, y \in V$. Let*

$m = \max\{d_G(x, p_i(x)), d_G(y, p_i(y)), d_G(x, y)\}$. *Then at least one of the following holds:*

1. $d_H(x, y) \leq 5m$
2. $d_H(x, p_{i+1}(x)) \leq 7m$

In the proof we set $\Delta = 5 + \frac{8(k-1)}{\varepsilon}$. The rest of the calculations follow analogously, one change is that when iteratively applying Lemma 20, the bound m increases by a factor of 7 (rather than 4, as in Lemma 3), but as $7 \leq 3\Delta$ is still true, it does not change anything.

D.3 Proof of Theorem 7

The stretch analysis is very similar to that of the emulator from Section B, the main difference is in the use of half-bunches rather than the full ones, but this will increase the distance to pivots by a factor of 2, and affect the additive stretch only. We follow the analysis and notation presented in Section B, but with $\Delta > 5$. We replace Lemma 15 with the following.

► **Lemma 21.** *Let $0 \leq i \leq k$ and let $x, y \in V$ such that $d_G(x, y) \leq D_i$ and $d_H(x, p_i(x)) \leq \frac{3\Delta}{\Delta-5}D_{i-1}$. Define $m = \max\{\Delta D_{i-1}, d_G(x, y)\}$. Then at least one of the following holds:*

1. $d_H(x, y) \leq (3 + \frac{16}{\Delta-5})m$.
2. $d_H(x, p_{i+1}(x)) \leq \frac{4\Delta}{\Delta-5}D_i$.

The main difference in the proof is in (15), which is replaced by

$$d_H(p_{i+1}(x), p_{i+2}(p_{i+1}(x))) \leq 2d_G(p_{i+1}(x), p_{i+1}(y)) ,$$

since we use half-bunches. One can then follow the calculations in the proof of Lemma 15, and check that the altered constants used in the 2 cases above suffice.

The proof of Theorem 7 is the same as the proof of Theorem 16, the only differences are taking $\Delta = 5 + \frac{16}{\epsilon}$ (which affects the value of β) and using the bounds of Lemma 21 rather than of Lemma 15.

Complexity of Finding Maximum Locally Irregular Induced Subgraphs

Foivos Fioravantes ✉

Université Côte d'Azur, Inria, CNRS, I3S, Valbonne, France

Nikolaos Melissinos ✉

Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Theofilos Triommatis ✉

School of Electrical Engineering, Electronics and Computer Science, University of Liverpool, UK

Abstract

If a graph G is such that no two adjacent vertices of G have the same degree, we say that G is *locally irregular*. In this work we introduce and study the problem of identifying a largest induced subgraph of a given graph G that is locally irregular. Equivalently, given a graph G , find a subset S of $V(G)$ with minimum order, such that by deleting the vertices of S from G results in a locally irregular graph; we denote with $I(G)$ the order of such a set S . We first examine some easy graph families, namely paths, cycles, trees, complete bipartite and complete graphs. However, we show that the decision version of the introduced problem is \mathcal{NP} -Complete, even for restricted families of graphs, such as subcubic planar bipartite, or cubic bipartite graphs. We then show that we can not even approximate an optimal solution within a ratio of $\mathcal{O}(n^{1-\frac{1}{k}})$, where $k \geq 1$ and n is the order of the graph, unless $\mathcal{P}=\mathcal{NP}$, even when the input graph is bipartite.

Then, looking for more positive results, we turn our attention towards computing $I(G)$ through the lens of parameterised complexity. In particular, we provide two algorithms that compute $I(G)$, each one considering different parameters. The first one considers the size of the solution k and the maximum degree Δ of G with running time $(2\Delta)^k n^{\mathcal{O}(1)}$, while the second one considers the treewidth tw and Δ of G , and has running time $\Delta^{2tw} n^{\mathcal{O}(1)}$. Therefore, we show that the problem is FPT by both k and tw if the graph has bounded maximum degree Δ . Since these algorithms are not FPT for graphs with unbounded maximum degree (unless we consider $\Delta + k$ or $\Delta + tw$ as the parameter), it is natural to wonder if there exists an algorithm that does not include additional parameters (other than k or tw) in its dependency.

We answer negatively, to this question, by showing that our algorithms are essentially optimal. In particular, we prove that there is no algorithm that computes $I(G)$ with dependence $f(k)n^{\mathcal{O}(k)}$ or $f(tw)n^{\mathcal{O}(tw)}$, unless the ETH fails.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases Locally irregular, largest induced subgraph, FPT, treewidth, W-hardness, approximability

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.24

Related Version *Full Version*: <https://hal.archives-ouvertes.fr/hal-03358273>

Funding *Foivos Fioravantes*: Supported by the French government through the UCA JEDI (ANR-15-IDEX-01) and the EUR DS4H (ANR-17-EURE-004) Investments in the Future projects.

Theofilos Triommatis: Supported by EP/S023445/1 EPSRC CDT in Distributed Algorithms, University of Liverpool.



© Foivos Fioravantes, Nikolaos Melissinos, and Theofilos Triommatis;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 24; pp. 24:1–24:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A graph G is said to be *locally irregular*, if every two adjacent vertices of G have different degrees. In this paper, we introduce and study the problem of finding a largest locally irregular induced subgraph of a given graph. This problem is equivalent to identifying what is the minimum number of vertices that must be deleted from G , so that what remains is a locally irregular graph.

Locally irregular graphs. The notion of locally irregular graphs was first introduced in [6]. The most interesting aspect of locally irregular graphs, comes from their connection to the so-called 1-2-3 Conjecture, proposed in [22]. Formally, the 1-2-3 Conjecture states that for almost every graph, we should be able to place weights from $\{1, 2, 3\}$ on the edges of that graph, so that the colouring, that assigns a colour to each vertex equal to the sum of the weights on its adjacent edges, is a proper vertex-colouring of the graph.

As we said earlier, the 1-2-3 Conjecture seems to have some very interesting links to locally irregular graphs. An obvious connection is that this conjecture holds for locally irregular graphs. Indeed, placing weight equal to 1 to all the edges of a locally irregular graph, suffices to produce a proper vertex-colouring, as each vertex receives a colour equal to its degree. Furthermore, there have been some steps towards proving that conjecture, which involve edge-decomposing a graph into a constant number of locally irregular subgraphs, i.e., given G , find an edge-colouring of G using a constant number of colours, such that each colour induces a locally irregular subgraph of G . This is the main motivation behind [6], and it seems to remain interesting enough to attract more attention [8, 25, 30].

Note that the class of locally irregular graphs can be seen as an antonym to that of *regular* graphs, i.e., graphs such that all of their vertices have the same degree. It is important to state here that there exist several alternative such notions. This is mainly due to the very well known fact that there are no non-trivial *irregular* graphs, i.e., graphs that do not contain two vertices (not necessarily adjacent) with the same degree (see [12]). Thus, the literature has plenty of slightly different definitions of irregularity (see for example [2, 12, 13, 20, 29]). One way to deal with the nonexistence of irregular graphs, is to define a notion of *local* irregularity. Intuitively, instead of demanding for all vertices of a graph to have different degrees, we are now considering each vertex v separately, and request that the vertices “around” v to verify some properties of irregularity. For example, the authors of [3] study graphs G such that for every vertex v of G , no two neighbours of v have the same degree. For an overview of other interesting notions of irregularity (local or otherwise), we refer the reader to [4].

Largest induced subgraph verifying specific properties. The problem we introduce belongs in a more general and well studied family of problems, which is about identifying a largest induced subgraph of a given graph that verifies a specific property Π . That is, given a graph $G = (V, E)$ and an integer k , is there a set $V' \subseteq V$ such that $|V'| \leq k$ and $G[V \setminus V']$ has the specified property Π ? In our case, the property Π is “the induced subgraph is locally irregular”. This generalised problem is indeed classic in graph theory, and it is known as the INDUCED SUBGRAPH WITH PROPERTY Π (ISPII for short) problem in [21]. Unfortunately, it was shown in [24], that ISPII is a hard problem for any property Π that is *hereditary*, i.e., all induced subgraphs of G verify Π if G itself verifies that property.

However, the ISPII problem remains interesting (one could say that it actually becomes more interesting) even if the property Π is not hereditary. Recently, the authors of [7] studied the problem for Π being “all vertices of the induced subgraph have odd degree”, which

clearly is not a hereditary property. Nevertheless, they showed that this is an \mathcal{NP} -hard problem, and they gave an FPT algorithm that solves the problem when parameterised by the rank-width. Also, the authors of [1, 5, 28] studied the ISPII problem, where Π is the rather natural property “the induced subgraph is d -regular”, where d is an integer given in the input (recall that a graph is said to be d -regular if all of its vertices have the same degree d). In particular, in [5] it is shown that finding a largest (connected) induced subgraph that is d -regular, is \mathcal{NP} -hard to approximate, even when restricted on bipartite or planar graphs. The authors of [5] also provide a linear-time algorithm to solve this problem for graphs with bounded treewidth. In contrast, the authors of [1] take a more practical approach, as they focus on solving the problem for the particular values of $d = 1$ and $d = 2$, by using bounds from quadratic programming, Lagrangian relaxation and integer programming.

It is quite clear that, in some sense, the property that interests us lies on the opposite side of the one studied in [1, 5, 28]. However, both properties, “the induced subgraph is regular” and “the induced subgraph is locally irregular” are not hereditary. This means that we do not get an \mathcal{NP} -hardness result directly from [24]. Furthermore, the ISPII problem always admits an FPT algorithm, when parameterised by the size of the solution, if Π is a hereditary property (proven in [11, 23]), but for a non-hereditary one, this is not always true. Indeed in [28], the authors proved that when considering Π as “the induced subgraph is regular”, the ISPII problem is W[1]-hard when parameterised by the size of the solution. That is, there should be no $f(k)n^c$ time algorithm for this problem, where c is a constant. For such problems, it is also interesting to see if there exists any algorithm with running time $n^{o(k)}$ or $f(k)n^{o(k)}$. The authors of [14, 15, 16] provide techniques that can be used to strongly indicate the non-existence of such algorithms, by applying them on a variety of W[1]-hard and W[2]-hard problems, such as the INDEPENDENT SET and the DOMINATING SET, parameterised by the size of their solutions. Usually these lower bounds are shown under the assumption of a weaker version of the EXPONENTIAL TIME HYPOTHESIS, which states that SAT can not be solved in time $2^{o(n+m)}$.

Our contribution. We begin in Section 2 by providing the basic notations and definitions that are going to be used throughout this paper. In Section 3, we deal with the complexity of the introduced problem. In particular, we show that the problem belongs in \mathcal{P} if the input graph is a path, cycle, tree, complete bipartite or complete graph. We then prove that finding the maximum induced locally irregular subgraph of a given graph G is \mathcal{NP} -hard, even if G is restricted to being a subcubic planar bipartite, or a cubic bipartite graph.

As the introduced problem seems to be computationally hard even for restricted families of graphs, we then investigate its approximability. Unfortunately, we prove in Section 4 that for any bipartite graph G of order n and $k \geq 1$, there can be no polynomial time algorithm that finds an approximation of $I(G)$ within ratio $\mathcal{O}(n^{1-\frac{1}{k}})$, unless $\mathcal{P}=\mathcal{NP}$. Nevertheless, we do provide a (simple) d -approximation algorithm for d -regular bipartite graphs.

We then decide to look into its parameterised complexity. In Section 5, we present two algorithms that compute $I(G)$, each one considering different parameters. The first considers the size of the solution k and the maximum degree Δ of G , and has running time $(2\Delta)^k n^{\mathcal{O}(1)}$, while the second considers the treewidth tw and Δ of G , and has running time $\Delta^{2tw} n^{\mathcal{O}(1)}$. Unfortunately, these algorithms can be considered as being FPT only if Δ is part of the parameter. In Section 5.1, we present two linear fpt-reductions which prove that the problem is W[2]-hard when parameterised only by the size of the solution and W[1]-hard when parameterised only by the treewidth. These reductions also show that we can not even have an algorithm that computes $I(G)$ in time $f(k)n^{o(k)}$ or $\mathcal{O}^*(f(tw)n^{o(tw)})$, unless the ETH fails. The \mathcal{O}^* notation is used to suppress polynomial factors in regards to n and tw .

2 Preliminaries

For notions and definitions on graph theory not explained here, we refer the reader to [18].

Let $G = (V, E)$ be a graph and $G' = (V', E')$ be a subgraph of G (*i.e.*, created by deleting vertices and/or edges of G). Recall first that the subgraph G' is *induced* if it can be created only by deleting vertices of G . That is, for each edge $uv \in E$, if $u, v \in V'$, then $uv \in E'$. For any vertex $v \in V$, let $N_G(v) = \{u \in V : uv \in E\}$ denote the *neighbourhood* of v in G , and let $d_G(v) = |N_G(v)|$ denote the *degree* of v in G . We also define $N_G[v] = N_G(v) \cup \{v\}$. Finally, for any $X \subseteq V$, we define $N_G[X] = \bigcup_{v \in X} N_G[v]$. Note that, whenever the graph G is clear from the context, we will omit the subscript and simply write $N(v)$, $d(v)$, $N[v]$ and $N[X]$.

One way to show that a problem can not be approximated within a certain ratio, is through a *gap reduction*. The goal of such a reduction is to show that it is \mathcal{NP} -hard to differentiate between instances that have a solution of size $\leq \alpha$ and those for which any solution has size $> \beta$. If such is the case, then we know that we cannot approximate the optimal solution within a ratio of $\frac{\beta}{\alpha}$, as otherwise we would get that $\mathcal{P} = \mathcal{NP}$.

Finally, recall that a *fixed parameter-tractable* (FPT for short) algorithm, is an algorithm with running time $f(k)n^{\mathcal{O}(1)}$, where f is a computable function and k is the considered parameter. We also make use of what is known as a *linear fpt-reduction*, a type of polynomial reduction such that the size of the parameter of the new problem is linear in regards to the size of the parameter of the original problem. Observe that if we have a linear fpt-reduction from a problem Q with parameter k to a problem Q' with parameter k' and the assumption that Q can not be solved in time $f(k)n_1^{\mathcal{O}(k)}$ (where n_1 is the size of the input of Q), then we can conclude that there is no $f(k')n_2^{\mathcal{O}(k')}$ time algorithm for Q (where n_2 is the size of the input of Q).

Let $G = (V, E)$ be a graph. We say that G is *locally irregular* if for every edge $uv \in E$, we have $d(u) \neq d(v)$. Now, let $S \subseteq V$ be such that $G[V \setminus S]$ is a locally irregular graph; any set S that has this property is said to be an *irregularator* of G . For short, we will say that S is an *ir*(G). Moreover, let $I(G)$ be the minimum order that any *ir*(G) can have. We will say that S is a *minimum* irregularator of G , for short S is an *ir**(G), if S is an *ir*(G) and $|S| = I(G)$.

We also define the following notion, which generalises *ir*(G). Let $G = (V, E)$ be a graph, $S, X \subseteq V$ and let $G' = G[V \setminus S]$. Now, let $S \subseteq V$ be such that, for each two neighbouring vertices u, v in $X \setminus S$, we have that $d_{G'}(u) \neq d_{G'}(v)$; any set S that has this property is said to be an *irregularator of X in G* , for short *ir*(G, X). We define the notions of *ir**(G, X) and $I(G, X)$ analogously to the previous definitions.

We will now provide some lemmas and an observation that will be useful throughout this paper. As the proofs of the following lemmas mainly follow from the definitions, we chose to only include them in the full version of this paper. In the three lemmas below, we investigate the relationship between $I(G)$ and $I(G, X)$.

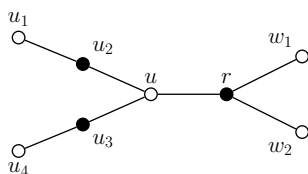
► **Lemma 1.** *Let $G = (V, E)$ be a graph and let $X \subseteq V$. Then $I(G, X) \leq I(G)$.*

► **Lemma 2.** *Let $G = (V, E)$ be a graph and $S, X \subseteq V$ such that S is an *ir**(G, X). Then, $S \subseteq N[X]$ and $I(G, X) = I(G[N[X]], X)$.*

► **Lemma 3.** *Let $G = (V, E)$ be a graph, and $X_1, \dots, X_n \subseteq V$ such that $N[X_i] \cap N[X_j] = \emptyset$ for every $1 \leq i < j \leq n$. Then $\sum_{i=1}^n I(G, X_i) \leq I(G)$.*

► **Lemma 4.** *Let $G = (V, E)$ be a graph, X be a subset of V and S be an *ir*(G). The set $S \cap N[X]$ is an *ir*(G, X) and an *ir*($G[N[X]], X$).*

The following, almost trivial, observation, will be useful throughout the rest of the paper.



■ **Figure 1** The gadget used in the proof of Theorem 7. The white and black vertices are used to denote vertices belonging to different bipartitions.

► **Observation 5.** *Let $G = (V, E)$ be a graph and S be an $ir(G)$. Then, for each edge $uv \in E$, if $d(u) = d(v)$, then S contains at least one vertex in $N[\{u, v\}]$. Additionally, for a set $X \subseteq V$, let S^* be an $ir(G[N[X]], X)$. Then for each edge $uv \in E(G[X])$, if $d(u) = d(v)$, then S^* contains at least one vertex in $N[\{u, v\}]$.*

3 (Classic) complexity

In this section, we deal with the complexity of the problem we introduced. In the following theorem, we sum up all the families of graphs for which we prove that $I(G)$ is computed in polynomial time.

► **Theorem 6.** *Let G be a graph. If G is a path, cycle, tree, complete bipartite or a complete graph, then the problem of computing $I(G)$ is in \mathcal{P} .*

The result for the case of paths and cycles is proven through induction on the order of the graph. Then, complete and complete bipartite graphs have a rather trivial structure in regards to the problem studied here. Finally, the polynomial algorithm for trees follows directly from upcoming Theorem 14.

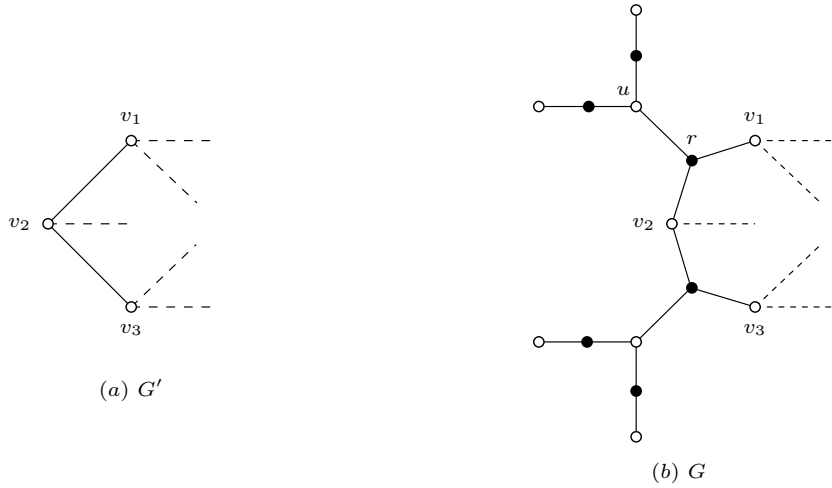
3.1 \mathcal{NP} -Hard Cases

We now show that finding a minimum irregulator of a graph is \mathcal{NP} -hard. Interestingly, this remains true even for quite restricted families of graphs, such as cubic (i. e., 3-regular) bipartite, and subcubic planar bipartite graphs, i. e., planar bipartite graphs of maximum degree at most 3.

► **Theorem 7.** *Let G be graph and $k \in \mathbb{N}$. Deciding if $I(G) \leq k$ is \mathcal{NP} -complete, even when G is a planar bipartite graph with maximum degree $\Delta \leq 3$.*

Proof. Since the problem is clearly in \mathcal{NP} , we will focus on proving it is also \mathcal{NP} -hard. The reduction is from the VERTEX COVER problem, which remains \mathcal{NP} -complete when restricted to planar cubic graphs [27]. In that problem, a planar cubic graph G and an integer $k \geq 1$ are given as an input. The question is, whether there exists a vertex cover of G of order at most k . That is, whether there exists a set $VC \subseteq V(G)$ such that for every edge $uv \in E(G)$, at least one of u and v belongs in VC and $|VC| \leq k$.

Let G' be a planar cubic graph and $k \geq 1$ given as input for VERTEX COVER. Let $|E(G')| = m$. We will construct a planar bipartite graph G as follows; we start with the graph G' , and modify it by using multiple copies of the gadget, illustrated in Figure 1. Note that we will be following the naming convention illustrated in Figure 1 whenever we talk about the vertices of our gadgets. When we say that we *attach* a copy H of the gadget to the vertices v and v' of G' , we mean that we add H to G' , and we identify the vertices w_1 and w_2 to the vertices v and v' respectively. Now, for each edge $vv' \in E(G')$, attach one



■ **Figure 2** The construction in the proof of Theorem 7. The graph G' is the initial planar cubic graph, and G is the graph built during our reduction. In G , the white and black vertices are used to denote vertices belonging to different bipartitions.

copy H of the gadget to the vertices v and v' , and then delete the edge vv' (see Figure 2). Clearly this construction is achieved in linear time (we have added m copies of the gadget). Note also that the resulting graph G has $\Delta(G) = 3$ and that the planarity of G' is preserved since G is constructed by essentially subdividing the edges of G' and adding a tree pending from each new vertex. Also, G is bipartite. Indeed, observe that after removing the edges of $E(G')$, the vertices of $V(G')$ form an independent set of G . Furthermore, the gadget is bipartite, and the vertices w_1, w_2 (that have been identified with vertices of $V(G')$) belong to the same bipartition (in the gadget). Finally, for any $1 \leq i \leq m$, let H_i be the i^{th} copy of the gadget attached to vertices of G' . We will also be using the vertices r^i and u^i to denote the copies of the vertices r and u (respectively) that also belong to H_i .

We are now ready to show that the minimum vertex cover of G' has size k' if and only if $I(G) = k'$.

Let VC be a minimum vertex cover of G' and $|VC| = k'$. We will show that the set $S = VC$ is an $ir(G)$. Let $G^* = G[V(G) \setminus S]$. First, note that S contains only vertices of G' . Thus, for each i , the vertices of H_i except from r^i , which also remain in G^* , have the same degree in G' and in G^* . Also note that each vertex of G' is adjacent only to copies of r . It follows that it suffices to only consider the vertices r^i to show that VC is an $ir(G)$. Now, for any $1 \leq i \leq m$, consider the vertex r^i . Since VC is a vertex cover of G' , for each edge $vv' \in E(G')$, VC contains at least one of v and v' . It follows that $d_{G^*}(r^i) \leq 2$. Note also that $N_{G^*}(r^i)$ contains the vertex $u^i \in V(H_i)$ and possibly one vertex $v \in V(G')$.

Also, since we only delete vertices in $V(H_i) \cap V(G')$, we have that $d_{G^*}(u^i) = 3 > d_{G^*}(r^i)$. In the case where $N_{G^*}(r^i)$ also contains a vertex $v \in V(G')$, the vertex v is adjacent only to vertices which do not belong in $V(G')$. Thus, $d_{G^*}(v) = d_G(v) = 3 > d_{G^*}(r^i)$. It follows that r^i has a different degree from all of its neighbours and that VC is an $ir(G)$.

Now, we prove that if $I(G) = k'$ then there exists a vertex cover of size at most k' . Assume that $I(G) = k'$ and let S be an $ir^*(G)$. Observe that since S is an $ir^*(G)$, S contains at least one vertex of H_i (for each $1 \leq i \leq m$). Let $X_i = V(H_i) \cap V(G')$. To construct a vertex cover VC of G' with $|VC| \leq k'$, we work as follows. For each $1 \leq i \leq m$:

1. for each vertex $v \in X_i$, if $v \in S$ then put v in VC . Then,
2. if $S \cap X_i = \emptyset$, put any one of the two vertices of X_i in VC .

Observe now that any vertex that is added to VC during step 1. of the above procedure, also belongs to S and any vertex that is added during step 2. of the above procedure corresponds to at least one vertex in S . It follows that $|VC| \leq k'$. Also note that VC contains at least one vertex of X_i , for each i , and that for each $uv \in E(G')$, there exists an i such that $V(X_i) = \{u, v\}$. Thus VC is indeed a vertex cover of G' .

Therefore G' has a minimum vertex cover of size k' if and only if $I(G) = k'$. To complete the proof note that deciding if $I(G) = k' < k$ for a given k , answers the question whether G' has a vertex cover of size less than k or not. ◀

In the following theorem we show that calculating $I(G)$ is \mathcal{NP} -hard even if G is a cubic bipartite graph.

► **Theorem 8.** *Let G be graph and $k \in \mathbb{N}$. Deciding if $I(G) \leq k$ is \mathcal{NP} -complete even in cubic bipartite graphs.*

This theorem is shown through a reduction from the 2-BALANCED 3-SAT, which was proven to be \mathcal{NP} -complete in [9].

4 (In)approximability

In the previous section we showed that computing $I(G)$ is \mathcal{NP} -hard, even for graphs G belonging to quite restricted families of graphs. So the natural question to pose next, which we investigate in this section, is whether we can approximate $I(G)$. Unfortunately, most of the results we present below are once again negative.

We start with a corollary that follows from the proof of Theorem 7 and the inapproximability of VERTEX COVER in cubic graphs [17]:

► **Corollary 9.** *Given a graph G , it is \mathcal{NP} -hard to approximate $I(G)$ to within a ratio of $\frac{100}{99}$, even if G is bipartite and $\Delta(G) = 3$.*

Now, we are going to show that there can be no algorithm that approximates $I(G)$ to within any decent ratio in polynomial time, unless $\mathcal{P} = \mathcal{NP}$, even if G is a bipartite graph (with no restriction on its maximum degree).

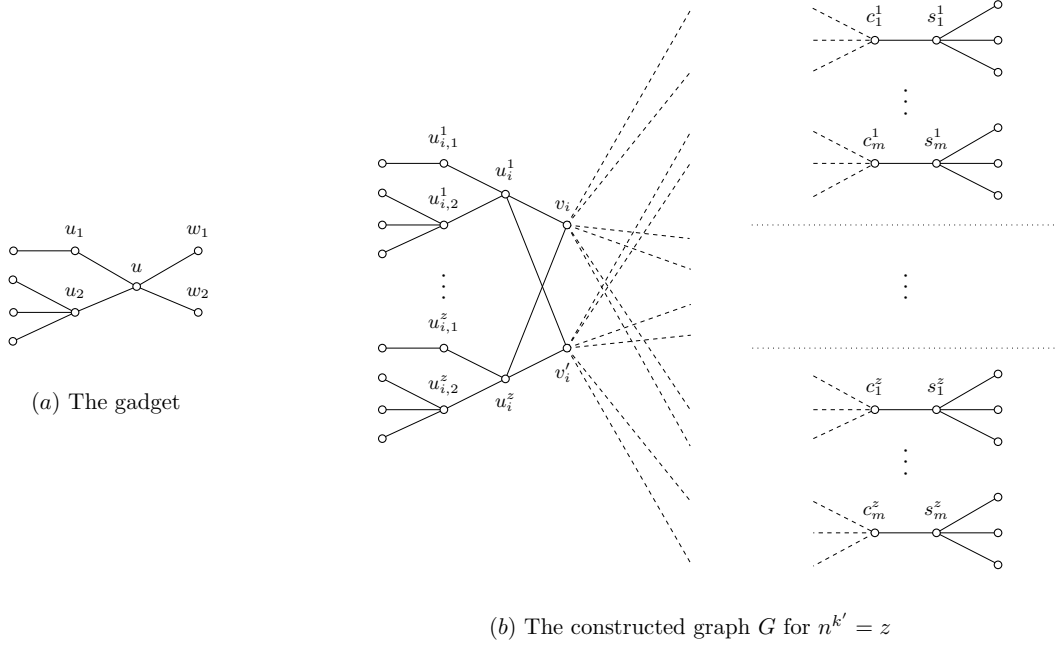
► **Theorem 10.** *Let G be a bipartite graph of order N and $k \in \mathbb{N}$ be a constant such that $k \geq 1$. It is \mathcal{NP} -hard to approximate $I(G)$ to within $\mathcal{O}(N^{1-\frac{1}{k}})$.*

Proof. The proof is by a *gap producing reduction* from 2-BALANCED 3-SAT, which was proven to be \mathcal{NP} -complete in [9]. In that problem, a 3CNF formula F is given as an input, comprised by a set C of clauses over a set of Boolean variables X . In particular, we have that each clause contains exactly 3 literals, and each variable $x \in X$ appears in F exactly twice as a positive and twice as a negative literal. The question is, whether there exists a truth assignment to the variables of X satisfying F .

Let F be a 3CNF formula with m clauses C_1, \dots, C_m and n variables x_1, \dots, x_n that is given as input to the 2-BALANCED 3-SAT problem. Let $2k = k' + 1$. Based on the instance F , we are going to construct a bipartite graph $G = (V, E)$ where $|V| = \mathcal{O}(n^{k'+1})$ and

- $I(G) \leq n$ if F is satisfiable
- $I(G) > n^{k'}$ otherwise.

To construct $G = (V, E)$, we start with the following graph: for each literal x_i ($\neg x_i$ resp.) in F , add a *literal vertex* v_i (v'_i resp.) in V , and for each clause C_j of F , add a *clause vertex* c_j in V . Next, for each $1 \leq j \leq m$, add the edge $v_i c_j$ ($v'_i c_j$ resp.) if the literal x_i ($\neg x_i$ resp.)



■ **Figure 3** The construction in the proof of Theorem 10. In subfigure (b), we illustrate how each pair of literal vertices is connected to the rest of the graph. Whenever there is an upper index $1 \leq l \leq n^{k'}$ on a vertex, it is used to denote the l^{th} copy of that vertex. The dashed lines are used to represent the edges between the literal and the clause vertices.

appears in C_j according to F . Observe that the resulting graph is bipartite, for each clause vertex c we have $d(c) = 3$ and for each literal vertex v we have $d(v) = 2$ (since in F , each variable appears twice as a positive and twice as a negative literal). To finish the construction of G , we will make use of the gadget shown in Figure 3(a), as well as some copies of S_5 , the star on 5 vertices. When we say that we *attach* a copy H of the gadget to the vertices v_i and v'_i (for some $1 \leq i \leq n$), we mean that we add H to G , and we identify the vertices w_1 and w_2 to the vertices v_i and v'_i respectively. Now:

- for each $1 \leq i \leq n$, we attach $n^{k'}$ copies of the gadget to the vertices v_i and v'_i of G . For convenience, we will give unique names to the vertices corresponding to each gadget added that way. So, the vertex u_i^l (for $1 \leq l \leq n^{k'}$ and $1 \leq i \leq n$) is used to represent the vertex u of the l^{th} copy of the gadget attached to v_i and v'_i , and $u_{i,1}^l$ ($u_{i,2}^l$ resp.) is used to denote the vertex u_1 (u_2 resp.) of that same gadget. Then,
- for each $1 \leq j \leq m$, we add $n^{k'} - 1$ copies of the clause vertex c_j to G , each one of these copies being adjacent to the same literal vertices as c_j . For $1 \leq l \leq n^{k'}$, the vertex c_j^l is the l^{th} copy of c_j . Finally,
- for each $1 \leq j \leq m$ and $1 \leq l \leq n^{k'}$, we add a copy of the star on 5 vertices S_5 to G and identify any degree-1 vertex of S_5 to c_j^l . Let s_j^l be the neighbour of c_j^l that also belongs to a copy of S_5 .

Observe that the resulting graph G (illustrated in Figure 3(b)) remains bipartite and that this construction is achieved in polynomial time in regards to $n + m$.

From the construction of G , we know that for every $1 \leq i \leq n$, $d(v_i) = d(v'_i) = \Theta(n^{k'})$. So, for sufficiently large n , the only pairs of adjacent vertices of G that have the same degrees are either the vertices u_i^l and $u_{i,2}^l$, or the vertices c_j^l and s_j^l (for every $1 \leq i \leq n$, $1 \leq l \leq n^{k'}$ and $1 \leq j \leq m$).

First, let F be a satisfiable formula and let t be a satisfying assignment of F . Also, let S be the set of literal vertices v_i (v'_i resp.) such that the corresponding literals x_i ($\neg x_i$ resp.) are assigned value *true* by t . Clearly $|S| = n$. We will also show that S is an $ir(G)$. Consider the graph $G' = G[V \setminus S]$. Now, for any $1 \leq i \leq n$, we have that either v_i or v'_i , say v_i , belongs to the vertices of G' . Now for every $1 \leq l \leq n^k$, we have that $d_{G'}(u_i^l) = 3$, while $d_{G'}(u_{i,1}^l) = 2$ and $d_{G'}(u_{i,2}^l) = 4$ (since none of the neighbours of $u_{i,1}^l$ and $u_{i,2}^l$ belongs to S). Also, for every $1 \leq j \leq m$ and $1 \leq l \leq n^{k'}$, since t is a satisfying assignment of F , $N(c_j^l)$ contains at least one vertex in S . It follows that $d_{G'}(c_j^l) = 3 < 4 = d_{G'}(s_j^l)$. Finally, since S does not contain any neighbours of v_i , we have that $d_{G'}(v_i) = d_G(v_i) = \mathcal{O}(n^{k'})$. It follows that S is an $ir(G)$ and thus that $I(G) \leq n$.

Now let F be a non-satisfiable formula and assume that there exists an S that is an $ir(G)$ with $|S| \leq n^{k'}$. As usual, let $G' = G[V \setminus S]$. Then:

1. For every $1 \leq j \leq m$, there exists a literal vertex v such that $v \in N(c_j^l)$ for every $1 \leq l \leq n^{k'}$. Assume that this is not true for a specific j . Then, since $d_G(c_j^l) = d_G(s_j^l) = 4$, for every $1 \leq l \leq n^{k'}$, we have that S contains at least one vertex in $N[\{c_j^l, s_j^l\}]$, which does not belong to the literal vertices. That is, S contains at least one (non-literal) vertex for each one of the $n^{k'}$ copies of c_j . Observe also that even if this is the case, S would also have to contain at least one more vertex to, for example, stop $u_{i,2}^1$ and u_i^1 , from having the same degree in G' . It follows that $|S| > n^{k'}$, which is a contradiction.
2. For every $1 \leq i \leq n$, S does not contain both v_i and v'_i . Assume this is not true for a specific i . Then, for every $1 \leq l \leq n^{k'}$, we have that $d_{G'}(u_i^l) = d_{G'}(u_{i,1}^l) = 2$, unless S also contains an additional vertex of the gadgets attached to v_i and v'_i , for each one of the $n^{k'}$ such gadgets. It follows that $|S| \geq n^{k'}$. Since we have also assumed that for a specific i , both v_i and v'_i belong to S , we have that $|S| > n^{k'}$, a contradiction.
3. For every $1 \leq i \leq n$, S contains at least one of v_i and v'_i . Assume this is not true for a specific i . Then, for every $1 \leq l \leq n^{k'}$, we have that $d_{G'}(u_i^l) = d_{G'}(u_{i,2}^l) = 4$, unless S also contains an additional vertex of the gadgets attached to v_i and v'_i , for each one of the $n^{k'}$ such gadgets. Even if this is the case, S would also have to contain at least one more vertex to, for example, stop c_1^1 and S_1^1 from having the same degree in G' . It follows that $|S| > n^{k'}$, which is a contradiction.

So from items 2. and 3. above, it follows that for each $1 \leq i \leq n$, S contains exactly one of v_i and v'_i . Now consider the following truth assignment: we assign the value *true* to every variable x_i if the corresponding literal vertex v_i belongs in S , and value *false* to every other variable. Now, from item 1. above, it follows that each clause C_j contains either a positive literal x_i which has been set to *true*, or a negative literal $\neg x_i$ which has been set to *false*. Thus F is satisfied, which is a contradiction.

Up to this point, we have shown that there exists a graph $G = (V, E)$ with $|V(G)| = N = \mathcal{O}(n^{k'+1})$ where

- $I(G) \leq n$ if F is satisfiable
- $I(G) > n^{k'}$ otherwise.

Therefore, we have that $I(G)$ is not $\mathcal{O}(n^{k'-1})$ approximable in polynomial time unless $\mathcal{P} = \mathcal{NP}$.

Now, since $N = |V(G)| = \Theta(n^{k'+1})$ and $2k = k' + 1$ we have $\mathcal{O}(n^{k'-1}) = \mathcal{O}(N^{\frac{k'-1}{k'+1}}) = \mathcal{O}(N^{1-\frac{2}{k'+1}}) = \mathcal{O}(N^{1-\frac{1}{k}})$. This ends the proof of this theorem. ◀

Now, we consider the case where G is regular bipartite graph. Below we present an upper bound to the size of $I(G)$. This upper bound is then used to obtain a (simple) Δ -approximation of an optimal solution.

24:10 Complexity of Finding Maximum Locally Irregular Induced Subgraphs

► **Theorem 11.** *For any d -regular bipartite graph $G = (L, R, E)$ of order n we have that $I(G) \geq n/2d$.*

Now recall that in any bipartite graph G , any bipartition of G is a vertex cover of G . Also observe that any vertex cover of a graph G , is also an irregulator of G . Indeed, deleting the vertices of any vertex cover of G , leaves us with an independent set, which is locally irregular. The next corollary follows from these observations and Theorem 11:

► **Corollary 12.** *For any d -regular bipartite graph $G = (L, R, E)$, any of the sets L and R is a d -approximation of $ir^*(G)$.*

5 Parameterised complexity

As the problem of computing a minimal irregulator of a given graph G seems to be rather hard to solve, and even to approximate, we focused our efforts towards finding parameterised algorithms that can solve it. First we present an FPT algorithm that calculates $I(G)$ when parameterised by the size of the solution and Δ , the maximum degree of the graph.

► **Theorem 13.** *For a given graph $G = (V, E)$ with $|V| = n$ and maximum degree Δ , and for $k \in \mathbb{N}$, there exists an algorithm that decides if $I(G) \leq k$ in time $(2\Delta)^k n^{\mathcal{O}(1)}$.*

The main tool we use to show Theorem 13 is Observation 5. Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. A high level description of our recursive algorithm is as follows: first find an edge $w \in E$ such that $d(u) = d(v)$. Now, assume that we are making a correct guess of a vertex $w \in N[\{u, v\}] \cap S$ where S is a minimum irregulator. Then, $G_w = G[V \setminus w]$ must have a minimum irregulator of size $|S| - 1$. Note that if we repeat the above process and we make correct guesses, we are going to stop after deleting $|S|$ vertices or when we have deleted k vertices (meaning that $I(G) > k$). Then, by considering all the 2Δ choices for w , we have a running time of $(2\Delta)^k$.

We now turn our attention towards graphs that are “close to being trees”, that is graphs of bounded treewidth. In particular, we provide an FPT algorithm that finds a minimum irregulator of G , when parameterised by the treewidth of the input graph and by Δ .

► **Theorem 14.** *For a given a graph $G = (V, E)$ and a nice tree decomposition of G , there exists an algorithm that returns $I(G)$ in time $\Delta^{2tw} n^{\mathcal{O}(1)}$, where tw is the treewidth of the given decomposition and Δ is the maximum degree of G .*

The idea of the proof of Theorem 14, is based on the classic dynamic programming technique on the given nice tree decomposition of G . Let us denote by B_c the bag of vertices of a node c of a nice tree decomposition of G . In essence, for each node c of the tree decomposition, we store the necessary information that allows us to find all the sets that are $ir(G, B_c^\downarrow \setminus B_c)$, where B_c^\downarrow denotes the vertices appearing in a sub-tree rooted at c . Then for the root r of the tree decomposition, we can check which of the stored sets that are $ir(G, B_r^\downarrow \setminus B_r)$, are also $ir(G)$; the minimum such set is an $ir^*(G)$.

The running time of our algorithm follows from the size of the tables we keep for these sets. In particular, for each set stored for a node c , for each vertex v of B_c , we keep the degree that we want v to have in the final, locally irregular graph (i. e. the graph G after the removal of $ir(G)$) and the degree that v has in $G[B_c^\downarrow \setminus S]$. This gives us Δ^2 choices for each vertex of B_c .

It is worth noting that the algorithms of Theorem 13 and 14 can be used in order to also return an $ir^*(G)$.

5.1 W-Hardness

Observe that both of the algorithms presented above, have to consider Δ as part of the parameter if they are to be considered as FPT. The natural question to ask at this point is whether we can have an FPT algorithm, when parameterised only by the size of the solution, or the treewidth of the input graph. In this section, we give a strong indication towards the negative answer for both cases, proving that, in some sense, the algorithms provided in Section 5 are optimal.

► **Theorem 15.** *Let G be a graph and $k \in \mathbb{N}$. Deciding if $I(G) \leq k$ is $W[2]$ -hard, when parameterised by k .*

The proof of Theorem 15 is done through a linear-fpt reduction from the DOMINATING SET problem, when parameterised by the size of the solution.

► **Theorem 16.** *Let G be a graph with treewidth tw , and $k \in \mathbb{N}$. Deciding if $I(G) = k$ is $W[1]$ -hard when parameterised by tw .*

Proof. We will present a reduction from the LIST COLOURING problem: the input consists of a graph $H = (V, E)$ and a list function $L : V \rightarrow \mathcal{P}(\{1, \dots, k\})$ that specifies the available colours for each vertex $u \in V$. The goal is to find a proper colouring $c : V \rightarrow \{1, \dots, k\}$ such that $c(u) \in L(u)$ for all $u \in V$. When such a colouring exists, we say that (H, L) is a *yes-instance* of LIST COLOURING. This problem is known to be $W[1]$ -hard when parameterised by the treewidth of H [19].

Now, starting from an instance (H, L) of LIST COLOURING, we will construct a graph $G = (V', E')$ (see Figure 4 (a)) such that:

- $|V'| = \mathcal{O}(|V|^6)$,
- $tw(G) = tw(H)$ and
- $I(G) = nk$ if and only if (H, L) is a yes-instance of LIST COLOURING.

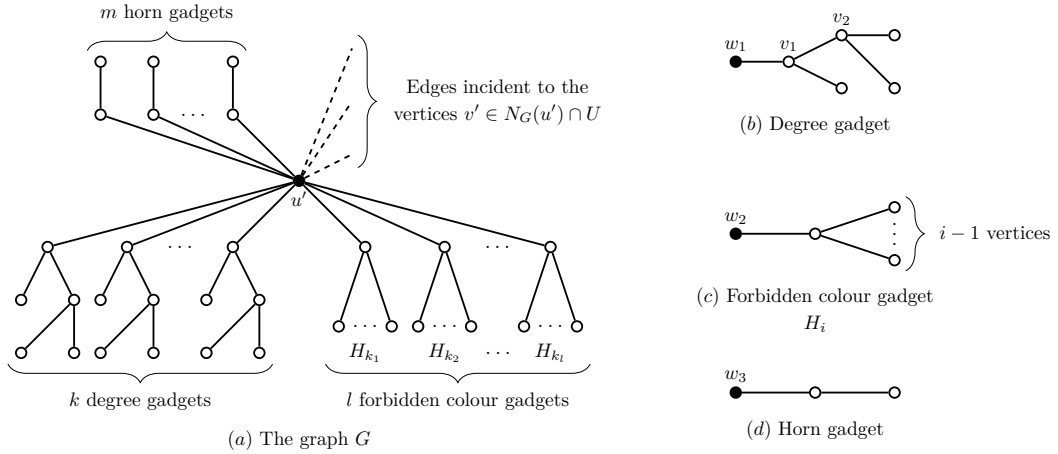
Before we start with the construction of G , let us give the following observation.

► **Observation 17.** *Let (H, L) be an instance of LIST COLOURING where $H = (V, E)$ and there exists a vertex $u \in V$ such that $|L(u)| > d(u)$. Then the instance $(H[V \setminus \{u\}], L')$, where $L'(v) = L(v)$ for all $v \in V \setminus \{u\}$, is a yes-instance of LIST COLOURING if and only if (H, L) is a yes-instance of LIST COLOURING.*

Indeed, observe that for any vertex $u \in V$, by any proper colouring c of H , $c(u)$ only has to avoid $d(u)$ colours. Since $|L(u)| > d(u)$, we will always have a spare colour to use on u that belongs in $L(u)$. From the previous observation, we can assume that in our instance, for all $u \in V$, we have $|L(u)| \leq d(u)$. Furthermore, we can deduce that $k \leq n(n-1)$ as the degree of any vertex is at most $n-1$. Finally, let us denote by $\bar{L}(u)$ the set $\{0, 1, \dots, k\} \setminus L(u)$. It is important to note here that for every $u \in V$, the list $L(u)$ contains at least one element belonging in $\{1, \dots, k\}$. It follows that $\bar{L}(u)$ also contains at least one element, the colour 0. To sum up, we have that $1 \leq |\bar{L}(u)| \leq k$.

Now, we present the three gadgets we are going to use in the construction of G . First, we have the “forbidden colour gadget” H_i , which is a star with i leaves (see Figure 4(c)). When we say that we attach a copy of H_i on a vertex v of a graph G , we mean that we add H_i to G and we identify the vertices v and w_2 (where here and in what follows, we are using the naming illustrated in Figure 4 when talking about the vertices w_1, w_2, w_3, v_1 and v_2). The second, will be the “degree gadget”, which is presented in Figure 4(b). Finally, we have the “horn gadget”, which is a path on three vertices (see Figure 4(d)). We define the

24:12 Complexity of Finding Maximum Locally Irregular Induced Subgraphs



■ **Figure 4** In (a) we illustrate the construction of G , as it is described in the proof of Theorem 16. The black vertex represents every vertex that belongs in U . For the specific vertex u' shown in the figure, we have that $\bar{L}(u) = \{c_1, \dots, c_l\}$ and $k_i = n^3 - c_i$ for all $i = 1, \dots, l$. We also have that $m = 2n^3 - d_G(u) - k - l$.

operation of attaching these two gadgets on a vertex v of a graph G similarly to how we defined this operation for the forbidden colour gadget (each time using the appropriate w_1 or w_3 , according to if it is a degree or a horn gadget respectively).

In order to construct G , we start from a copy of H . Let us use $G|_H$ to denote the copy of H that lies inside of G and, for each vertex $u \in V$, let u' be its copy in V' . We will call the set of these vertices U . That is, $U = \{v \in V(G|_H)\}$. Then, we are going to attach several copies of each gadget to u' , for each vertex $u' \in U$. We start by attaching k copies of the degree gadget to each vertex $u' \in U$. Then, for each $u \in V$ and each $i \in \bar{L}(u)$, we attach one copy of the forbidden colour gadget H_{2n^3-i} to the vertex u' . Finally, for each $u' \in U$, we attach to u' as many copies of the horn gadget as are needed, in order to have $d_G(u') = 2n^3$.

Before we continue, observe that, for sufficiently large n , we have attached more than n^3 horn gadgets to each vertex of U . Indeed, before attaching the horn gadgets, each vertex $u' \in U$ has $d_G(u) \leq n - 1$ neighbours in U , k neighbours from the degree gadgets and at most $k < n^2$ neighbours from the forbidden colour gadgets (recall that $|\bar{L}(u)| \leq k$). We will now show that $|V'| = \mathcal{O}(n^6)$. For that purpose, let us calculate the number of vertices in all the gadgets attached to a single vertex $u' \in U$. First, we have $5k < 5n^2$ vertices in the degree gadgets. Then, we have less than $4n^3$ vertices in the horn gadgets (as we have less than $2n^3$ such gadgets). Finally, we have at most $k < n^2$ forbidden colour gadgets, each one of which containing at most $2n^3$ vertices. So, for each vertex $u' \in U$, we have at most $2n^5 + 4n^3 + 5n^2$ vertices in the gadgets attached to u' . Therefore, we have $|V'| = \mathcal{O}(n^6)$.

Before we prove that $I(G) \leq nk$ if and only if (H, L) is a yes-instance of LIST COLOURING, we need to argue about two things. First, about the treewidth of the graph G and second, about the minimum value of $I(G)$. Since our construction only attaches trees to each vertex of $G|_H$ (and recall that a tree has a treewidth of 1 by definition), we know that $tw(G) = tw(G|_H) = tw(H)$. As for $I(G)$, we will show that it has to be at least equal to nk . For that purpose we have the following two claims.

▷ **Claim 18.** Let S be an $ir(G)$ and $S \cap U \neq \emptyset$. Then $|S| > n^3$.

▷ **Claim 19.** Let S be an $ir(G)$ and $S \cap U = \emptyset$. Then $|S| \geq nk$. In particular, S includes at least one vertex from each copy of the degree gadget used in the construction of G .

By the previous two claims, we conclude that $I(G) \geq nk$. We are ready to show that, if (H, L) is a yes-instance of LIST COLOURING, then there exists a set $S \subseteq V'$ such that S is an $ir(G)$ and $|S| = nk$. Let c be a proper colouring of H such that $c(u) \in L(u)$ for all $u \in V$. We will construct an $ir(G)$ as follows. For each $u \in V$, we partition (arbitrarily) the k degree gadgets attached to the vertex u' to $c(u)$ “good” and $(k - c(u))$ “bad” degree gadgets. For each good degree gadget, we add the copy of the vertex v_1 of that gadget to S and for each bad degree gadget we add the copy of the vertex v_2 of that gadget to S . This process creates a set S of size nk , as it includes k distinguished vertices for each vertex $u' \in U$.

Now we need to show that S is an $ir(G)$. Let $G' = G[V' \setminus S]$; observe that each vertex $u' \in U$ has degree $d_{G'}(u') = 2n^3 - c(u)$. Therefore, u' does not have the same degree as any of its neighbours that do not belong in U . Indeed, for every $v \in N_{G'}(u') \setminus U$, we have that $d_{G'}(v) \in \{1, 2\}$ (if v belongs to a bad degree or a horn gadget) or $d_{G'}(v) \in \{2n^3 - i : i \in \bar{L}(u)\}$ (if v belongs to a forbidden colour gadget). Furthermore, since c is a proper colouring of H , for all $uv \in E$, we have that $c(u) \neq c(v)$. This gives us that for any edge $u'v' \in E'$ with $u', v' \in U$, we have that $d_{G'}(u') = 2n^3 - c(u) \neq 2n^3 - c(v) = d_{G'}(v')$.

So, we know that for every vertex $u' \in U$, there is no vertex $w \in N_{G'}(u')$ such that $d_{G'}(u') = d_{G'}(w)$. It remains to show that, in G' , there exist no two vertices belonging to the same gadget, which have the same degrees. First of all, we have that S does not contain any vertex from any of the horn and forbidden colour gadgets, nor from U . Thus any adjacent vertices belonging to these gadgets have different degrees. Last, it remains to check the vertices of the degree gadgets. Observe that for any copy of the degree gadget, S contains either v_1 or v_2 . In both cases, after the deletion of the vertices of S , any adjacent vertices belonging to any degree gadget have different degrees. Therefore, S is an $ir(G)$ of order nk and since $I(G) \geq nk$ we have that $I(G) = nk$.

Now, for the opposite direction, assume that there exists a set $S \subseteq V'$ such that S is an $ir^*(G)$ and $|S| = nk$. Let $G' = (V'', E'')$ be the graph $G[V' \setminus S]$. It follows from Claim 18 and Claim 19, that $S \cap U = \emptyset$ and that S contains exactly one vertex from each copy of the degree gadget in G and no other vertices. Consider now the colouring c of H defined as $c(u) = 2n^3 - d_{G'}(u')$. We will show that c is a proper colouring for H and that $c(u) \in L(u)$. First, we have that c is a proper colouring of H . Indeed, for any edge $uv \in E$, there exists an edge $u'v' \in E''$ (since $S \cap U = \emptyset$). Since G' is locally irregular we have that $d_{G'}(u') \neq d_{G'}(v')$, and thus $c(u) \neq c(v)$. It remains to show that $c(u) \in L(u)$ for all $u \in V$. First observe that, during the construction of G , we attached exactly k degree gadgets to each $u' \in U$. It follows that $d_{G'}(u') = 2n^3 - j$ and $c(u) = j$ for a $j \in \{0, 1, \dots, k\}$. It is sufficient to show that $j \notin \bar{L}(u)$. Since S contains only vertices from the copies of the degree gadgets, we have that each $u' \in U$ has exactly one neighbour of degree $2n^3 - i$ for each $i \in \bar{L}(u)$ (this neighbour is a vertex of the H_i forbidden colour gadget that was attached to u'). Furthermore, for all $u' \in U$, since G' is locally irregular, we have that $d_{G'}(u') \neq 2n^3 - i$ for all $i \in \bar{L}(u)$. Equivalently, $d_{G'}(u') = 2n^3 - j$ for any $j \in L(u)$. Thus, $c(u) \in L(u)$ for all $u \in V$. ◀

Note that the reductions presented in the proofs of Theorem 15 and Theorem 16 are linear fpt-reductions. Additionally we know that

- there is no algorithm that answers if a graph G of order n has a Dominating Set of size at most k in time $f(k)n^{o(k)}$ unless the ETH fails [26] and
- there is no algorithm that answers if an instance (G, L) of the LIST COLOURING is a yes-instance in time $\mathcal{O}^*(f(tw)n^{o(tw)})$ unless the ETH fails [19].

So, the following corollary holds.

► **Corollary 20.** *Let G be a graph of order n and assume the ETH. For $k \in \mathbb{N}$, there is no algorithm that decides if $I(G) \leq k$ in time $f(k)n^{o(k)}$. Furthermore, assuming that G has treewidth tw , there is no algorithm that computes $I(G)$ in time $\mathcal{O}^*(f(tw)n^{o(tw)})$.*

6 Conclusion

In this work we introduce the problem of identifying the largest locally irregular induced subgraph of a given graph. There are many interesting directions that could be followed for further research. An obvious one is to investigate whether the problem of calculating $I(G)$ remains \mathcal{NP} -hard for other, restricted families of graphs. The first candidate for such a family would be the one of chordal graphs. On the other hand, there are some interesting families, for which the problem of computing an optimal irregularator could be decided in polynomial time, such as split graphs. Also, it could be feasible to conceive approximation algorithms for regular bipartite graphs, which have a better approximation ratio than the (simple) algorithm we present. The last aspect we find intriguing, is to study the parameterised complexity of calculating $I(G)$ when considering other parameters, like the size of the minimum vertex cover of G , with the goal of identifying a parameter that suffices, by itself, in order to have an FPT algorithm. Finally, it is worth investigating whether calculating $I(G)$ could be done in FPT time (parameterised by the size of the solution) in the case where G is a planar graph.

References

- 1 Agostinho Agra, Geir Dahl, Torkel Andreas Haufmann, and Sofia J. Pinheiro. The k -regular induced subgraph problem. *Discrete Applied Mathematics*, 222:14–30, 2017. doi:10.1016/j.dam.2017.01.029.
- 2 Yousef Alavi, Alfred Boals, Gary Chartrand, Ortrud Oellermann, and Paul Erdős. K -path irregular graphs. *Congressus Numerantium*, 65, January 1988.
- 3 Yousef Alavi, Gary Chartrand, Fan R. K. Chung, Paul Erdős, Ronald L. Graham, and Ortrud R. Oellermann. Highly irregular graphs. *Journal of Graph Theory*, 11(2):235–249, 1987. doi:10.1002/jgt.3190110214.
- 4 Akhbar Ali, Gary Chartrand, and Ping Zhang. *Irregularity in Graphs*. Springer briefs in mathematics. Springer, 2021. doi:10.1007/978-3-030-67993-4.
- 5 Yuichi Asahiro, Hiroshi Eto, Takehiro Ito, and Eiji Miyano. Complexity of finding maximum regular induced subgraphs with prescribed degree. *Theoretical Computer Science*, 550:21–35, 2014. doi:10.1016/j.tcs.2014.07.008.
- 6 Olivier Baudon, Julien Bensmail, Jakub Przybyło, and Mariusz Woźniak. On decomposing regular graphs into locally irregular subgraphs. *European Journal of Combinatorics*, 49:90–104, 2015. doi:10.1016/j.ejc.2015.02.031.
- 7 Rémy Belmonte and Ignasi Sau. On the complexity of finding large odd induced subgraphs and odd colorings. *Algorithmica*, 83(8):2351–2373, 2021. doi:10.1007/s00453-021-00830-x.
- 8 Julien Bensmail, Martin Merker, and Carsten Thomassen. Decomposing graphs into a constant number of locally irregular subgraphs. *European Journal of Combinatorics*, 60:124–134, 2017. doi:10.1016/j.ejc.2016.09.011.
- 9 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity*, 2003. URL: <https://ecc.weizmann.ac.il/eccc-reports/2003/TR03-049/index.html>.
- 10 Hans L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.
- 11 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

- 12 Gary Chartrand, Paul Erdős, and Ortrud Oellermann. How to define an irregular graph. *The College Mathematics Journal*, 19, January 1988. doi:10.2307/2686701.
- 13 Gary Chartrand, Michael Jacobon, Jenő Lehel, Ortrud Oellermann, Sergio Ruiz, and Farrokh Saba. Irregular networks. *Congressus Numerantium*, 64, January 1986.
- 14 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized np-hard problems. *Information and Computation*, 201(2):216–231, 2005. doi:10.1109/CCC.2004.1313826.
- 15 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. On the computational hardness based on linear FPT-reductions. *Journal of Combinatorial Optimization*, 11(2):231–247, 2006. doi:10.1007/s10878-006-7137-6.
- 16 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.
- 17 Miroslav Chlebík and Janka Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theor. Comput. Sci.*, 354(3):320–338, 2006. doi:10.1016/j.tcs.2005.11.029.
- 18 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. doi:10.1007/978-3-662-53622-3.
- 19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 20 Alan M. Frieze, Ronald J. Gould, Michal Karonski, and Florian Pfender. On graph irregularity strength. *Journal of Graph Theory*, 41(2):120–137, 2002. doi:10.1002/jgt.10056.
- 21 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 22 Michał Karoński, Tomasz Łuczak, and Andrew Thomason. Edge weights and vertex colors. *Journal of Combinatorial Theory*, 91:151–157, May 2004. doi:10.1016/j.jctb.2003.12.001.
- 23 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.
- 24 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
- 25 Carla Negri Lintzmayer, Guilherme Oliveira Mota, and Maycon Sambinelli. Decomposing split graphs into locally irregular graphs. *Discrete Applied Mathematics*, 292:33–44, 2021. doi:10.1016/j.entcs.2019.08.053.
- 26 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the European Association for Theoretical Computer Science*, 105:41–72, 2011. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/92>.
- 27 Bojan Mohar. Face covers and the genus problem for apex graphs. *J. Comb. Theory, Ser. B*, 82(1):102–117, 2001. doi:10.1006/jctb.2000.2026.
- 28 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009. doi:10.1016/j.jda.2008.09.005.
- 29 Jakub Przybyło. Irregularity strength of regular graphs. *Electronic Journal of Combinatorics*, 15, June 2008. doi:10.37236/806.
- 30 Jakub Przybyło. On decomposing graphs of large minimum degree into locally irregular subgraphs. *Electronic Journal of Combinatorics*, 23(2):2–31, 2016. doi:10.37236/5173.

A

 Omitted proofs

A.1 Proof of Theorem 13

Let us first present the following lemma:

► **Lemma 21.** *Let $G = (V, E)$ be a graph such that, G is not locally irregular, and S be an $ir^*(G)$. Furthermore let $G_v = (V', E')$ be the graph $G[V \setminus \{v\}]$ for a vertex $v \in S$. Then $I(G_v) = I(G) - 1$.*

Proof. First observe that $S' = S \setminus \{v\}$ must be an $ir(G_v)$ as $G_v[V' \setminus S'] = G[V \setminus S]$. It follows that $I(G_v) \leq I(G) - 1$. Assume that $I(G_v) < I(G) - 1$. Then there exists an S'' such that $|S''| < I(G) - 1$ and S'' is an $ir(G_v)$. Since $G_v[V' \setminus S''] = G[V \setminus (S'' \cup \{v\})]$, we have that $S'' \cup \{v\}$ is an $ir(G)$ and $|S'' \cup \{v\}| = |S''| + 1 < I(G)$. This is a contradiction. ◀

Now, we are ready to present the proof of the theorem.

Proof of Theorem 13. In order to decide if $I(G) \leq k$ we are going to use a recursive algorithm. The algorithm has input (G, k) , where $G = (V, E)$ is a graph and $k \geq 0$ is an integer. The basic idea of this algorithm, is to take advantage of Observation 5. We present the exact procedure in Algorithm 1.

■ **Algorithm 1** [IsIrregular(G, k) decision function].

Input: A graph $G = (V, E)$ and an integer $k \geq 0$.

Output: Is $I(G) \leq k$ or not?

```

1: if  $G$  is irregular then
2:   return yes
3: else if  $k = 0$  then
4:   return no
5: else ▷  $k > 0$  and  $G$  is not irregular
6:    $ans \leftarrow no$ 
7:   find an edge  $vu \in E$  such that  $d_G(v) = d_G(u)$ 
8:   for all  $w \in N_G[\{u, v\}]$  do
9:     set  $G_w = G[V \setminus \{w\}]$ 
10:    if IsIrregular( $G_w, k - 1$ ) returns yes then
11:       $ans \leftarrow yes$ 
12:   return  $ans$ 

```

Now, let us argue about the correctness and the efficiency of this algorithm. We claim that for any graph $G = (V, E)$ and any integer $k \geq 0$, Algorithm 1 returns yes if $I(G) \leq k$ and no otherwise. Furthermore, the number of steps that the algorithm requires, is $f(k, n) = (2\Delta)^k n^{\mathcal{O}(1)}$, where $n = |V|$. We will prove this by induction on k .

Base of the induction ($k = 0$): Here, we only need to check if G is locally irregular. Algorithm 1 does this in line 1 and returns yes if it is (line 2) and no otherwise (line 4). Furthermore, we can check if G is locally irregular in polynomial time. So, the claim is true for the base.

Induction hypothesis ($k = k_0 \geq 0$): We assume that we have a $k_0 \geq 0$ such that Algorithm 1 can decide if any graph G with n vertices and maximum degree Δ has $I(G) \leq k_0$ in $f(k_0, n) = (k_0 + 1)(2\Delta)^{k_0} n^{\mathcal{O}(1)}$ steps.

Induction step ($k = k_0 + 1$): Let $G = (V, E)$ be a graph. If G is locally irregular then $I(G) = 0$ and Algorithm 1 answers correctly (in line 2). Assume that G is not locally irregular; then there exist an edge $vu \in E$ such that $d_G(v) = d_G(u)$. Now, let S be an $ir^*(G)$. It follows from Observation 5 that S must include at least one vertex $w \in N_G[\{v, u\}]$. Since Algorithm 1 considers all the vertices in $N_G[\{v, u\}]$, at some point it also considers the vertex $w \in S \cap N_G[\{v, u\}]$. Now, observe that for any $x \in S$, the set $S_x = S \setminus \{x\}$ is an $ir^*(G_x)$, where $G_x = G[V \setminus \{x\}]$. Furthermore, by Lemma 21, we have $I(G_x) \leq k - 1 = k_0$ iff $I(G) \leq k$. By the induction hypothesis, we know that the algorithm answers correctly for all the instances (G_x, k_0) . Thus, if $I(G) \leq k = k_0 + 1$, there must exist one instance (G_w, k_0) , where $w \in S \cap N_G[\{v, u\}]$, for which the Algorithm 1 returns yes. Therefore the algorithm answers for $(G, k_0 + 1)$ correctly. Finally, this process request $n^{\mathcal{O}(1)}$ steps in order to check if the graph is locally irregular and $2\Delta f(k - 1, n - 1)$ steps (by induction hypothesis) in order to check if for any graph G_x we have $I(G_x) \leq k - 1 = k_0$ (where $x \in N[\{u, v\}]$). So, the algorithm decides in $n^{\mathcal{O}(1)} + 2\Delta f(k - 1, n - 1) \leq n^{\mathcal{O}(1)} + 2\Delta k(2\Delta)^{k-1}(n - 1)^{\mathcal{O}(1)} \leq n^{\mathcal{O}(1)} + k(2\Delta)^k n^{\mathcal{O}(1)} \leq (k + 1)(2\Delta)^k n^{\mathcal{O}(1)}$ steps. Finally, note that $k \leq n - 1$, and the result follows. \blacktriangleleft

A.2 Proof of Theorem 14

Proof. As the techniques we are going to use are standard, we are sketching some of the introductory details. For more details on tree decompositions (definition and terminology) see [19]. We are going to perform dynamic programming on the nodes of the given nice tree decomposition (see [10] for the definition of a nice tree decomposition). For a node t of the given tree decomposition of G , we denote by B_t the bag of this node and by B_t^\downarrow the set of vertices of the graph that appears in the bags of the nodes of the subtree with t as a root. Observe that $B_t \subseteq B_t^\downarrow$.

The idea behind our algorithm, is that for each node t we store all the sets $S \subseteq B_t^\downarrow$ such that S is an $ir(G, B_t^\downarrow \setminus B_t)$. We will also store the necessary “conditions” (explained more in what follows) such that if there exists a set S' , where $S' \setminus S \subseteq V \setminus B_t^\downarrow$, that meets these conditions, then S' is an $ir(G, B_t^\downarrow)$. Observe that if we manage to do such a thing for every node of the tree decomposition, then we can find $I(G)$. To do so, it suffices to check the size of all the irregulators we stored for the root r of the tree decomposition, which also meet the conditions we have set. In that way, we can find a set S that is an $ir(G, B_r^\downarrow \setminus B_r)$, satisfies our conditions and is of minimum order, and since $B_r^\downarrow = V$, this set S is a minimum irregulator of G and $I(G) = |S|$.

Let us now present the actual information we are keeping for each node. Assume that t is a node of the tree decomposition and $S \subseteq B_t^\downarrow$ is an irregulator of $B_t^\downarrow \setminus B_t$ in G , i. e., S is an $ir(G, B_t^\downarrow \setminus B_t)$. For this S we want to remember which vertices of B_t belong to S as well as the degrees of the vertices $v \in B_t \setminus S$ in $G[B_t^\downarrow \setminus B_t]$. This can be done by keeping a table D of size $tw + 1$ where, if $v \in B_t \setminus S$ we set $D(v) = d_{G[B_t^\downarrow \setminus B_t]}(v)$ and if $v \in B_t \cap S$ we set $D(v) = \emptyset$ (slightly abusing the notation, by $D(v)$ we mean the position in the table D that corresponds to the vertex v). Like we have already said, we are going to keep some additional information about the conditions that could allow these sets to be extended to irregulators of B_t^\downarrow in G if we add vertices of $V \setminus B_t^\downarrow$. For that reason, we are also going to keep a table with the “target degree” of each vertex; in this table we assign to each vertex $v \in B_t \setminus S$ a degree d_v such that, if there exists S' where $S' \setminus S \subseteq V \setminus B_t^\downarrow$ and for all $v \in B_t \setminus S$ we have $d_{G[V \setminus S]}(v) = d_v$, then S is an $ir(G, B_t^\downarrow)$. This can be done by keeping a table T of size $tw + 1$ where for each $v \in B_t \setminus S$ we set $T(v) = i$, where i is the target degree, and for each $v \in B_t \cap S$ we set $T(v) = \emptyset$. Such tables T will be called *valid* for S in B_t . Finally, we are going to keep the set $X = S \cap B_t$ and the value $min = |S|$. Note that the set X does not gives us any extra information, but we keep it as it will be useful to refer to it directly.

24:18 Complexity of Finding Maximum Locally Irregular Induced Subgraphs

To sum up, for each node t of the tree decomposition of G , we keep a set of quadruples (X, D, T, \min) , each quadruple corresponding to a valid combination of a set S that is an $ir(G, B_t^\downarrow \setminus B_t)$ and the target degrees for the vertices of $B_t \setminus S$. Here it is important to say that when treating the node B_t , for every two quadruples (X_1, D_1, T_1, \min_1) and (X_2, D_2, T_2, \min_2) such that for all $v \in B_t$ we have that $D_1(v) = D_2(v)$ and $T_1(v) = T_2(v)$ (this indicates that $X_1 = X_2$ as well), then we are only going to keep the quadruple with the minimum value between \min_1 and \min_2 as we will prove that this is enough in order to find $I(G)$.

▷ **Claim 22.** Assume that for a node t , we have two sets S_1 and S_2 that are both $ir(G, B_t^\downarrow \setminus B_t)$, and that T is a target table that is common to both of them. Furthermore, assume that $(X_1, D_1, T, |S_1|)$ and $(X_2, D_2, T, |S_2|)$ are the quadruples we have to store for S_1 and S_2 respectively (both respecting T), with $D_1(v) = D_2(v)$ for every $v \in B_t$. Then for any set $S \subseteq V \setminus B_t^\downarrow$ such that $d_{G[V \setminus (S_1 \cup S)]}(v) = T(v)$ for all $v \in B_t$, we also have that $d_{G[V \setminus (S_2 \cup S)]}(v) = T(v)$ for all $v \in B_t$.

Proof. Assume that we have such an S for S_1 , let v be a vertex in B_t and $H = G[v \cup ((V \setminus B_t^\downarrow) \setminus S)]$ (observe that H does not depend on S_1 or S_2). Since $d_{G[V \setminus (S_1 \cup S)]}(v) = T(v)$, we know that in the graph H , v has exactly $T(v) - D_1(v)$ neighbours (as $D_1(v) = d_{G[B_t^\downarrow \setminus S_1]}(v)$). Now, since $D_1(v) = D_2(v) = d_{G[B_t^\downarrow \setminus S_2]}(v)$ we have that $d_{G[V \setminus (S_2 \cup S)]}(v) = T(v)$. Therefore, the claim holds. ◁

Simply put, Claim 22 states that for any two quadruples $Q_1 = (X, D, T, \min_1)$ and $Q_2 = (X, D, T, \min_2)$, any extension S of S_1 is also an extension of S_2 (where S_1 and S_2 are the two sets that correspond to Q_1 and Q_2 respectively). Therefore, in order to find the minimum solution, it is sufficient to keep the quadruple that has the minimum value between \min_1 and \min_2 .

Now we are going to explain how we create all the quadruples (X, D, T, \min) for each type of node in the tree decomposition. First we have to deal with the Leaf Nodes. For a Leaf node t we know that $B_t = B_t^\downarrow = \emptyset$. Therefore, we have only one quadruple (X, D, T, \min) , where the size of both D and T is zero (so we do not need to keep any information in them), $S = \emptyset$ and $\min = |S| = 0$.

Now let t be an Introduce node; assume that we have all the quadruples (X, D, T, \min) for its child c and let v be the introduced vertex. By construction, we know that v is introduced in B_t and thus it has no neighbours in $B_t^\downarrow \setminus B_t$. It follows that if $S \subseteq B_c^\downarrow$ is an irregularator for $B_c^\downarrow \setminus B_c$, then both S and $S \cup \{v\}$ are irregularators for $B_t^\downarrow \setminus B_t$ in G . Furthermore, there is no set $S \subseteq B_t^\downarrow \setminus \{v\}$ that is an irregularator of $B_t^\downarrow \setminus B_t$ and is not an irregularator of $B_c^\downarrow \setminus B_c$. So, we only need to consider two cases for the quadruples we have to store for c ; if v belongs in the under-construction irregularator of $B_t^\downarrow \setminus B_t$ in G or not.

Case 1. (v is in the irregularator): Observe that for any S that is an $ir(G, B_c^\downarrow \setminus B_c)$, which is stored in the quadruples of B_c , for every $u \in B_c \setminus S$, we have that $d_{G[B_c^\downarrow \setminus S]}(u) = d_{G[B_t^\downarrow \setminus (S \cup \{v\})]}(u)$. Moreover, for any target table T which is valid for S in c , the target table T' is valid for $S \cup \{v\}$ in t , where T' is almost the same as T , the only difference being that T' also contains the information about v , i.e., $T'(v) = \emptyset$. So, for each quadruple (X, D, T, \min) in c , we need to create one quadruple $(X \cup \{v\}, D', T', \min + 1)$ for t , where D' is the almost the same as D , except that it also contains the information about v , i.e., $D'(v) = \emptyset$.

Case 2. (v is not in the irregularator): Let $q = (X, D, T, \min)$ be a stored quadruple of c and S be the corresponding $ir(G, B_c^\downarrow \setminus B_c)$. We will first explain how to construct D' of t , based on q . Observe that the only change between $G[B_c^\downarrow \setminus S]$ and $G[B_t^\downarrow \setminus S]$, is that in the latter there exist some new edges from v to some of the vertices of B_c . Therefore, for

each vertex $u \in B_c \setminus X$ we set $D'(u) = D(u) + 1$ if $u \in N[v]$ and $D'(u) = D(u)$ otherwise. Finally, for the introduced vertex v , we set $D'(v) = |N(v) \cap (B_c \setminus X)|$. We will now treat the target degrees for t . Observe that the target degrees for each vertex in $B_t \setminus \{v\}$ are the same as in T , since v only has edges incident to vertices in B_t . Now, we only need to decide which are the valid targets for v . Since $d_{G[B_t^\downarrow \setminus S]}(v) = D'(v)$, we know that for every target t' , we have that $D'(v) \leq t' \leq \Delta$. Furthermore, we can not have the target degrees of v to be the same as the targets of one of its neighbours in B_c (these values are stored in T), as, otherwise, any valid target table T' of t would lead to adjacent vertices in B_t having the same degree. Let $\{t_1, \dots, t_k\} \subset \{D(v), \dots, \Delta\}$ be an enumeration of all the valid targets for v (i.e. $t_i \neq T(u)$ for all $u \in N[v] \cap B_c \setminus X$). Then, for each quadruple (X, D, T, \min) in c , and for each $i = 1, \dots, k$, we need to create the quadruple (X, D', T_i, \min) , such that $T_i(u) = T(u)$ for all $u \in B_c$ and $T_i(v) = t_i$. In total, we have $k \leq \Delta$ such quadruples.

Now, let us explain how we deal with the Join nodes. Assume that t is a Join Node with c_1 and c_2 as its two children in the tree decomposition. Here, it is important to mention that $B_{c_1} = B_{c_2}$ and $(B_{c_1}^\downarrow \setminus B_{c_1}) \cap (B_{c_2}^\downarrow \setminus B_{c_2}) = \emptyset$. Assume that there exists an irregulator S of $B_t^\downarrow \setminus B_t$ in G , a valid target table T of S , and let (X, D, T, \min) be the quadruple we need to store in t for this pair (S, T) . Observe that this pair (S, T) is valid for both c_1 and c_2 , so we must already have stored at least one quadruple in each node. Let $X \subseteq B_t$ and a target table T such that (X, D_1, T, \min_1) and (X, D_2, T, \min_2) are stored for c_1 and c_2 respectively. We create the quadruple (X, D, T, \min) for t by setting $D(u) = D_1(u) + D_2(u) - d_{G[B_t \setminus X]}(u)$ for all $u \in B_t \setminus X$, $D(u) = \emptyset$ for all $u \in X$ and $\min = \min_1 + \min_2 - |X|$. Observe that these are the correct values for the $D(u)$ and \min , as otherwise we would count $d_{G[B_t \setminus X]}(u)$ and $|X|$ twice. Finally, we need to note that we do not store any quadruple (X, D, T, \min) we create for the Join Node such that $D(u) > T(u)$ for a vertex $u \in B_t \setminus X$. This is because for such quadruples, the degree of vertex u will never be equal to any of the target degrees we have set, as it can only increase when we consider any of the ancestor (i.e. parent, grandparent etc.) nodes of t .

Finally, we need to treat the Forget nodes. Let t be a Forget node, c be the its child and v be the forgotten vertex. Assume that we have to store in t a quadruple (X, D, T, \min) . Then, since $X = B_t \cap S$ for an irregulator S of B_t in G , we know that in c we must have already stored a quadruple (X', D', T', \min') such that, $X' = S \cap B_c$, $D'(u) = D(u)$ for all $u \in B_c$, $T'(u) = T(u)$ for all $u \in B_c$ and $\min' = \min$. Therefore, starting from the stored quadruples in c , we can create all the quadruples of t . For each quadruple (X', D', T', \min') in c , we create at most one quadruple (X, D, T, \min) for t by considering two cases; the forgotten vertex v_f belongs to X' or not.

Case 1. (v belongs to X'): then the quadruple (X, D, T, \min) is almost the same as (X', D', T', \min') , with the following differences: $X = X' \setminus \{v\}$, $\min = \min'$, $D(u) = D'(u)$ and $T(u) = T'(u)$ for all $u \in B_t$ and the tables D and T do not include any information for v as this vertex does not belong to B_t anymore.

Case 2. (v does not belong to X'): we will first check if $D'(v_f) = T'(v_f)$ or not. This is important because the degree of the v will never again be considered by our algorithm, and thus its degree will remain unchanged. So, if $D'(v_f) = T'(v_f)$, we create the quadruple (X, D, T, \min) where $X = X'$, $\min = \min'$, $D(u) = D'(u)$ and $T(u) = T'(u)$ for all $u \in B_t$ and the tables D and T do not include any information for v .

For the running time, observe that the number of nodes of a nice tree decomposition is $\mathcal{O}(tw \cdot n)$ and all the other calculations are polynomial in $n + m$. Thus we only need to count the different quadruples in each node. Now, for each vertex v , we either include it in X or we have $\Delta + 1$ options for the value $D(u)$ and $\Delta + 1 - i$ for the value $T(u)$ if $D(u) = i$. Also,

24:20 Complexity of Finding Maximum Locally Irregular Induced Subgraphs

for sufficiently large Δ , we have that $1 + \sum_{i=0}^{\Delta} (\Delta + 1 - i) < \Delta^2$. Furthermore, the set X and the value min do not increase the number of quadruples because $X = \{u \mid D(u) = \emptyset\}$ and from all quadruples $(X, D_1, T_1, min_1), (X, D_2, T_2, min_2)$ such that $D_1(u) = D_2(u)$ and $T_1(u) = T_2(u)$ for all $u \in B_t$, we only keep one of them (by Claim 22).

In total, the number of different quadruples in each node is Δ^{2tw} , and therefore the algorithm decides in $\Delta^{2tw} n^{\mathcal{O}(1)}$ time. \blacktriangleleft

An Almost Optimal Algorithm for Unbounded Search with Noisy Information

Junhao Gan  

School of Computing and Information Systems, The University of Melbourne, Australia

Anthony Wirth  

School of Computing and Information Systems, The University of Melbourne, Australia

Xin Zhang  

School of Computing and Information Systems, The University of Melbourne, Australia

Abstract

Given a sequence of integers, $\mathcal{S} = s_1, s_2, \dots$ in ascending order, called the *search domain*, and an integer t , called the *target*, the *predecessor problem* asks for the *target index* N such that s_N is the largest integer in \mathcal{S} satisfying $s_N \leq t$. We consider solving the predecessor problem with the least number of queries to a *binary comparison oracle*. For each query index i , the oracle returns whether $s_i \leq t$ or $s_i > t$. In particular, we study the predecessor problem under the UNBOUNDEDNOISY setting, where (i) the search domain \mathcal{S} is *unbounded*, i.e., $n = |\mathcal{S}|$ is unknown or infinite, and (ii) the binary comparison oracle is *noisy*. We denote the former setting by UNBOUNDED and the latter by NOISY. In NOISY, the oracle, for each query, *independently* returns a *wrong* answer with a fixed constant probability $0 < p < 1/2$. In particular, even for two queries on the same index i , the answers from the oracle may be different. Furthermore, with a noisy oracle, the goal is to correctly return the target index with probability at least $1 - Q$, where $0 < Q < 1/2$ is the *failure probability*.

Our first result is an algorithm, called **NoS**, for NOISY that improves the previous result by Ben-Or and Hassidim [FOCS 2008] from an expected query complexity bound to a worst-case bound. We also achieve an expected query complexity bound, whose leading term has an *optimal* constant factor, matching the lower bound of Ben-Or and Hassidim. Building on **NoS**, we propose our **NoSU** algorithm, which correctly solves the predecessor problem in the UNBOUNDEDNOISY setting. We prove that the query complexity of **NoSU** is $\sum_{i=1}^k (\log^{(i)} N) / (1 - H(p)) + o(\log N)$ when $\log Q^{-1} \in o(\log N)$, where N is the target index, $k = \log^* N$, the iterated logarithm, and $H(p)$ is the entropy function. This improves the previous bound of $O(\log(N/Q) / (1 - H(p)))$ by reducing the coefficient of the leading term from a large constant to 1. Moreover, we show that this upper bound can be further improved to $(1 - Q) \sum_{i=1}^k (\log^{(i)} N) / (1 - H(p)) + o(\log N)$ in expectation, with the constant in the leading term reduced to $1 - Q$. Finally, we show that an information-theoretic lower bound on the expected query cost of the predecessor problem in UNBOUNDEDNOISY is at least $(1 - Q) (\sum_{i=1}^k \log^{(i)} N - 2k) / (1 - H(p)) - 10$. This implies the constant factor in the leading term of our expected upper bound is indeed optimal.

2012 ACM Subject Classification Theory of computation \rightarrow Predecessor queries; Theory of computation \rightarrow Sorting and searching

Keywords and phrases Fault-tolerant search, noisy binary search, query complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.25

Funding *Junhao Gan*: J.G. is supported in part by Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) DE190101118.

Anthony Wirth: A.W. is supported by the Faculty of Engineering and Information Technology at The University of Melbourne.

Xin Zhang: X. Z.'s research is supported by an Australian Government Research Training Program (RTP) Scholarship.

Acknowledgements We thank Przemysław Uznański and Dariusz Dereniowski for the discussions on noisy search, particularly for highlighting the nuances of the binary and ternary comparison models.



© Junhao Gan, Anthony Wirth, and Xin Zhang;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 25; pp. 25:1–25:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider a sequence of integers $\mathcal{S} = s_1, s_2, \dots$ sorted in ascending order. The *predecessor problem* for a given integer, t , asks for the index N such that s_N is the largest integer in \mathcal{S} satisfying $s_N \leq t$. We assume s_N always exists. We refer to \mathcal{S} as the *search domain* and to N as the *target index*. The goal of the predecessor problem is to find the target index with the least number of *queries* of a given *binary comparison oracle*, F . Specifically, let $n = |\mathcal{S}|$; F is a function from the *index domain* of \mathcal{S} , i.e., $\{1, 2, \dots, n\}$, to a binary domain $\{1, -1\}$, such that $F(i) = -1$, if $s_i \leq t$, but $F(i) = 1$, if $s_i > t$. Clearly, when n is known and the oracle F always returns correct answers, the well-known binary search algorithm solves the predecessor problem optimally with $\lceil \log_2 n \rceil$ queries.

Arguably, the predecessor problem is one of the most important and classic problems in computer science. Departing from the standard setting, there is much research on variants of binary search [2, 23, 31]. We are particularly interested in two settings of the predecessor problem, UNBOUNDED and NOISY, as well as the combination of them.

The Unbounded Setting. In the UNBOUNDED setting, the size of \mathcal{S} , i.e., n , is either *unknown* or *unbounded*, i.e., infinite. The UNBOUNDED problem was studied by Bentley and Yao [5] and has many applications, e.g., in range-count queries and in shortlisting [16]. Several algorithms solving the UNBOUNDED problem are *result sensitive*: their query complexity does not depend on the (possibly infinite) size of the search domain, but rather the location of the result (i.e., the target index N) in \mathcal{S} , a property desirable in the context of local algorithms [21]. Bentley and Yao [5] wrote the state-of-the-art algorithm, **BY**, with query complexity $\sum_{1 \leq i \leq k} \lceil \log^{(i)}(N) \rceil + 5 + 2k$, where $\log^{(i)} N$ is the nested logarithm and $k = \log^* N$ is the iterated logarithm of N .

The Noisy Setting. In the NOISY setting, the comparison oracle might not always respond correctly, or truthfully: it could return incorrect results. In this case, the comparison oracle is said to be *noisy*. This setting captures the fact that real-world information can be noisy and, hence, the results of comparisons might not be correct. There has been a long line of work for dealing with faulty information or uncertainties in searching problems, from the 1965 Rényi-Ulam game [27] to some more recent work [15, 11, 16]. In this paper, we focus on the *probabilistic error model* [27, 29, 17, 11, 15], where the oracle F behaves as follows.

► **Definition 1.** *For a specified fixed constant error probability p , with $0 < p < 1/2$, in the probabilistic error model, on each query the comparison oracle F independently returns a wrong answer with probability p .*

The goal of the predecessor problem under NOISY is to return the *correct* target index with probability at least $1 - Q$, where $0 < Q < 1/2$ is the *failure probability*.

The UnboundedNoisy Setting. The main focus of this paper is on algorithms solving the predecessor problem in the UNBOUNDEDNOISY setting. This setting combines both UNBOUNDED and NOISY, and UNBOUNDEDNOISY has the following characteristics:

- First, due to the unbounded search domain, the anticipated query complexity is a function of the target index, N , rather than the unknown or unbounded n ;
- Second, inheriting from NOISY, the query complexity should also be related to both the error probability, p , and the failure probability, Q . On the one hand, aligned with previous work [17, 4, 15, 14, 16], parameter p is treated as a constant. On the other hand, as the

target index N depends on the target integer t , it no longer makes sense to aim at “high success probability” in terms of N . We require that $1/Q$ is far less than N ; otherwise, one can turn this problem into the bounded NOISY case by searching the first c/Q elements for some constant c . In particular, we assume throughout that $\log Q^{-1} \in o(\log N)$. As we see shortly, in this case, the *leading* term in the query complexity is $\log_2 N$, which is typically the most important term in the query cost. We hence focus on reducing the hidden constant in such a leading term.

1.1 Our contributions

Bounded search

Our first contribution is an improved algorithm, **NoS**, for the NOISY setting. It achieves the same asymptotic query complexity as the state-of-the-art algorithm by Ben-Or and Hassidim [4]. However, our query bound is more powerful by being *worst-case*, while theirs is in expectation. Comparing the constant factors, when $\log Q^{-1} \in o(\log n)$, the leading term in their query bound is $(1-Q)/(1-H(p)) \log_2 n$, where $H(p) = -p \log p - (1-p) \log(1-p)$ is the well-known entropy function. We show that **NoS** can achieve the same leading term in the query bound, in expectation, but the randomness stems purely from the algorithm mechanism rather than the assumption on the target index distribution. Furthermore, according to the lower bound of Ben-Or and Hassidim [4], the constant factor on this leading term in the query bounds of both their algorithm and **NoS** is indeed optimal.

► **Theorem 2.** *Our NoS algorithm solves the predecessor problem in the NOISY setting, with constant error probability $0 < p < 1/2$, and a failure probability $0 < Q < 1/2$, with worst-case query complexity:*

$$\frac{1}{1-H(p)} \left(\log_2 n + O(\log \log n) + O(\sqrt{\log n \log Q^{-1}} \cdot \log \frac{\log n}{\log Q^{-1}}) + O(\log Q^{-1}) \right). \quad (1)$$

► **Corollary 3.** *When $\log Q^{-1} \in o(\log n)$, **NoS** achieves expected query complexity:*

$$\frac{1-Q}{1-H(p)} (\log_2 n + o(\log n)). \quad (2)$$

According to the lower bound of Ben-Or and Hassidim [4], we have the following:

► **Fact 4** ([4]). *The expected query complexity for solving the predecessor problem under NOISY, parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$, is at least:*

$$\frac{1-Q}{1-H(p)} \log_2 n - 10.$$

Therefore, the constant factor of the leading term in the query complexity of **NoS** is tight, when $\log Q^{-1} \in o(\log n)$. Very recently, and concurrently with the technical development of this paper, Dereniowski et al. [10] also proposed improvements over the results of Ben-Or and Hassidim, achieving a query bound with the same leading term as ours. We compare the two contributions further in Section 2.

Unbounded search

Building upon Theorem 2, our second contribution is a new algorithm, **NoSU**, which improves the query complexity bound for UNBOUNDEDNOISY.

► **Theorem 5.** *Our **NoSU** algorithm solves the predecessor problem in the **UNBOUNDEDNOISY** setting, parameterized by a constant error probability $0 < p < 1/2$ and a failure probability $0 < Q < 1/2$, with worst-case query complexity:*

$$\frac{1}{1 - H(p)} \left(\sum_{i=1}^k \log_2^{(i)} N + O(\log \log N) + O(\sqrt{\log N \log Q^{-1}} \cdot \log \frac{\log N}{\log Q^{-1}}) + O(k \log \frac{k}{Q}) \right),$$

where $k = \log^* N$ is the iterated logarithm of N .

► **Corollary 6.** *When $\log Q^{-1} \in o(\log N)$, **NoSU** has expected query complexity:*

$$\frac{1 - Q}{1 - H(p)} \left(\sum_{i=1}^k \log_2^{(i)} N + o(\log N) \right). \quad (3)$$

Our **NoSU** algorithm significantly reduces the hidden constant factor in the state-of-the-art bound by Epa et al. [16] and Dereniowski et al. [11], i.e., $O(\log(N/Q)/(1 - H(p)))$. The coefficients of the leading terms in these bounds are required to be sufficiently large to invoke the Chernoff bound. In contrast, by Corollary 6, the constant factor in the leading term, $\log_2 N$, of **NoSU**, is just $\frac{1-Q}{1-H(p)}$.

Our final contribution is a lower bound on the expected query complexity for the predecessor problem under **UNBOUNDEDNOISY**.

► **Theorem 7.** *The expected query complexity for solving the predecessor problem under **UNBOUNDEDNOISY**, parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$, is at least:*

$$\frac{1 - Q}{1 - H(p)} \left(\sum_{i=1}^k \log_2^{(i)} N - 2k \right) - 10. \quad (4)$$

Combining Theorem 7 and Corollary 6, the leading term in the expected query complexity of **NoSU** is thus optimal.

1.2 Applications of UnboundedNoisy

The **UNBOUNDEDNOISY** setting naturally produces algorithms whose costs are result-sensitive: the query complexity relies on the target index rather than the size of the search domain. Epa et al. [16] listed a range of applications that solves the predecessor in a noisy setting, including, for instances, counting the number of elements from a sorted list that fall into a range, and obtaining the top-ranked elements from two sorted lists. When applying **NoSU** to bounded domains, **NoSU** is an improvement to the algorithm by Epa et al. [16] in the result-sensitive setting.

The unbounded domain also arises from the context of local algorithms [21], where computing units in a distributed environment have access to local, but not some global information, e.g., the total size of the domain. Since the search domain is distributed, it is hard to discern the total size and run the normal binary search algorithm. Kim and Winston [20] adapted **BY** [5], an unbounded binary search algorithm to the problem maximum power point tracking, processing a large volume of data created from the voltage change in logarithmic time and avoiding a linear scan of the input. Since unbounded domains are often a consequence of large-scale, automatically generated, or distributed datasets, it is natural to consider the comparisons on those data points with the presence of errors [33]. In software

testing, there often is a breakpoint for certain resource, be it time, space or workload [32]. For instance, finding the number of requests of services that break a load-balancer requires a tester to check the status of the software under a range of different request numbers. Due to the randomness commonly existing in those tests, it is reasonable to assume that the search domain (possible breakpoints) is not only unbounded but also noisy.

2 Related Work

Computing with faulty or uncertain information has long been active field of research. An early model that deals with uncertainty is the Rényi-Ulam game [26, 28, 34]. In this two-player game, Player 1’s goal is to identify an element in a finite set by posing questions to Player 2. In answering these questions, Player 2 tries to stop Player 1 from meeting their goal by lying sometimes. One standard model for is *fixed lies* [11, 25, 7, 9], where the number of lies Player 2 can tell is bounded. Another common *error model* is *linearly bounded lies* [13, 1], where the number of lies the adversary is allowed to tell is at all times linear in the number of queries. The error model in this paper is the *probabilistic model* [17], where for each query the adversary (a.k.a., oracle) independently lies with fixed probability. Feige et al. [17] proposed algorithms for not just searching, but also sorting, ranking and merging with the probabilistic model. They refer to this model as the noisy comparison tree: at each “node” (query), the result leads us in two different directions, and the configurations of the query responses naturally form a binary tree. When searching an integer domain, algorithms are often based on binary search [6, 29, 30].

There are well-known connections between these models. We consider the **MWU** algorithm by Dereniowski et al. [12], who studied a graph search framework proposed by Emamjomeh-Zadeh et al. [15]. Though designed for the fixed-lies model, with carefully selected parameters, it could return the correct answer with desirable guarantees under the probabilistic error model, which is the setting of **NOISY**.

Bentley and Yao first introduced the **UNBOUNDED** setting for binary search and the definitive algorithm, **BY** [5], detailed in Section 4. Invoking results from prefix codes, they also established a lower bound for **UNBOUNDED** predecessor search, which is at the heart of our lower bound in Section 6. Their bounds [5] were later improved by Beigel [3] on the non-leading terms. As our analysis focuses on the leading term, we still consider **BY** as the state of the art.

Combining the two settings, **UNBOUNDEDNOISY** was studied by Pelc [24] and then by Aslam and Dhagat [1], who achieved a bound of $O(\log N)$ for the linearly bounded lies model. Dereniowski et al. [11] and Epa et al. [16] recently studied the problem under the same setting as ours, but our algorithm **NoSU** has a better constant on the leading term.

Table 1 displays state-of-the-art results for those problems and the query complexity.

Although the hidden constant on the leading term $\log n$ of **Feige**’s bound is not as small as that of **BH**, a nice feature of **Feige** is that it is a Monte-Carlo algorithm and provides a worst-case query complexity. We incorporate **Feige** as a subroutine in our algorithm, **NoSU**.

Simultaneously with the technical development of our work, Dereniowski et al. [10] applied the Bayesian-update technique, further developing and improving the result of Ben-Or and Hassidim [4], to sorted integers as well as graphs. Similar to **NoS**, their algorithms also improve the previous bounds [17, 4, 10] in **NOISY**. Their algorithm **PŁU** and our **NoS** (stated in Theorem 2) achieve the same query complexity bound in the leading term. Upon closer inspection, **PŁU** is tighter on the second dominating term ($O(\sqrt{\log n \log Q^{-1}})$) than ours. While we are interested in constant error probability, **PŁU** is able to handle the

■ **Table 1** Summary of complexity results for predecessor search in a variety of settings.

| <i>Problem</i> | <i>Algorithm</i> | <i>Upper Bound Complexity</i> | <i>Source</i> |
|----------------|---------------------|--|---------------|
| BOUNDED | BinarySearch | $\lceil \log n \rceil$ | Folklore |
| UNBOUNDED | BY | $\sum_{1 \leq i \leq \log^* N} \lceil \log^{(i)}(N) \rceil + 5 + 2 \log^* N$ | [5] |
| NOISY | BH | Expected $(1 - H(p))^{-1} \cdot (\log n + O(\log \log n) + O(\log Q^{-1}))$ | [4] |
| NOISY | Feige | $(1 - H(p))^{-1} O(\log n / Q)$ | [17] |
| NOISY | PŁU | $(1 - H(p))^{-1} \cdot (\log n + O(\sqrt{\log n \log Q^{-1}}) + O(\log Q^{-1}))$ | [10] |
| UNBOUNDEDNOISY | Unbounded | $(1 - H(p))^{-1} O(\log(N/Q))$ | [11] |
| UNBOUNDEDNOISY | EGW | Expected $(1 - H(p))^{-1} O(\log(N/Q))$ | [16] |

case when p is not a constant. For this work, this is not an issue as we already require that p be a constant. Although both their and our algorithms share similar ideas of selecting items by adjusting the pre-assigned weights, interestingly, the actual details are substantially different: while **PŁU** processes the items in epochs with repeated queries, our **NoS** first adopts the existing **MWU** algorithm as a blackbox and then refines the certain candidates in the subsequent stages. Thanks to this, the proof of **NoS** is relatively simpler. Moreover, **NoS** is also easily incorporated into the UNBOUNDEDNOISY setting which is the main focus of this work.

There is a variety of models, and accompanying algorithms, for solving binary search in noisy conditions; several of these models differ substantially from ours [19, 22, 18, 8]. For instance, Karp and Kleinberg [19] considered a setting where the search is conducted on a sequence of coins, each equipped with a fixed (but distinct) probability of showing head when tossed. The coins are sorted according to their head probabilities. The goal is to find the leftmost coin whose head probability is lower than some target probability, with the least number of coin tosses. This is a more general noisy binary search model than NOISY as the probability of making a mistake varies from query to query. Interestingly, **NoS** almost matches the information-theoretical lower bound obtained by Karp and Kleinberg [19].

3 Bounded Search with Noisy Information

We begin our technical presentation with an introduction to an existing algorithm in NOISY setting, but with a different oracle definition, culminating in our algorithm **NoS**, as part of the proof of Theorem 2.

3.1 Preliminary: A Graph-based Noisy Search Algorithm

We outline the **MWU** (Multiplicative **W**eight **U**ppdate) algorithm [11]¹ in this section, as a preliminary to our **NoS** algorithm. **MWU** is designed for the problem of searching in a graph via queries to an oracle. Since a sorted sequence of integers can be viewed as a *path graph*, **MWU** is also applicable to the predecessor problem. However, **MWU** requires a *ternary oracle* in the probability model, which is a stronger oracle than the binary oracle as discussed earlier. Specifically, for the index domain $[n]$, a *ternary comparison oracle* is

¹ Here and onwards we cite the ArXiv version [11] instead of the conference version [12]: as the authors themselves later noted, the bounds established in the conference version turn out to be inaccurate.

defined as a function F' such that for all $i \in [n]$, $F'(i) = -1$ if $s_i < t$, $F'(i) = 0$ if $s_i = t$, while $F'(i) = 1$ if $s_i > t$. Nonetheless, as we show shortly, we can strengthen the **MWU** algorithm to be applicable with a binary oracle without affecting its query complexity.

To avoid distractions, we describe **MWU** in the context of predecessor problem on a sorted sequence of integers. Let us define some notation first. For an index $i \in [n]$, define its *left set* as $\mathcal{L}_i = [1, i - 1]$ and its *right set* as $\mathcal{R}_i = [i + 1, n]$, each an interval that includes both endpoints. For a query on i , the response $F'(i) = 1$ implies $t \in \mathcal{L}_i$, while $F'(i) = -1$ implies $t \in \mathcal{R}_i$, and $F'(i) = 0$ implies $t = s_i$. For a specified error probability $0 < p < 1/2$, each response is correct, independently, with probability at least $1 - p$. An index j is called *compatible* with the oracle's response for index i if j is in the implied interval², and is called *incompatible* otherwise. We learn the target t by eliminating indices that are incompatible with too many queries, and assess the likelihood of an index being the target via the assignment of weights at each round of the algorithm. For every index $i \in [n]$, let $\mu(i)$ be the weight of i , initialized to be 1. We overload function μ so that for a subset of indices $S \subseteq [n]$, define $\mu(S) = \sum_{i \in S} \mu(i)$, and let $\mu = \mu([n])$, the total weight in array $[n]$. The weighted distance sum of index $i \in [n]$ is

$$\Phi(i) = \sum_{j \in [n]} \mu(j) \cdot |i - j|.$$

Let $q = \operatorname{argmin}_{i \in [n]} \Phi(i)$ be the index that minimizes the weighted distance sum, Φ . Analogous to standard binary search, q is the *median* that we query, and the response suggests eliminating *half* of the potential candidates for the target. In **NOISY**, we choose q so that the nodes preceding q constitute *roughly* half the weight and those following q *roughly* possess half the total weight.

Algorithm **MWU** proceeds in iterations. Initially each index has weight 1. At each iteration, **MWU** queries the comparison oracle at q : it reduces the weight of each *incompatible* index multiplying by factor $1/\Gamma$, for some $\Gamma > 0$ defined later. **MWU** terminates when only one index in $[n]$ has weight at least $1/\Gamma^L$, for some pre-determined integer $L > 0$. This termination condition assumes that the oracle makes at most L mistakes throughout the search, a property of an oracle in the *fixed-lies* error model. However, by setting the parameters appropriately, the fixed-lies error model can be transformed into the probabilistic error model. In particular, Dereniowski et al. [11], show the following:

► **Fact 8.** *Over all iterations of the entire process of **MWU**, the amortized rate of decrease of the total weight is at least $\frac{\Gamma+1}{2\Gamma}$. Specially, if the total weight at the beginning of an iteration is μ , at the end of this iteration, the total weight decreases to $\frac{\Gamma+1}{2\Gamma}\mu$, amortized.*

The main result of **MWU** is as follows.

► **Theorem 9** (Section 3.3 [11]). *Algorithm **MWU** solves the predecessor problem under **NOISY** parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$ with a ternary comparison oracle, with at most*

$$\frac{1}{1 - H(p)} \left(\log_2 n + O(\sqrt{\log n \log Q^{-1}} \cdot \log \frac{\log n}{\log Q^{-1}}) + O(\log Q^{-1}) \right) \quad (5)$$

queries, and the parameters are set as $L = (r \log_2 n)/(1 - H(r))$, $\Gamma = (1 - r)/r$, $r = (1 - \varepsilon_0)/2$, and $\varepsilon_0 = (1 - 2p)/(1 + \sqrt{8 \ln Q^{-1}/\ln n})$.

² Treat $F'(i) = 0$ as $t \in [s_i, s_i]$

In fact, Dereninowski et al. [11] showed that as long as an iterative algorithm can reduce the total weight by an (amortized) rate of $(\Gamma + 1)/(2\Gamma)$, the algorithm satisfies the bound in Expression (5) with the parameters set in Theorem 9. With high probability, it takes that many (Expression 5) iterations to reduce the total weight to an amount such that only one node can possibly have weight more than $1/\Gamma^L$. Unfortunately, Theorem 9 is not directly applicable to the setting with a binary comparison oracle. The binary oracle cannot provide answers to the queries as strong as those a ternary oracle provides. Specifically, for a query index i , a binary oracle can only separate $[n]$ into two parts, $\mathcal{L}_i \cup \{i\}$ and \mathcal{R}_i , rather than three parts: \mathcal{L}_i , $\{i\}$ and \mathcal{R}_i . Hence Fact 8 need not hold for the binary oracle.

3.2 Our Two-Stage Noisy Search Algorithm

In this section, we present our algorithm **NoS** (Noisy Search), to solve the predecessor problem under the NOISY setting with a binary oracle, achieving the same query complexity as **MWU**. Algorithm **NoS** first finds a smaller candidate set that includes the target index and then searches on the set to find the target index. Recall that for a queried index v , the possible compatible sets yielded from a query response are $\mathcal{L}_v \cup \{v\}$ when $F(v) = -1$ and \mathcal{R}_v when $F(v) = 1$. We call the first response that includes the queried index an *inclusive answer* and the second an *exclusive answer*. The basic idea of our **NoS** is that it first collects all the queried indices into a set M as if a ternary oracle is adopted, and then, it further searches the target index in M in the second stage.

■ **Algorithm 1** Algorithm **WeightedBinary** with Γ and L as parameters (defined in Theorem 9).

```

1: function WeightedBinary( $A, L, \Gamma$ )
2:    $M \leftarrow \emptyset$ 
3:   for  $v \in [1, n]$  do  $\mu(v) \leftarrow 1$  and  $l_v \leftarrow 0$ 
4:   while more than one index  $x \in [n]$  has  $l_x \leq L$  do
5:     Query the oracle at  $q \leftarrow \arg \min_{x \in [1, n]} \Phi(x)$ 
6:     if the query response is an inclusive answer then
7:        $M \leftarrow M \cup \{q\}$ ,  $\mu(q) \leftarrow 0$ , and  $l_q \leftarrow L + 1$ 
8:     for all incompatible  $v$  do
9:        $\mu(v) \leftarrow \mu(v)/\Gamma$  and  $l_v \leftarrow l_v + 1$ 
10:   $M \leftarrow M \cup \{x \in [n] : l_x \leq L\}$ 
11:  return  $M$ 

```

Algorithm 1 is the first stage of **NoS**, where a small set of potential targets is identified. We adapt the ternary-oracle algorithm **MWU** in the previous section to our setting. Compared to its ternary counterpart, **WeightedBinary** returns a set M that contains the single index that has sufficiently large weight, *plus* all queried indices for which an inclusive answer is returned. We have:

► **Lemma 10.** *At each iteration, if the total weight at the beginning of the iteration is μ , then **WeightedBinary** reduces the total weight to $\frac{\Gamma+1}{2\Gamma}\mu$.*

Proof. To see this, we compute the rate by which the total weight diminishes at an iteration. It is easy to prove that by the definition of q , $\mu(\mathcal{L}_q) \leq \mu/2$ and $\mu(\mathcal{R}_q) \leq \mu/2$.

At any iteration, consider the response of the binary oracle for a queried node q . Suppose the total weight is μ . Then by definition $\mu = \mu(\mathcal{L}_q) + \mu(\mathcal{R}_q) + \mu(q)$. If the answer is exclusive, then weights in $\mathcal{L}_q \cup \{q\}$ are lowered. Since $\mu(\mathcal{L}_q \cup \{q\})/\Gamma + \mu(\mathcal{R}_q) \leq \frac{\mu}{\Gamma} + \frac{\Gamma-1}{\Gamma}\mu(\mathcal{R}) \leq \frac{\Gamma+1}{2\Gamma}\mu$, the total weight is reduced sufficiently. If the answer is inclusive, then the total weight is lowered by a factor at least $\frac{\Gamma+1}{2\Gamma}$, as here q has its weight reduced to 0. ◀

Observe that the bound in Lemma 10 holds at every iteration, contrasting with Fact 8, which holds in amortization. From Theorem 9 we conclude the following.

► **Theorem 11.** *With suitable parameter settings (as in Theorem 9), for the predecessor problem under NOISY, with $0 < p < 1/2$ and $0 < Q < 1/2$, Algorithm **WeightedBinary** returns a set M containing the target, taking at most β queries, where $|M| \leq \beta + 1$ and*

$$\beta = \frac{1}{1 - H(p)} \left(\log_2 n + O(\sqrt{\log n \log Q^{-1}} \cdot \log \frac{\log n}{\log Q^{-1}}) + O(\log Q^{-1}) \right).$$

The second stage of **NoS** can be conducted with an existing algorithm, e.g., **Feige** [17] whose query complexity is $O(\log(n/Q))/(1 - H(p))$, as shown in Algorithm 2.

■ **Algorithm 2** Algorithm **NoS**.

```

1: function NoS( $A, Q$ )
2:    $Q \leftarrow Q/2$  ▷ For the union bound
3:    $r \leftarrow \left( 1 - \frac{1-2p}{1 + \sqrt{8(\ln Q^{-1})/(\ln n)}} \right) / 2$ 
4:    $L \leftarrow (r \log_2 n)/(1 - H(r))$  and  $\Gamma \leftarrow (1 - r)/r$ 
5:    $M \leftarrow \mathbf{WeightedBinary}(A, L, \Gamma)$ 
6:   Run Feige on  $(M, Q/2)$ 
7:   return the target found

```

It first runs **WeightedBinary** on the input to obtain M , whose size is bounded by $O(\log_2 n/(1 - H(p)))$. Second, it runs **Feige** [17] on M and obtain a target within $(O(\log \frac{\log n}{1 - H(p)}) + O(\log Q^{-1})) / (1 - H(p)) = O(\log \log n + \log Q^{-1}) / (1 - H(p))$ queries. A union bound on the failure probability and summing up these two costs lead to Theorem 2.

3.3 Proof of Corollary 3

To obtain the promised expected query complexity of **NoS**, we apply a familiar trick [4]. Observe that when $\log Q^{-1} \in o(\log n)$, the worst-case query complexity of **NoS** can be written as $\frac{1}{1 - H(p)} (\log_2 n + o(\log n))$. As a result, the “gap” (i.e., the difference) between this worst-case bound and the expected bound, in Expression (2), is $\frac{Q}{1 - H(p)} (\log_2 n + o(\log n))$.

We strengthen **NoS** as follows. On the one hand, when $Q \leq \frac{1}{\log_2 n}$, this gap becomes at most $\frac{1}{1 - H(p)} (1 + o(1)) \in o(\log n)$. In this case, the worst-case query complexity of **NoS** suffices to meet the bound $\frac{1 - Q}{1 - H(p)} (\log_2 n + o(\log n))$. But when $Q > \frac{1}{\log_2 n}$, we perform the following steps. With probability $Q - \frac{1}{\log_2 n}$, we return index 1, and are done. Otherwise, i.e., with probability $1 - Q + \frac{1}{\log_2 n}$, we run **NoS** with a failure probability $Q' = \frac{1}{\log_2 n}$.

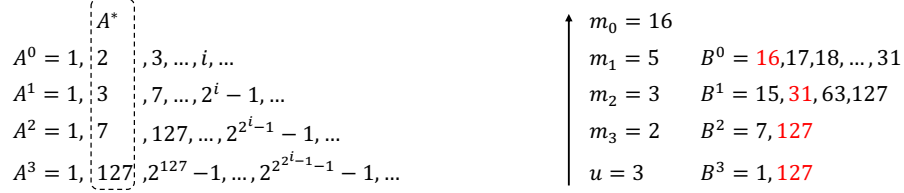
Clearly, the probability of not returning the target index by the above strengthened version of **NoS** is at most $Q - \frac{1}{\log_2 n} + Q' = Q$. Furthermore, in expectation, the query cost is $(1 - Q + \frac{1}{\log_2 n}) \cdot \frac{1}{1 - H(p)} (\log_2 n + o(\log n))$ which is bounded by $\frac{1 - Q}{1 - H(p)} (\log_2 n + o(\log n))$.

4 Unbounded Search Without Noise

In this section we introduce an algorithm for the UNBOUNDED setting by Bentley and Yao [5] (denoted by **BY**), a central plank of our approach to UNBOUNDED. Bentley and Yao express their results, i.e., Lemmas 12, 13 and 14 via some customized functions. Our exposition has more standard notation, via the logarithmic function.

25:10 An Almost Optimal Algorithm for Unbounded Search with Noisy Information

We consider UNBOUNDED for the index domain $A = \{1, 2, \dots\}$, an infinite sequence. We first define a series of subsequences of A . Specifically, put $A^0 \equiv A$, and for $i, j \geq 1$ let $A^i[j] = A^{i-1}[2^j - 1]$. We illustrate the definition of the $\{A^i\}$ in Figure 1's left panel.



■ **Figure 1** Left panel: An illustration of the first four subsequences, A_0, A_1, A_2, A_3 , and A^* . Right panel: an example of running **BY** for target index $N = 16$, with u, m_i and B^i . The arrow indicates the execution order, and the red-colored item indicates the answer to the predecessor query. Quantity m_i is the index of the red-colored item in A^i .

For all $i \geq 0$, define m_i to be the answer to the predecessor query for t on A^i : the goal of UNBOUNDED is to find m_0 . We can also express the value of m_i in terms of N , the target.

► **Lemma 12** (Derived from [5]). *For $1 \leq i \leq k$, where $k = \log^* N$, $m_i \leq \lceil \log^{(i)} N \rceil + 2$.*

Next, the unbounded sequence A^* consists of the *second* item in each sequence A^i . Formally, $A^*[j] = A^j[2]$, for $j \geq 1$. Let u denote the answer to the predecessor query on A^* .

► **Lemma 13** (Derived from [5]). *We have $u \leq k + 1$, where $k = \log^* N$.*

Algorithm 3 outlines the **BY** algorithm. It has two stages: first we find u via a unary search on A^* ; then we repeatedly call **BinarySearch** on suitable subsequences, B^i . For some array R , **BinarySearch**(R) returns the lowest i satisfying $F(i) = 1$, in $\lceil \log |R| \rceil$ queries. Line 6 defines B^{i-1} , a subsequence of A^{i-1} , so that it contains all items that are between the immediate preceding item of the current search result, $A^i[m_i - 1]$, and the current search result, $A^i[m_i]$. Formally, for $1 \leq j < 2^{m_i-1} + 1$, we have $B^{i-1}[j] = A^{i-1}[j + 2^{m_i-1} - 1]$. The intuition of the algorithm is simple: at each iteration, we identify, via binary search, the answer to the predecessor problem for a subsequence of A^i , and the result helps us to “zoom in” to the items contained by an interval of a more fine-grained subsequence, A^{i-1} . The right panel of Figure 1 shows an example of running **BY**.

■ **Algorithm 3 BY.**

```

1: function BY( $A$ )
2:   Find  $u$  by evaluating  $A^*$  linearly
3:    $B^u \leftarrow A^u$ 
4:   for  $i \leftarrow u$  down to 1 do
5:      $m_i \leftarrow \mathbf{BinarySearch}(B^i)$ 
6:      $B^{i-1} \leftarrow$  the subsequence of  $A^{i-1}$  for indexes in  $[2^{m_i-1}, 2^{m_i}]$ 
7:    $m_0 \leftarrow \mathbf{BinarySearch}(B^0)$ 
8:   return  $m_0$ 

```

► **Lemma 14** (Derived from [5]). *Let $k = \log^* N$. The query complexity of algorithm **BY** is $5 + 2k + \sum_{1 \leq i \leq k} \lceil \log^{(i)}(N) \rceil$.*

5 Unbounded Noisy Search

With the help of algorithms **BY** and **NoS**, we can tackle UNBOUNDEDNOISY. Algorithm **NoSU** builds on the idea of an *almost* result-sensitive algorithm [16, Section 3.1].

5.1 An Existing Algorithm

We first describe an existing UNBOUNDEDNOISY algorithm called **Unbounded**, derived from NOISY algorithm **Feige**.

► **Theorem 15** (Theorem C.3 [11]). *For the predecessor problem under UNBOUNDEDNOISY parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$, Algorithm **Unbounded**, with probability at least $1 - Q$, correctly finds the target index. **Unbounded** is built upon an existing NOISY algorithm \mathcal{A} and it uses at least 10 but at most $O(1)$ times as many queries as does \mathcal{A} . When \mathcal{A} is **Feige** [17], **Unbounded** has query complexity $O\left(\frac{1}{1-H(p)} \cdot \log(N/Q)\right)$.*

The **Unbounded** algorithm invokes **Feige** as a blackbox and inherits the same asymptotic query complexity bound. Theorem 15 provides a satisfactory algorithm for solving UNBOUNDEDNOISY. In contrast, **NoSU** achieves a bound with leading term $\frac{1}{1-H(p)} \cdot \log_2(N/Q)$. The hidden constant leading coefficient in **Unbounded** is at least 10, i.e., the query complexity of algorithm **Unbounded** is not as tight as desired. The bound of Epa et al. [16] also includes a large constant, derived from the Chernoff bound.

5.2 Algorithm NoSU

Algorithm 4 details our procedure, **NoSU**. Symbols m_i , u and B^i are the same as in **BY**, defined in Section 4. Similar to **BY**, **NoSU** has stages, one to figure out u , the other to simulate the iterations of B^i . We prove Theorem 5 here.

■ Algorithm 4 NoSU.

```

1: function NoSU( $A, Q$ )
2:   run Unbounded( $A^*, Q/3$ ) to find  $u$ 
3:    $B^u \leftarrow A^u$ 
4:   for  $i$  from  $u$  to 1 do
5:      $m_i \leftarrow \mathbf{NoS}(B^i, Q/(3u))$ 
6:     Let  $B^{i-1}$  be a subsequence of  $A^{i-1}$  for indexes in  $[2^{m_i-1}, 2^{m_i}]$ 
7:    $m_0 \leftarrow \mathbf{NoS}(B^0, Q/3)$ 
8:   return  $m_0$ 

```

Proof of Theorem 5. Clearly, by the union bound, Algorithm 4 correctly finds the target index N with probability at least $1 - Q$.

We now focus on the query complexity of each of the two stages of the algorithm. Line 2 identifies u by calling **Unbounded**($A^*, Q/3$). Recall that $k = \log^* N$. Lemma 13 shows that the target index is u and $k + 1 \geq u$, so Theorem 15 implies that $O\left(\frac{1}{1-H(p)} \cdot \log(k/Q)\right)$ many queries are needed for this step.

For $1 \leq i \leq u$, let $g_i(N, p, Q)$ be the upper bound on the number of queries issued by algorithm **NoS** (Theorem 2) when applied to B^i . For $0 \leq i \leq u - 1$, the size of B^i at iteration i is $2^{m_{i+1}-1}$; and according to Lemma 12, for $1 \leq j \leq k$, $m_j \leq \lceil \log^{(j)} N \rceil + 2$.

25:12 An Almost Optimal Algorithm for Unbounded Search with Noisy Information

Therefore, for $0 \leq i \leq u-1$, $|B^i| = 4 \log^{(i)}(N) - 1$. Also, since $|B^u| = 2$, the cost of calling **NoS** on B^u is a constant. Summing over all $g_i(N, p, Q)$ terms yields

$$\sum_{i=0}^{u-1} g_i(N, p, Q) \leq \sum_{i=0}^k g_i(N, p, Q) = \frac{1}{1-H(p)} (C_1 + C_2 + C_3 + C_4),$$

where $C_1 = \sum_{i=0}^k \log_2^{(i+1)} N$, $C_2 = \sum_{i=0}^k O(\log^{(i+2)} N)$, $C_3 =$

$$\sum_{i=1}^k O\left(\sqrt{\log^{(i+1)} N \log(3k/Q)} \cdot \log \frac{\log^{(i+1)} N}{\log(3k/Q)}\right) + O\left(\sqrt{\log N \log(3/Q)} \cdot \log \frac{\log N}{\log(3/Q)}\right),$$

and $C_4 = \sum_{i=1}^k O(\log(3k/Q)) + O(\log(3/Q))$.

We inspect each of the four terms individually. For the first term, C_1 , we have simply $\sum_{i=1}^{k+1} \log_2^{(i)} N$, which is $\sum_{i=1}^k \log_2^{(i)} N$ plus a small constant. For the second term, C_2 , the quantity folds into $O(\log \log N)$. For the third term, C_3 , with large enough N , we have

$$\sum_{i=1}^k O(\sqrt{\log^{(i+1)} N \log(3k/Q)} + O(\sqrt{\log N \log(3/Q)}) \leq O(\sqrt{\log N \log Q^{-1}}).$$

And for $1 \leq i \leq k$ the multiplicative factor of C_3 satisfies $\log \frac{\log^{(i+1)} N}{\log(3k/Q)} \leq \log \frac{\log N}{\log Q^{-1}}$, and $\log \frac{\log N}{\log(3/Q)} \leq \log \frac{\log N}{\log Q^{-1}}$. The fourth term, C_4 , adds up to $O(k \log(k/Q))$, and it absorbs the cost of **Unbounded**. We thus obtain the bound of Theorem 5. \blacktriangleleft

Although tight, the upper bound in Theorem 5 does not immediately provide a clear perspective on the bound. Consider the worst-case query complexity of **NoSU**. In our setting, we have $\sum_{i=1}^k \log_2^{(i)} N + O(k \log k) < \log_2 N + O(\log \log N)$, so the Theorem 5 bound is at most

$$\frac{1}{1-H(p)} \left(\log_2 N + O(\log \log N) + O(\sqrt{\log N \log Q^{-1}} \log \frac{\log N}{\log Q^{-1}}) + O(k \log Q^{-1}) \right).$$

This is remarkably similar to the bound in Theorem 2. We now prove Corollary 6.

Proof of Corollary 6. Recall that **Unbounded** requires $O((1-H(p))^{-1} \log(k/Q))$ many queries, which resolves to $o(\log N)$ under our setting. For $i \in [0, k]$, the call of **NoS** on B^i only results at most $\log_2^{(i+1)} N + o(\log^{(i+1)} N)$ queries, as $|B^i| = 4 \log^{(i)} N - 1$. The cost of searching in B^u is again a small constant. Corollary 3 hence confirms **NoSU** has expected query complexity $(1-Q)/(1-H(p)) \sum_{i=1}^k \log_2^{(i)} N + o(\log N)$. \blacktriangleleft

6 The Lower Bound

This section is dedicated to prove Theorem 7, a lower bound on the expected query complexity for the predecessor problem under **UNBOUNDEDNOISY**. Let $\bar{\gamma}$ be the expected query complexity of an arbitrary algorithm \mathcal{A} that solves the predecessor problem under **UNBOUNDEDNOISY**, parameterized by $0 < p < 1/2$ and $0 < Q < 1/2$. We lower bound $\bar{\gamma}$.

The idea of the lower bound proof is to reduce to the predecessor problem under **UNBOUNDEDNOISY**, from a well-studied task in information theory. The proof for the **NOISY** setting by Ben-Or and Hassidim [4] is not directly applicable.

First we introduce some basic concepts and terminologies in information theory. In a classic communication problem, we have a transmitter, A , who wants to send some information to a receiver, B . The information is from a *source* set S , and it is sent via a *communication*

channel where only special symbols are allowed to travel, one at a time. In our context, we focus on a *binary channel* that transmits only a binary string bit by bit. A *codeword* w from a binary alphabet is defined as a non-empty binary string, i.e., $w \in \{0, 1\}^+$. A *code scheme* is an injective mapping $c : S \rightarrow \{0, 1\}^+$ that maps an item in the source set S to its unique codeword. The communication channel we consider here is a *noisy binary symmetric memoryless channel with feedback* (BSM-F). The channel is *noisy* if for every bit sent from A to B , upon arrival, it could be flipped (from 1 to 0 or from 0 to 1) independently with certain probability. A channel is *symmetric* if the probability of flipping is the same for the bit 1 and 0, and *memoryless* if the probability also does not vary throughout the transmission, independent to the transmission history before that bit. A communication channel has a *feedback channel* if for every bit A sends, A knows what B receives from the feedback (sometimes called backward) channel. The feedback is useful here as A is adaptive to the noise and can make a decision on the next bit to transmit accordingly.

From the full version of the paper by Ben-Or and Hassidim [4] we have:

► **Fact 16** (Theorem B.1 of full version [4]). *For a fixed flipping probability $0 < p < 1/2$ and a failure probability $0 < Q < 1/2$, consider the communication problem where A sends a message to B over a BSM-F. A bit is received wrongly with probability p , and the message is of length η . Besides, the codeword is correctly received with probability at least $1 - Q$. Let $\tau(\eta)$ be the expected number of bits A has to send to achieve the goal. Then for a large enough η , we have $\tau(\eta) > (1 - Q) \frac{\eta}{1 - H(p)} - 10$.*

Define a communication task *Comm* that requires A to send to B the information, an item of $S = \{1, 2, \dots\}$, over a BSM-F. This naturally induces an UNBOUNDEDNOISY instance with error probability p and failure probability Q . Denote by \mathcal{A} an algorithm that solves the predecessor problem under UNBOUNDEDNOISY. We invoke \mathcal{A} to solve *Comm*, and the lower bound on *Comm* implies a lower bound on the query complexity of \mathcal{A} .

The following construction of the communication protocol between A and B is inspired by the proof of Theorem 2.8 by Ben-Or and Hassidim [4]. B employs Algorithm \mathcal{A} to identify the index of the item in S (which is equivalent to the item itself in this case) that A wants to send; and B also uses the feedback channel to inform A which index it wants to query. Specifically, A sends back a bit 0 if $F(i) = -1$, and a bit 1 if $F(i) = 1$ via the noisy channel to B . The communication is terminated once Algorithm \mathcal{A} decides that the target index is found, i.e., B knows the target item in S . In the above definition of the feedback channel, A only knows the bit B received from the last transmission, i.e., the noisy answer for the latest query. However, we can achieve the same effect by simulating algorithm \mathcal{A} on A as well as on B . As both A and B know the same queried answer, each copy of the algorithm yields the same steps. Conceptually, we can think of the feedback channel is capable of requesting a specific item index to inquire.

As alluded to earlier, the channel handles not directly the index N of the item but a binary code, which we refer to as the codeword of N . Algorithm \mathcal{A} defines a code scheme $c : S \rightarrow \{0, 1\}^+$, where $c(N)$ is the codeword for the index N transmitted in the channel when none of the bits of $c(N)$ is flipped by the noise. Intuitively, when B receives from A successfully the information of N , $c(N)$ is the message that B actually *discovers* from the transmission. B may not receive exactly the bits in $c(N)$ but after correcting errors caused by the channel noise, B should be able to conclude that the right codeword is $c(N)$.

We define a code scheme to connect *Comm* to Theorem 16. Let $\eta_N = |c(N)|$ be the number of bits of $c(N)$. Crucially, as observed by Bentley and Yao [5], the code scheme c determines a prefix code, i.e., $c(i)$ is not a prefix of $c(j)$ for all $i < j$. This is also easy to

verify, because if for some i , $c(i)$ is a prefix of another codeword, then algorithm \mathcal{A} will not know whether to terminate or not when $c(i)$ arrives. They proved the following lower bound on the message length of a prefix-code codeword.

► **Fact 17** (Theorem A, section 3 of [5]). *Let $k = \log^* N$. We have $\eta_N > \sum_{i=1}^k \log_2^{(i)} N - 2k$.*

Our lower bound result (Theorem 7) follows from Fact 16 and Fact 17.

7 Conclusion

In this work, we provide improved algorithms for both NOISY and UNBOUNDEDNOISY settings of predecessor search. For the former, our algorithm **NoS** achieves, in the leading term, query complexity $(\log n)(1 - H(p))^{-1}$ and expected complexity $((1 - Q) \log n)(1 - H(p))^{-1}$. For the latter, our algorithm **NoSU** achieves, in the leading term, query complexity $\sum_{i=1}^k (\log N)(1 - H(p))^{-1}$ and expected complexity $(1 - Q)(\sum_{i=1}^k \log N)(1 - H(p))^{-1}$. Our expected upper bounds also closely match lower bounds. We construct **NoSU** by creatively combining results from the UNBOUNDED and NOISY settings. Our result emphasizes obtaining the best constant on the leading term, particularly in the UNBOUNDEDNOISY setting.

Since our **NoS** algorithm is derived from algorithms designed for graph searches, it would be interesting in future to explore the unbounded setting in graphs and test if our idea can be generalized to more general graphs as well. The error model assumes here that the probability of the oracle making a mistake is the same for all queries. In a more flexible setting, the error probability follows a different distribution, rather than the uniform distribution. As others found [16, 17], our algorithm could potentially be applied as a subroutine to many more problems that rely on the predecessor problem.

References

- 1 Javed A Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 486–493, 1991.
- 2 Paul Beame and Faith E Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.
- 3 Richard Beigel. Unbounded searching algorithms. *SIAM Journal on Computing*, 19(3):522–537, 1990.
- 4 Michael Ben-Or and Avinatan Hassidim. The Bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 221–230, 2008.
- 5 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(SLAC-PUB-1679), 1976.
- 6 Ryan S Borgstrom and S Rao Kosaraju. Comparison-based search in the presence of errors. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 130–136, 1993.
- 7 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms: Reliable Computation with Unreliable Information*. Springer, 2013.
- 8 Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems*, volume 17, pages 337–344, 2005.
- 9 Christian Deppe. Coding with feedback and searching with lies. In Imre Csiszár, Gyula OH Katona, and Gábor Tardos, editors, *Entropy, Search, Complexity*, pages 27–70. Springer, 2007.
- 10 Dariusz Dereniowski, Aleksander Łukasiewicz, and Przemysław Uznański. Noisy searching: simple, fast and correct. *arXiv preprint*, 2021. [arXiv:2107.05753](https://arxiv.org/abs/2107.05753).
- 11 Dariusz Dereniowski, Stefan Tiegel, Przemysław Uznański, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. *arXiv preprint*, 2018. [arXiv:1804.02075](https://arxiv.org/abs/1804.02075).

- 12 Dariusz Dereniowski, Stefan Tiegel, Przemyslaw Uznanski, and Daniel Wolleb-Graf. A framework for searching in graphs in the presence of errors. In *Proc. 2nd Symposium on Simplicity in Algorithms*, volume 69, pages 4:1–4:17, 2019.
- 13 Aditi Dhagat, Peter Gács, and Peter Winkler. On playing “twenty questions” with a liar. In *Proc. 3rd ACM/SIAM Symposium on Discrete Algorithms*, volume 92, pages 16–22, 1992.
- 14 Ehsan Emamjomeh-Zadeh and David Kempe. A general framework for robust interactive learning. In *Proc. 31st International Conference on Neural Information Processing Systems*, pages 7082–7091, 2017.
- 15 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *Proc. 48th ACM Symposium on Theory of Computing*, pages 519–532, 2016.
- 16 Narthana S Epa, Junhao Gan, and Anthony Wirth. Result-sensitive binary search with noisy information. In *Proc. 30th International Symposium on Algorithms and Computation*, pages 60:1–60:15, 2019.
- 17 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
- 18 Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *Advances in Neural Information Processing Systems*, volume 23, pages 766–774, 2010.
- 19 Richard M Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 881–890, 2007.
- 20 Yong Sin Kim and Roland Winston. Unbounded binary search for a fast and accurate maximum power point tracking. In *AIP Conference Proceedings*, volume 1407, pages 289–292, 2011.
- 21 Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. Toward more localized local algorithms: removing assumptions concerning global knowledge. *Distributed Computing*, 26(5-6):289–308, 2013.
- 22 Robert Nowak. Generalized binary search. In *Proc. 46th IEEE Allerton Conference on Communication, Control, and Computing*, pages 568–574, 2008.
- 23 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing*, pages 232–240, 2006.
- 24 Andrzej Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.
- 25 Andrzej Pelc. Searching games with errors – fifty years of coping with liars. *Theoretical Computer Science*, 270(1-2):71–109, 2002.
- 26 Alfréd Rényi. On a problem of information theory. *MTA Mat. Kut. Int. Kozl. B*, 6(MR143666):505–516, 1961.
- 27 Alfréd Rényi. On the foundations of information theory. *Revue de l’Institut International de Statistique*, pages 1–14, 1965.
- 28 Alfréd Rényi and Zsuzsanna Makkai-Bencsáth. *A Diary on Information Theory*. Akadémiai Kiadó Budapest, 1984.
- 29 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.
- 30 Michael Saks and Avi Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. 27th IEEE Symposium on Foundations of Computer Science*, pages 29–38, 1986.
- 31 Pranab Sen. Lower bounds for predecessor searching in the cell probe model. In *Proc. 18th IEEE Conference on Computational Complexity*, pages 73–83, 2003.
- 32 Sanjay Kumar Singh and Amarjeet Singh. *Software testing*. Vandana Publications, 2012.
- 33 John A Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014.
- 34 Stanislaw M Ulam. *Adventures of a Mathematician*. University of California Press, 1991.

Optimal Bounds for Weak Consistent Digital Rays in 2D

Matt Gibson-Lopez ✉ 

Department of Computer Science, The University of Texas at San Antonio, TX, USA

Serge Zamarripa ✉

Department of Computer Science, The University of Texas at San Antonio, TX, USA

Abstract

Representation of Euclidean objects in a digital space has been a focus of research for over 30 years. Digital line segments are particularly important as other digital objects depend on their definition (e.g., digital convex objects or digital star-shaped objects). It may be desirable for the digital line segment systems to satisfy some nice properties that their Euclidean counterparts also satisfy. The system is a consistent digital line segment system (CDS) if it satisfies five properties, most notably the subsegment property (the intersection of any two digital line segments should be connected) and the prolongation property (any digital line segment should be able to be extended into a digital line). It is known that any CDS must have $\Omega(\log n)$ Hausdorff distance to their Euclidean counterparts, where n is the number of grid points on a segment. In fact this lower bound even applies to consistent digital rays (CDR) where for a fixed $p \in \mathbb{Z}^2$, we consider the digital segments from p to q for each $q \in \mathbb{Z}^2$. In this paper, we consider families of weak consistent digital rays (WCDR) where we maintain four of the CDR properties but exclude the prolongation property. In this paper, we give a WCDR construction that has optimal Hausdorff distance to the exact constant. That is, we give a construction whose Hausdorff distance is 1.5 under the L_∞ metric, and we show that for every $\epsilon > 0$, it is not possible to have a WCDR with Hausdorff distance at most $1.5 - \epsilon$.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Digital Geometry, Consistent Digital Rays

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.26

Related Version *Full Version*: <https://arxiv.org/abs/2205.03450>

Funding Supported by the National Science Foundation under Grant No. 1733874.

1 Introduction

In this paper, we consider the digital representation of Euclidean objects. For example, suppose we take a photograph of a convex object O . Since O is convex, we have that the Euclidean line segment connecting any two points inside of O is contained within O . If one wants an image segmentation algorithm to be able to find the pixels in our photograph that correspond to O , we may want the algorithm to be able to ensure that the output pixels are in some sense “convex”. How should we define a set of pixels to be convex? The most natural way would be a similar definition to the Euclidean setting: a set of pixels is a digital convex object if the digital line segment connecting any two pixels is contained within the object. This raises the question of how to define digital line segments.

In particular, consider unit grid \mathbb{Z}^2 where each point in the grid represents a pixel in an infinite image, and in particular consider the unit grid graph: for any two points $p = (p_x, p_y)$ and $q = (q_x, q_y)$ in \mathbb{Z}^2 , p and q are neighbors if and only if $|p_x - q_x| + |p_y - q_y| = 1$. Now consider some pair of grid points p and q ; we would like to define a digital line segment $R_p(q)$ from p to q that is a path in the unit grid graph. There may be multiple “good” ways to digitally represent a Euclidean line segment. For example, consider Figure 1 (a). In isolation, it may seem that either of Figure 1 (b) or (c) would be a fine representation. However when considering it within a family of digital line segments, the choice we make could impact



© Matt Gibson-Lopez and Serge Zamarripa;

licensed under Creative Commons License CC-BY 4.0

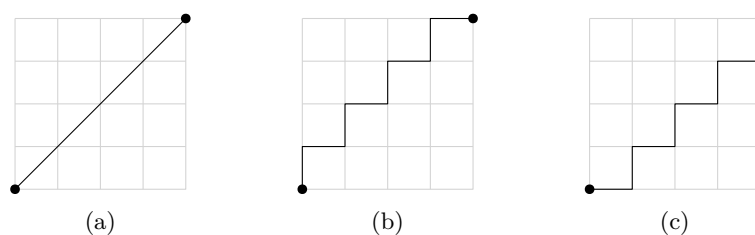
18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 26; pp. 26:1–26:20

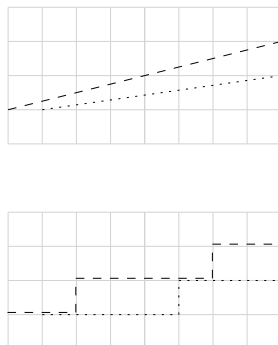
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** (a) Euclidean line to digitize. (b) One option. (c) Another option.



■ **Figure 2** Rounding two Euclidean line segments.

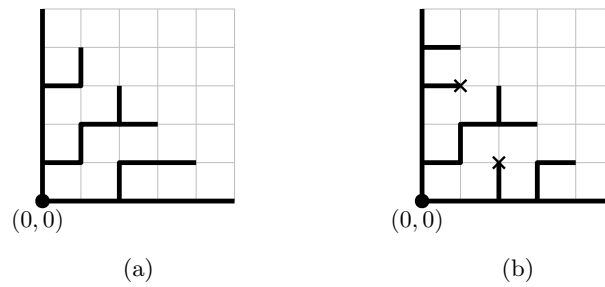
the results for segmentation algorithms that rely on them. In particular, the most simple definitions of digital line segments (“rounding” the Euclidean line segment to the closest pixel) have a potentially undesirable property that their intersections may not be connected. See Figure 2. In scenarios where we are not concerned with individual digital line segments but rather multiple digital line segments (e.g., we are interested in digital convex objects), it may be desirable to consider carefully constructed digital line segment systems that satisfy some nice properties.

For any point $o \in \mathbb{Z}^2$, we call the set of all digital line segments $R_o(p)$ for each $p \in \mathbb{Z}^2$ a *digital ray system* R_o . Intuitively a digital ray system is a family of digital line segments that all share o as a common endpoint. A *digital line segment system* has $R_p(q)$ defined for every $p, q \in \mathbb{Z}^2$.

1.1 Consistent Digital Line Segments

To deal with these issues, past researchers have considered systems of digital rays and digital line segments that collectively satisfy some properties that are also satisfied by their Euclidean counterparts. In particular [7, 6, 4, 5, 1] has considered systems that satisfy the following five properties.

- (S1) *Grid path property*: For all $p, q \in \mathbb{Z}^2$, $R_p(q)$ is the points of a path from p to q in the grid topology.
- (S2) *Symmetry property*: For all $p, q \in \mathbb{Z}^2$, we have $R_p(q) = R_q(p)$.
- (S3) *Subsegment property*: For all $p, q \in \mathbb{Z}^2$ and every $r, s \in R_p(q)$, we have $R_r(s) \subseteq R_p(q)$.
- (S4) *Prolongation property*: For all $p, q \in \mathbb{Z}^2$, there exists $r \in \mathbb{Z}^2$, such that $r \notin R_p(q)$ and $R_p(q) \subseteq R_p(r)$.
- (S5) *Monotonicity property*: For all $p, q \in \mathbb{Z}^2$, if $p_x = q_x = c_1$ for any c_1 (resp. $p_y = q_y = c_2$ for any c_2), then every point $r \in R_p(q)$ has $r_x = c_1$ (resp. $r_y = c_2$).



■ **Figure 3** (a) A CDR for $(0,0)$ that satisfies (S4). (b) A set of digital rays from $(0,0)$ that do not satisfy (S4). In particular, the segments $R_{(0,0)}((1,3))$ and $R_{(0,0)}((2,1))$ do not prolong.

Properties (S2) and (S3) are quite natural to ask for; the subsegment property (S3) is motivated by the fact that the intersection of any two Euclidean line segments is connected, and this property is violated by simple rounding schemes. The prolongation property (S4) is motivated by the fact that any Euclidean line segment can be extended to an infinite line, and we may want a similar property to hold for our digital line segments. While (S1)-(S4) form a natural set of axioms for digital segments, there are pathological examples of segments that satisfy these properties which we would like to rule out. For example, Christ et al. [6] describe a CDS where a double spiral is centered at some point in \mathbb{Z}^2 , traversing all points of \mathbb{Z}^2 . A CDS is obtained by defining $R_p(q)$ to be the subsegment of this spiral connecting p and q . To rule out these CDSes, property (S5) was added.

A digital ray system that satisfies properties (S1)-(S5) is called a *consistent digital ray system* or CDR for short. A digital line segment system that satisfies (S1)-(S5) is called a *consistent digital line segment system* or CDS for short. A CDR R_o (or the rays for a single point in a CDS) can be viewed as a tree rooted at o , where the segment $R_o(p)$ is the unique simple path between o and p in the tree. Note that the segments must be a tree because of property (S3). See Figure 3 (a) for an example CDR for $(0,0)$ to all grid points (x,y) such that $x \geq 0, y \geq 0$, and $x + y \leq 5$. See Figure 3 (b) for an example of digital segments that satisfy all properties except for (S4). Since they still satisfy (S3), the rooted tree perspective still applies to these segments.

Previous Work on CDSes and CDRs

Luby [8] considers *grid geometries* which are equivalent to systems of digital line segments satisfying (S1), (S2), (S5) described in this paper, and various works have considered CDRs and CDSes as defined above. Past work has shown that there are many different CDR and CDS systems, so how should we measure the quality of such a system? Past work usually measures the error of a system by considering the Hausdorff distance of a system. For each grid point $v \in R_p(q)$, the distance from v to the Euclidean line segment \overline{pq} is usually defined to be the Euclidean distance between v and the closest point to v on \overline{pq} . Then the error of $R_p(q)$, which we denote $E(R_p(q))$, is the maximum distance from v to \overline{pq} over all $v \in R_p(q)$. The error of a CDR or a CDS is then defined to be $\sup_{p,q \in \mathbb{Z}^2} \{E(R_p(q))\}$. Chun et al. [7] give an $\Omega(\log n)$ lower bound on the error of a CDR where n is the number of grid points on a digital segment (i.e., $n := |p_x - q_x| + |p_y - q_y|$ for $R_p(q)$) which of course also applies to CDSes. Chun et al. give a construction of CDRs that satisfy the desired properties (S1)-(S5) with a tight upper bound of $O(\log n)$ on the error. Note that the lower bound is due to combining properties (S3), (S4) and (S5). For example, if we are willing to drop property

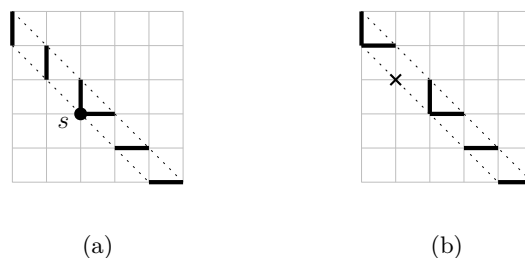
(S3) then digital line segment systems with $O(1)$ error are easily obtained, for example the trivial “rounding” scheme used in Figure 1 (d). This system clearly satisfies (S5), and we can see that it will satisfy (S4) as well. Without loss of generality assume that $p = (0, 0)$. Then $R_p(q)$ will clearly extend to $R_p(r)$ where $r = (2q_x, 2q_y)$ as p , q , and r are co-linear. Chun et al. [7] also show that if (S5) is relaxed, then $O(1)$ error is possible, although they describe the segments as “locally snake-like almost everywhere” but the segments may be acceptable if the resolution of the grid is sufficiently large. Christ et al. [6] extend the upper bound results from CDRs to CDSs by giving an optimal $O(\log n)$ upper bound the error for a CDS in \mathbb{Z}^2 . Chowdhury and Gibson have a pair of papers [4, 5] providing a characterization of CDSes in \mathbb{Z}^2 .

Most of the previous works listed above only apply to two-dimensional grids, but each of the properties (S1)-(S5) have natural generalizations to higher dimensions, and we may be interested in computing CDRs and CDSes for higher dimensions. The construction of Chun et al. [7] for two-dimensional CDRs can be extended to obtain an $O(\log n)$ construction for a CDR in a three-dimensional grid. More recently Chiu and Korman [1] have considered extending the two-dimensional results of [6] to three dimensions, and they show that at times they are able to obtain three-dimensional CDRs with error $\Omega(\log n)$, and even at times they can obtain a three-dimensional CDS, but unfortunately these systems have error $\Omega(n)$.

1.2 Weak Consistent Digital Rays

Suppose we have a CDR R_o in \mathbb{Z}^3 where $o = (0, 0, 0)$, and suppose we consider the two-dimensional “slice” of points $v = (v_x, v_y, v_z)$ such that $v_z = 0$. Now consider the digital rays $R_o(v)$ for each such v . These rays must satisfy (S1), (S2), (S3), and (S5). If any of these properties would be violated then the original CDR system would have to violate the same property; however, the two-dimensional slice does *not* have to satisfy (S4). Indeed, there may be a v with $v_z = 0$ such that the digital segment $R_o(v)$ does not “extend” to any other point v' such that $v'_z = 0$ but instead extends “up” to the point $(v_x, v_y, v_z + 1)$.

This sparked the initial interest in what are called *weak consistent digital rays* (WCDR) where the segments should satisfy all of the CDR properties except (S4). In particular, the $\Omega(\log n)$ error lower bound of [7] for two-dimensional CDRs critically relies upon (S4). Consider a (not-weak) CDR R_o in two-dimensions, and let Q_1 denote the “first quadrant” of o , that is $v \in Q_1$ if and only if $v_x \geq 0$ and $v_y \geq 0$. For any $d \in \mathbb{N}$, let *diagonal* d denote the Euclidean line $x + y = d$. For any $v \in Q_1$, we say that v is *on* diagonal $v_x + v_y$. Consider the “extension” of R_o from the points on diagonal d to the points on diagonal $d + 1$. Chun et al. show that there must be exactly 1 *split point* s on diagonal d such that $R_o(s)$ extends to both $(s_x, s_y + 1)$ and $(s_x + 1, s_y)$. For an example, see Figure 4 (a) which shows the extensions of the CDR from Figure 3 (a) from diagonal 4 to diagonal 5. Then for every point v on diagonal d such that $v_x < s_x$, it only extends vertically to $(v_x, v_y + 1)$, and every point v on diagonal d such that $v_x > s_x$ only extends horizontally to $(v_x + 1, v_y)$. This structure of CDRs helps lead to the error lower bound. In the context of a WCDR, this split point property no longer holds. Instead of picking a single split point on diagonal d that then forces the extension of all other points on diagonal d , we can let each point on diagonal $d + 1$ pick a “parent” point on diagonal d (using the rooted tree perspective of a WCDR), and these parent selections do not need to be coordinated in any way since we do not have the requirement that segments on diagonal d must extend to diagonal $d + 1$. See Figure 4 (b) which shows the WCDR from Figure 3 (b) between diagonals 4 and 5. Points that are not chosen to be a parent are called *inner leaves*. Suppose on diagonal d we have that the number of inner leaves is x . Then it is not difficult to see that there will be $x + 1$ split points



■ **Figure 4** (a) A CDR extension with a single split point s . (b) A WCDR extension with two split points and an inner leaf.

on diagonal d , and when scanning the points on diagonal d from left to right, we alternate between encountering split points and inner leaves, and the first and last will be split points. This difference in structure creates the possibility of having $o(\log n)$ WCDRs, which in turn implies that it may be possible to obtain a (non-weak) CDR in \mathbb{Z}^3 with $o(\log n)$ error.

Chiu et al. [2] considered 2D WCDRs and considered their impact on (non-weak) CDRs in higher dimensions. In particular, they consider the tradeoff between the number of inner leaves a 2D WCDR can have and the error of the system. Indeed if a system does not have any inner leaves, then it is just a regular CDR and the $\Omega(\log n)$ error bound applies. It may be possible to introduce some number of inner leaves to obtain a 2D WCDR with error $o(\log n)$. How many inner leaves do we need to, say, obtain an error of $O(1)$? They show that any WCDR defined for all points $p \in Q_1$ such that $p_x + p_y \leq N$ and has k inner leaves between diagonals $N/2$ and N has error $\Omega(\frac{N \log N}{N+k})$. They then show the impact this has on (non-weak) CDRs in higher dimensions, as every inner leaf on a 2D slice must extend to a point not on this slice which will impact other slices. They use this to show that any CDR in d dimensions has error $\Omega(\log^{\frac{1}{d-1}} N)$. They also consider what is the minimum number of inner leaves needed to obtain a WCDR with error e . In the full version of their paper [3], they give a system with error 2.5 under the L_∞ metric establishing that $O(1)$ error is in fact possible. Note that their $\Omega(\frac{N \log N}{N+k})$ lower bound implies that $k \in \Omega(N \log N)$ in order to achieve $O(1)$ error, but their construction has $k \in \Theta(N^2)$. This leaves open the question as to whether is possible to have a WCDR with $O(1)$ error and $o(N^2)$ inner leaves or if the lower bound could be improved.

1.3 Our Contribution

We consider optimizing the error of a 2D WCDR to the exact constant as we view the WCDR to be of general interest. For some users, the $\Omega(\log n)$ lower bound that comes from including (S3), (S4) and (S5) may be unacceptable. If we wish to achieve $o(\log n)$ error, then we are forced to drop at least one of these properties. For a user who elects to drop (S3), there are plenty of options available that achieve $O(1)$ error and also satisfy (S4) and (S5) (e.g., a greedy rounding strategy). It's possible to only drop (S5) and obtain $O(1)$ error, but a drawback of this system is the “locally snake-like” property that causes the segments to be of different “widths” on different diagonals (e.g, a segment may pass through 1 point on one diagonal but it passes through 3 points on another diagonal). The WCDR is the option for the user who does not want $\Omega(\log n)$ error but wants (S3) and (S5).

Since we want to optimize the exact constant, we need to pick the error metric carefully (i.e., use L_∞ , L_2 , etc.). Chiu et al. used L_∞ metric in their 2.5 error construction, and we believe that the L_∞ metric indeed is the metric that best captures the error of a system. That is, the diagonals 1 to $p_x + p_y$ form a kind of parametrization of \overline{op} , and when picking a point v on diagonal d to be on $R_o(p)$, we argue the goal should be to minimize the distance of v to the intersection of diagonal d and \overline{op} . Let i denote this intersection point. Using L_∞ , the point on \overline{op} that is closest to v will always be i regardless which v on diagonal d we are considering. This does not hold for the L_2 metric where the closest point of \overline{op} to (v_x, v_y) and $(v_x + 1, v_y - 1)$ could be two different points. For this reason, we believe that when being careful with constant factors, it is best to consider L_∞ .

Our Results

In this paper, we give a tight bound on the error of a WCDR in 2D to the exact constant. We prove the following two theorems.

► **Theorem 1.** *There is a WCDR in \mathbb{Z}^2 with error 1.5 in the L_∞ metric.*

► **Theorem 2.** *For every $\epsilon > 0$, there is no WCDR in \mathbb{Z}^2 with error at most $1.5 - \epsilon$ in the L_∞ metric.*

We give a WCDR construction R_o such that for every $p \in \mathbb{Z}^2$, we have that the error of $R_o(p)$ is less than 1.5 in the L_∞ metric, and we show that for any $\epsilon > 0$ that it is not possible to have a WCDR in 2D with error at most $1.5 - \epsilon$ in the L_∞ metric. Essentially, as the length of the segments gets larger, the error of our construction approaches 1.5 but never reaches it, and our lower bound shows that it is not possible to do better than this.

To state our results in the context of the work of Chiu et al. [2], recall they showed that for all segments with length N and k inner leaves between diagonals $N/2$ and N has error $\Omega(\frac{N \log N}{N+k})$. This implies that $\Omega(N \log N)$ inner leaves are required to obtain $O(1)$ error. We remark that for our construction, $k \in \Theta(N^2)$. Our goal was to optimize the error and not minimize the number of inner leaves, but we are not aware of a “simple” way to modify our construction to obtain $o(N^2)$ inner leaves while maintaining $O(1)$ error.

1.4 Organization of the Paper

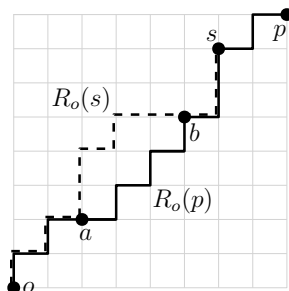
In Section 2, we present our construction with optimal error. In Section 3, we present a high-level overview of our lower bound argument, and we then state the lemmas and theorem that lead to the result.

2 Upper Bound

In this section, we present a WCDR construction with error 1.5. We begin with some preliminaries and definitions.

2.1 Preliminaries

We now define some notation that we will use throughout the paper. Let o be the origin $(0, 0)$. For any point $p \in Q_1$ such that $p \neq o$, let $D(p) := p_x + p_y$ denote the diagonal that p is on, and let $\ell(p)$ denote the Euclidean line through o and p . We define $M(p) := \frac{p_y}{p_x}$ to be the slope of $\ell(p)$ if $p_x > 0$, and otherwise we define $M(p)$ to be ∞ . Let p^\leftarrow , p^\downarrow , p^\rightarrow , and p^\uparrow denote the points $(p_x - 1, p_y)$, $(p_x, p_y - 1)$, $(p_x + 1, p_y)$, and $(p_x, p_y + 1)$ respectively. Let p^\nwarrow



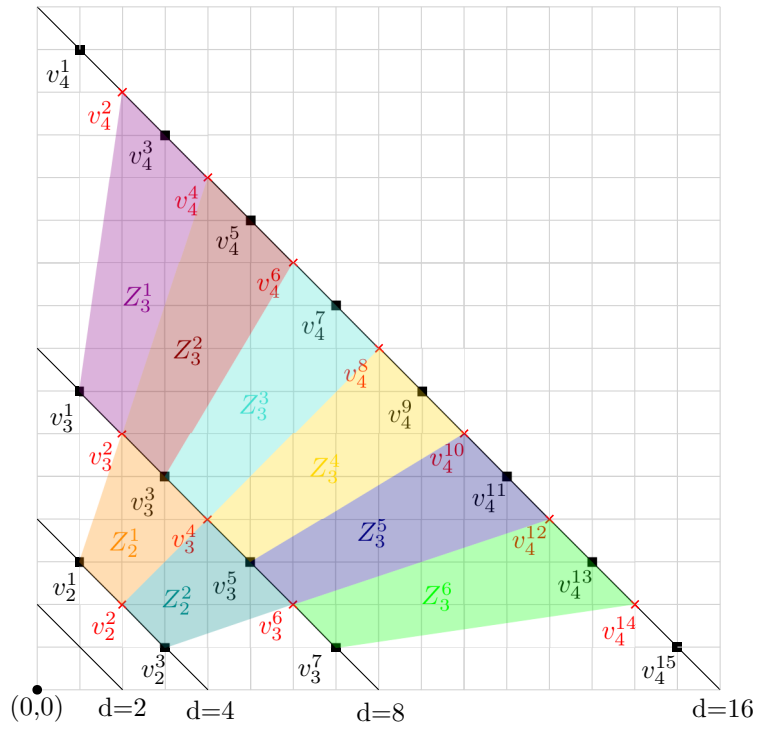
■ **Figure 5** An example of $R_o(s)$ and $R_o(p)$ splitting apart at a and coming back together at b . The dashed line represents $R_o(s)$ and the solid line represents $R_o(p)$.

and p^{\searrow} denote the points $(p_x - 1, p_y + 1)$ and $(p_x + 1, p_y - 1)$ respectively. For two points p and q , we say p is *above* $\ell(q)$ if $M(p) > M(q)$, and we say p is *below* $\ell(q)$ if $M(p) \leq M(q)$ (i.e., we break ties by saying p is below $\ell(q)$). We say that p is *between* $\ell(q)$ and $\ell(q')$ if p is below $\ell(q)$ and is above $\ell(q')$.

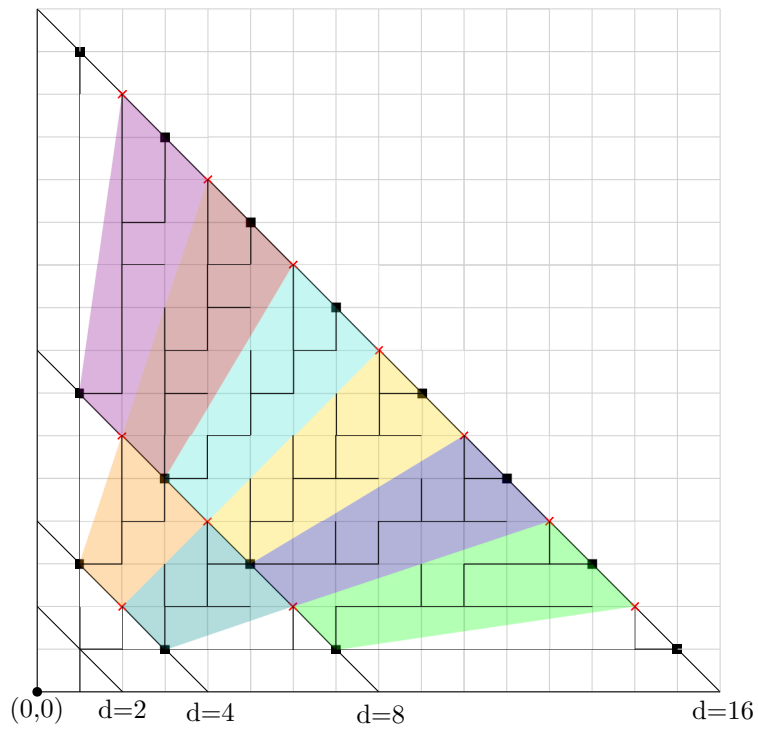
We can view any WCDR as a binary tree rooted at o , and then in this setting each point $v \in Q_1$ such that $v \neq o$ will have a “parent” on diagonal $D(v) - 1$. From this perspective, the following procedure will produce a WCDR R_o in Q_1 . For each point $v \in Q_1 \setminus \{o\}$, if $v_x = 0$ set $v.parent = v^\uparrow$, else if $v_y = 0$ set $v.parent = v^\leftarrow$, else arbitrarily choose one of v^\uparrow and v^\leftarrow to be $v.parent$. Then the digital ray $R_o(p)$ can be computed in “reverse” by starting at p and following the parents back to o . This procedure clearly satisfies (S1), (S2), and (S5). We can show that (S3) will be satisfied with a simple proof by contradiction. Assume there exists an $s \in R_o(p)$ such that $R_o(s) \not\subseteq R_o(p)$. As pointed out in [6] there must be a point a where $R_o(s)$ and $R_o(p)$ split apart for the first time, and a point b when they first come back together, see Figure 5. Note in the figure $b.parent = b^\leftarrow$ in $R_o(s)$ and $b.parent = b^\uparrow$ in $R_o(p)$. However, our procedure only allows b to select one parent for all segments that pass through b , a contradiction. Hence, (S3) holds. Therefore a construction that was obtained from this procedure will certainly produce a feasible WCDR, and then the goal can be to carefully choose the parents so as to minimize the error. After a WCDR for Q_1 is obtained, it can easily be extended to \mathbb{Z}^2 by “mirroring” the construction to the other quadrants. We remark that regular CDRs that satisfy (S4) also can be viewed as a rooted binary tree in this way, but the key difference between the two problems is that if we want (S4), we have to ensure that every point on diagonal $d - 1$ gets picked to be the parent for some point on diagonal d . We have no such restriction when considering WCDRs; points on diagonal $d - 1$ that have no points on diagonal d that pick them to be a parent will be inner leaves.

2.2 The Construction

In this section we describe a method for each point in Q_1 to pick its parent so that the resulting WCDR has error 1.5. We maintain a pattern on each diagonal d such that d is a power of 2 starting with diagonal 4. In particular, let p be a point on diagonal 2^i for an integer $i \geq 2$. Then p will be a split point that will extend to points on diagonal 2^{i+1} if and only if p_x is odd. If p_x is even such that $0 < p_x < 2^i$, then p will be an inner leaf. If $p_x = 0$ or $p_x = 2^i$, then it is on the x or y axis and only extends to the point of diagonal 2^{i+1} that is on the same axis. See Figure 6 where the split points are represented as squares and the inner leaves are represented as crosses.



■ Figure 6 Illustrating the definitions used in the construction.



■ Figure 7 Illustration of our construction of a WCDR with error 1.5 up to $d = 16$.

If p is such that $p_x \leq 1$ or $p_y \leq 1$, then any monotone segment has error less than 1. We will then focus on the points p such that $p_x \geq 2$ and $p_y \geq 2$. For each $i \geq 0$, let C_i denote the points p such that $2^i < D(p) \leq 2^{i+1}$, $p_x \geq 2$, and $p_y \geq 2$. Let v_i^j denote the point $(j, 2^i - j)$ (i.e., the point on diagonal 2^i with x-coordinate j) for each $j \in \{1, \dots, 2^i - 1\}$, and now consider the lines $\ell(v_i^1), \ell(v_i^2), \dots, \ell(v_i^{2^i-1})$. Again suppose that $i \geq 2$ (so that there are at least 3 lines). Then let Z_i^j denote all $r \in \mathbb{R}^2$ such that $2^i < r_x + r_y \leq 2^{i+1}$, r is below $\ell(v_i^j)$, and r is above $\ell(v_i^{j+1})$. We call each Z_i^j a *zone*. Again see Figure 6 where the shaded regions represent the zones. The intuition behind the zones is that they designate which points of C_i we want the split points of diagonal 2^i to extend to. Point v_i^j will be a split point if and only if j is odd. Then we will have that all grid points in Z_i^{j-1} and Z_i^j will have v_i^j as an “ancestor”. More specifically, as v_i^j will be a split point, we will have $(v_i^j)^\uparrow$ and $(v_i^j)^\rightarrow$ will both pick v_i^j as their parent. Then all of the points of Z_i^{j-1} will have $(v_i^j)^\uparrow$ as an ancestor and all the points of Z_i^j will have $(v_i^j)^\rightarrow$ as an ancestor.

Now let $M(i, j, d)$ for some diagonal $d \in \{2^i + 1, \dots, 2^{i+1} - 1\}$ denote the “midpoint” of Z_i^j with respect to diagonal d . That is, it is the point (not necessarily with integer coordinates) on diagonal d whose L_∞ distance to $\ell(v_i^j)$ and $\ell(v_i^{j+1})$ is the same. Also observe that for every diagonal $d \in \{2^i + 1, 2^{i+1}\}$ there is either one or two grid points on diagonal d that is in Z_i^j . This follows from the fact that the distance between $\ell(v_i^j)$ and $\ell(v_i^{j+1})$ is 1 on diagonal 2^i , and therefore for any such d the distance between the lines is greater than 1 (implying there must be at least 1 grid point in Z_i^j on diagonal d). Also on diagonal 2^{i+1} the distance between the lines is 2, and therefore there cannot be 3 or more grid points on the diagonal (since we break ties in the same direction).

Now we are ready to formally state the construction. For each $p \in Q_1$, we use Algorithm 1 to pick its parent. The digital ray $R_o(p)$ (which we will now call $R(p)$ for brevity) then is determined “in reverse” by going from p , $p.parent$, $(p.parent).parent$, etc. until we reach the origin. See Figure 7 for an illustration. As mentioned before, this will certainly produce a feasible WCDR. It remains to argue that the error of the resulting WCDR is 1.5. We remind the reader that the error of the WCDR is the supremum of the errors of all digital rays. In this section we prove that every digital ray in our WCDR has error less than 1.5. Then in the next section we prove that it is not possible to have *any* WCDR with error strictly better than 1.5 which implies that the supremum for our construction is in fact 1.5.

■ **Algorithm 1** pickParent(p).

-
- 1: **if** $p_x \leq 1$ **and** p is not $(1, 0)$ **then**
 - 2: $p.parent = p^\downarrow$
 - 3: **else if** $p_y \leq 1$ **then**
 - 4: $p.parent = p^\leftarrow$
 - 5: **else if** p_y is 2 **and** $D(p)$ is a power of 2 **then**
 - 6: $p.parent = p^\downarrow$
 - 7: **else if** $D(p) - 1$ is a power of 2 **then**
 - 8: Set $p.parent$ to be whichever of p^\downarrow and p^\leftarrow has odd x-coordinate.
 - 9: **else**
 - 10: Let Z_i^j be the zone that p belongs to. Set $p.parent$ to be whichever of p^\downarrow and p^\leftarrow is closest to $M(i, j, D(p) - 1)$, breaking ties arbitrarily.
-

► **Lemma 3.** *The WCDR produced by Algorithm 1 is such that for every $p \in Q_1$, $R(p)$ has error less than 1.5 in the L_∞ metric.*

26:10 Optimal Bounds for Weak Consistent Digital Rays in 2D

Proof. Let p be any point in Q_1 . If $p_x \leq 1$ or $p_y \leq 1$ then trivially any monotone digital ray will have error less than 1, so let us consider the points p such that $p_x \geq 2$ and $p_y \geq 2$.

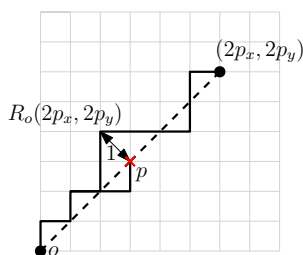
In this paragraph, we handle the case where p does not have a zone. The only such points p that do not have a zone are the points p such that $D(p)$ is a power of 2 (at least 4) and p_y is 2 (this is due to the fact that we break ties by rounding down, and the region below these points are not in any zone). Consider “walking” along $R(p)$ for these p starting at p and walking back towards o . The segment moves vertically once, then moves horizontally until we reach $(1, 1)$, then moves vertically once, then finally horizontally once to reach o . Clearly this segment has error less than 1 for all such p .

For the remainder of the proof, assume that p does have a zone. At a high level, we show that $R(p)$ will only contain grid points that are in zones that are intersected by $\ell(p)$ (until we reach a grid point with some coordinate that is 1). This gets us most of the way there, but zones have diagonal widths that approach 2 as we reach the next power of 2 diagonal from the origin. This means that it could be possible to have that $R(p)$ only contains grid points that are in zones intersected by $\ell(p)$ and yet the error approaches 2. We will show that given our construction, the error of $R(p)$ will in fact be less than 1.5.

For every $q \in R(p)$ such that $q_x > 1$ and $q_y > 1$, $\ell(p)$ intersects the zone of q . Let Z_i^j denote the zone that p belongs to. We first argue that when walking from p to o along $R(p)$, the segment “stays inside” Z_i^j until we reach diagonal 2^i . This can be argued inductively: take any $q \in R(p)$ that is in Z_i^j . If $D(q) = 2^i + 1$ then we are done, so suppose $D(q) > 2^i + 1$. Then q picked its parent by choosing whichever of q^\downarrow and q^\leftarrow is closest to the midpoint of Z_i^j . Since there must be at least one grid point on diagonal $D(q) - 1$ in Z_i^j , it must be that at least one of q^\downarrow and q^\leftarrow is in Z_i^j . Indeed, if q^\downarrow is above $\ell(v_i^j)$ or q^\leftarrow is below $\ell(v_i^{j+1})$ then q would not be in Z_i^j , and if q^\leftarrow is above $\ell(v_i^j)$ and q^\downarrow is below $\ell(v_i^{j+1})$ then there would not be any grid point on $D(q) - 1$ in Z_i^j . So q has at least one parent option in Z_i^j , and a point in Z_i^j clearly must be closer to the midpoint than a point outside Z_i^j , and therefore $q.parent$ will be in Z_i^j .

Now consider the first point we encounter on $R(p)$ that is not in Z_i^j when walking along $R(p)$ towards o . This point is either v_i^{j+1} or v_i^j (depending on if j is odd or even). If this point has some coordinate that is 1, then we are done, so suppose it doesn’t and therefore has a zone. In the case where j is even, then we reach v_i^{j+1} “from above”. Note that the “top line” of v_i^{j+1} ’s zone is the same as the “top line” of p ’s zone (the points used to define the respective lines have the same slope). Therefore $\ell(p)$ will intersect v_i^{j+1} ’s zone. Symmetrically, when j is odd then we reach v_i^j “from the right”, and in this scenario we have that $\ell(v_i^{j+1})$ is the same line as the “bottom line” of v_i^j ’s zone, and therefore $\ell(p)$ intersects v_i^j ’s zone. We then can apply these arguments inductively to see that $R(p)$ will stay inside the zones intersected by $\ell(p)$ until we reach a point with some coordinate that is 1.

If $R(p)$ contains a point q that is distance at least 1.5 to one of the lines ℓ defining its zone, then every q' such that $D(q) < D(q') \leq D(p)$ is in the same zone and is distance at least 1.5 from ℓ . Consider such a q , and let Z_i^j denote the zone of q . Note that by the above argument, it must be that $\ell(p)$ intersects Z_i^j . If q is p then we are done, so suppose q is not p . Then some point $q' \in R(p)$ picked q to be its parent. We will first argue that q' must also be in Z_i^j . Of course it must be in a zone intersected by $\ell(p)$, and therefore it cannot be in Z_i^{j-1} or Z_i^{j+1} . Moreover the only point in Z_i^j that is chosen to be the parent of a point in a different zone is the split point on diagonal 2^{i+1} , but this point is distance 1 to both lines defining the zone, and therefore cannot be q . This implies that q' must also be in Z_i^j .



■ **Figure 8** Illustrating for any inner leaf p the segment $R_o(2p_x, 2p_y)$ must have error of at least 1.

Now suppose without loss of generality we have that the distance from q to $\ell(v_i^{j+1})$ is at least 1.5. We will show that the distance from q' to $\ell(v_i^{j+1})$ is also at least 1.5. We will do this by showing that q' must be q^\uparrow . Note that when q' picked q as its parent, it must have done so in the last line of Algorithm 1 since $D(q)$ cannot be a power of 2. Since the distance from q to $\ell(v_i^{j+1})$ is at least 1.5 it must be that q^\searrow is also in Z_i^j . Moreover since the distance between $\ell(v_i^j)$ and $\ell(v_i^{j+1})$ is less than 2 on $D(q)$ it must be that the distance from q to $\ell(v_i^j)$ is less than 0.5, which implies that the distance from q^\searrow to $\ell(v_i^j)$ is less than 1.5. So which point could have picked q as its parent? It could not have been q^\rightarrow , as it would have preferred q^\searrow as its parent over q . Therefore it must have been q^\uparrow that picked q as its parent, and q^\uparrow will be farther from $\ell(v_i^{j+1})$ than q .

Putting it all together. Now consider $R(p)$, let Z_i^j denote p 's zone, and consider any point $q \in R(p)$. We will argue that the distance from q to $\ell(p)$ is less than 1.5 on $D(q)$. If q is distance at least 1.5 from one of the lines for its zone, then by the previous argument we have that p is in the same zone as q and is distance more than 1.5 from the same line. Since the “width” of the zone at $D(q)$ is less than 2, this implies that the distance from q to $\ell(p)$ on $D(q)$ is less than 0.5. So now assume that q is less than 1.5 to both lines of its zone. Since we know that $\ell(p)$ intersects the zone of q , it directly follows that the distance from q to $\ell(p)$ on $D(q)$ is less than 1.5. ◀

3 Lower Bound

In this section, we prove Theorem 2 which implies that Algorithm 1 produces a WCDR with an optimal error of 1.5. Due to lack of space, we will only sketch the proof.

3.1 Trivial Lower Bounds

There is no known previous work in obtaining a lower bound. There is a trivial lower bound of 0.5 for the segment $R((1, 1))$. There are two options for the segment (pass through either $(0, 1)$ or $(1, 0)$), and both of them have an error of 0.5. We can also obtain a fairly easy lower bound of 1 by considering the effects of inner leaves. If a WCDR does not have any inner leaves, then it satisfies (S4) as well and therefore is actually a CDR and therefore has $\Omega(\log n)$ error, so consider an WCDR, R_o , that has an inner leaf p . We can show that it must have an error of at least 1 in the following way. Consider the segment $R_o((2p_x, 2p_y))$. Trivially, $\ell(2p_x, 2p_y)$ passes through p , but by assumption p is an inner leaf and therefore $R_o((2p_x, 2p_y))$ must pass through a point on $D(p)$ that is either “above” p or “below” p , see Figure 8. Hence, any WCDR must have error of at least 1.

So obtaining a lower bound of 0.5 is trivial, improving it to 1 is fairly simple, but improving it to 1.5 (which is tight given our construction) is more technical.

3.2 Tight Lower Bound Preliminaries

We begin with some definitions that we use in the lower bound proof. Fix any WCDR in \mathbb{Z}^2 . Recall we are proving that for any $\epsilon > 0$, it is not possible for the WCDR to have error at most $1.5 - \epsilon$. For a sufficiently large integer N (which depends on ϵ), we will “cut off” the WCDR at diagonal N , obtaining a finite WCDR that is sufficient for proving the lower bound. This allows us to use maximums and minimums in our definitions rather than supremums and infimums. Similar to the previous section, we only consider the first quadrant Q_1 of o . We say that a point v extends to diagonal $d > D(v)$ if there is some point p with $D(p) = d$ such that $v \in R(p)$. Similarly, if we say that v extends to p if $v \in R(p)$. We let $Subtree(v)$ denote the set of all grid points that v extends to. For any subset S of the grid points, we define $Cone(S)$ to denote all points that are between $\ell(t)$ and $\ell(r)$, where t is the point in S with maximum slope, and r is the point in S with minimum slope. We define $ConeWidth(S, d)$ to be the distance between $\ell(t)$ and $\ell(r)$ on diagonal d . For a split point s , we call $Subtree(s^{\rightarrow})$ the *bottom branch* of s , and we call $Subtree(s^{\uparrow})$ the *top branch* of s .

We will use the following lemma in our lower bound proof.

► **Lemma 4.** *Suppose we have a WCDR in \mathbb{Z}^2 with error less than 1.5, and further suppose there is a point v that does not extend to diagonal d for some $d > D(v)$. Then v^{\nwarrow} extends to diagonal d (if it is in Q_1), and v^{\swarrow} extends to diagonal d (if it is in Q_1).*

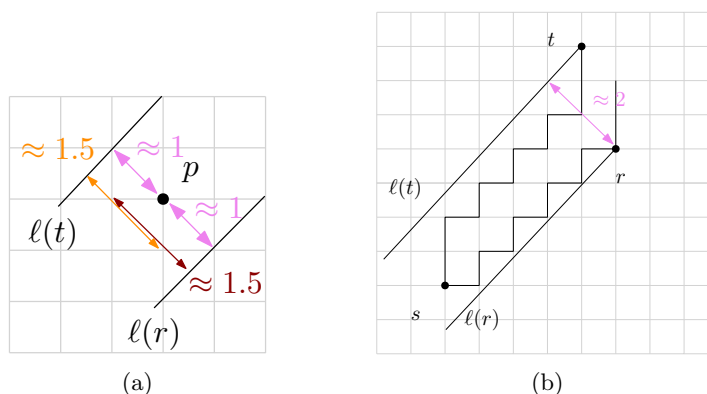
Proof. Suppose the contrary. Without loss of generality, suppose that v^{\swarrow} also does not extend to diagonal d . Then on some diagonal $d' > d$ there will be a grid point p such that $M(p) = \frac{M(v) + M(v^{\swarrow})}{2}$ (because the slopes are rational). Then v and v^{\swarrow} are both distance 0.5 to $\ell(p)$, but $R(p)$ cannot contain v or v^{\swarrow} . Therefore no matter which point from diagonal $D(v)$ is on $R(p)$, we must have that the error at that point is at least 1.5, a contradiction. ◀

3.3 Tight Lower Bound Proof Sketch

We now give a high level overview of our lower bound proof. Suppose we have any WCDR construction with error at most 1.5. We will show that there is some point $p \in \mathbb{Z}^2$ such that

1. $p_y > p_x$ but $M(p)$ is “very close” to 1,
2. $ConeWidth(Subtree(p), D(p))$ is “very close” to 2,
3. and p is “very close” to the center of $Cone(Subtree(p))$ on diagonal $D(p)$.

Suppose we can show the existence of such a point p . We will show that this implies that no matter which point p picks as its parent, the error of that choice will be “very close” to 1.5. See Figure 9 (a). Let t be the point in $Subtree(p)$ with maximum slope, and let r be the point in $Subtree(p)$ with minimum slope. From the assumptions on p , we have that $\ell(t)$ is a distance of close to 1 “above” p on $D(p)$, and $\ell(r)$ is a distance of close to 1 “below” p on $D(p)$. Moreover $M(p)$ is “very close” to 1 (which implies that $M(t)$ and $M(r)$ also are “very close” to 1 if $D(p)$ is “sufficiently large”). Whichever point p picked as its parent will be on $R(t)$ and $R(r)$. But if p picks p^{\leftarrow} as its parent, p^{\leftarrow} will have a distance that is “very close” to 1.5 from $\ell(r)$. If p picks p^{\downarrow} as its parent, p^{\downarrow} will have a distance that is “very close” to 1.5 from $\ell(t)$. Therefore the WCDR will have to have an error that is “very close” to 1.5 no matter which choice p made.



■ **Figure 9** (a) p satisfies the three properties above and no matter which parent it picks, the error will be close to 1.5. (b) $ConeWidth(Subtree(s), D(r))$ is close to 2. $D(r) - D(s)$ will be sufficiently small so that $ConeWidth(Subtree(s), D(s))$ is also close to 2.

Now to show that for any $\epsilon > 0$ that there is no WCDR with error at most $1.5 - \epsilon$, we pick a δ such that $0 < \delta < \min\{\epsilon, 0.1\}$ and we show that such a point p exists where “very close” is a function of δ . Then the above analysis will show that no matter which point p picks as its parent, the error will have to be at least $1.5 - \delta > 1.5 - \epsilon$.

We show that p exists by showing that there must be a split point s that satisfies:

- $1 + \frac{2\delta}{3-2\delta} < M(s) < 1 + \frac{4\delta}{3-2\delta}$
- $ConeWidth(Subtree(s), D(s))$ is “very close” to 2.

Since $M(s) > 1$, eventually it will either have to make two consecutive vertical extensions in its bottom branch, or it will have to have a split point in $Subtree(s)$. We can show that this will imply that $ConeWidth(Subtree(s), D(s))$ is “very close” to 2. For example, see Figure 9 (b) which shows a split point with two consecutive vertical extensions in its bottom branch. It then follows that s must extend to the points t and r , where r is the point at the “bottom” of the double vertical extension in the bottom branch of s , and t is $((r^\uparrow)^\uparrow)^\leftarrow$. Since $M(s), M(r)$ and $M(t)$ are all “very close” to 1, it follows that $ConeWidth(Subtree(s), D(r))$ is “very close” to 2, and we in fact will show that $ConeWidth(s, D(s))$ remains “very close” to 2 as $D(r)$ is sufficiently close to $D(s)$. Recall we are looking for a point p with 3 properties. Certainly s satisfies both properties (1) and (2) above, but it may not satisfy (3). But we will show that we do not have to go “too many” diagonals before $D(s)$ before we must find a point $p \in R(s)$ that is close to the center of its cone and still satisfies (1) and (2). In order to show this last part, we need $D(s)$ to be sufficiently large.

3.4 Formal Proof

For any $\epsilon > 0$, fix any δ such that $0 < \delta < \min\{\epsilon, 0.1\}$. The following lemma proves the existence of the split point s as we described in the proof sketch. The proof appears in Appendix A.

► **Lemma 5.** *For any WCDR in \mathbb{Z}^2 with error less than 1.5 and any $0 < \delta \leq 0.1$ there is a split point s that satisfies the following:*

1. For every $p \in Subtree(s)$, $1 + \frac{2\delta}{3-2\delta} < M(p) < 1 + \frac{4\delta}{3-2\delta}$
2. $ConeWidth(Subtree(s), D(s) + \frac{7+\delta}{\delta}) > 2 - \frac{2\delta}{3}$
3. $\frac{63}{\delta^2} \leq D(s) \leq \frac{382}{\delta^2 - \frac{2}{3}\delta^3}$

Using the existence of s from Lemma 5, we can prove the lower bound. We show that given such an s , there must be a point p as described in the proof sketch. Then it is the case that p is on a segment whose error is at least $1.5 - \delta > 1.5 - \epsilon$. The proof is omitted due to lack of space.

References

- 1 Man-Kwun Chiu and Matias Korman. High dimensional consistent digital segments. *SIAM J. Discret. Math.*, 32(4):2566–2590, 2018. doi:10.1137/17M1136572.
- 2 Man-Kwun Chiu, Matias Korman, Martin Suderland, and Takeshi Tokuyama. Distance bounds for high dimensional consistent digital rays and 2-d partially-consistent digital rays. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPICs*, pages 34:1–34:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.34.
- 3 Man-Kwun Chiu, Matias Korman, Martin Suderland, and Takeshi Tokuyama. Distance bounds for high dimensional consistent digital rays and 2-d partially-consistent digital rays. *CoRR*, abs/2006.14059, 2020. arXiv:2006.14059.
- 4 Iffat Chowdhury and Matt Gibson. A characterization of consistent digital line segments in \mathbb{Z}^2 . In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 337–348, 2015. doi:10.1007/978-3-662-48350-3_29.
- 5 Iffat Chowdhury and Matt Gibson. Constructing consistent digital line segments. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2016. doi:10.1007/978-3-662-49529-2_20.
- 6 Tobias Christ, Dömötör Pálvölgyi, and Milos Stojakovic. Consistent digital line segments. *Discrete & Computational Geometry*, 47(4):691–710, 2012. doi:10.1007/s00454-012-9411-y.
- 7 Jinhee Chun, Matias Korman, Martin Nöllenburg, and Takeshi Tokuyama. Consistent digital rays. *Discrete & Computational Geometry*, 42(3):359–378, 2009. doi:10.1007/s00454-009-9166-2.
- 8 Michael G. Luby. Grid geometries which preserve properties of euclidean geometry: A study of graphics line drawing algorithms. In Rae A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume 40, pages 397–432, 1988.

A Proof of Lemma 5

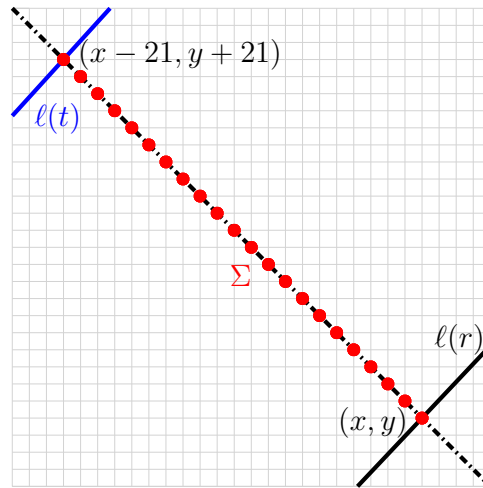
Our proof will use the following observations.

► **Observation 6.** *Given a set of points, S , the number of grid points between $\text{Cone}(S)$ on diagonal d must be either $\lfloor \text{ConeWidth}(S, d) \rfloor$ or $\lfloor \text{ConeWidth}(S, d) \rfloor + 1$.*

Let $\text{Inter}(\ell(p), d)$ be the point $\ell(p)$ intersects diagonal d .

► **Observation 7.** *Given a point p and a diagonal d , the x -coordinate of the $\text{Inter}(\ell(p), d)$ is $d \frac{p_x}{p_x + p_y}$, and the y -coordinate of $\text{Inter}(\ell(p), d)$ is $d \frac{p_y}{p_x + p_y}$.*

Fix any $0 < \delta \leq 0.1$. Suppose there exists a WCDR with error at most $1.5 - \delta$. We prove the lemma by first showing how to pick the split point s from the WCDR, and then we show that this choice of s must satisfy the three conditions of Lemma 5.

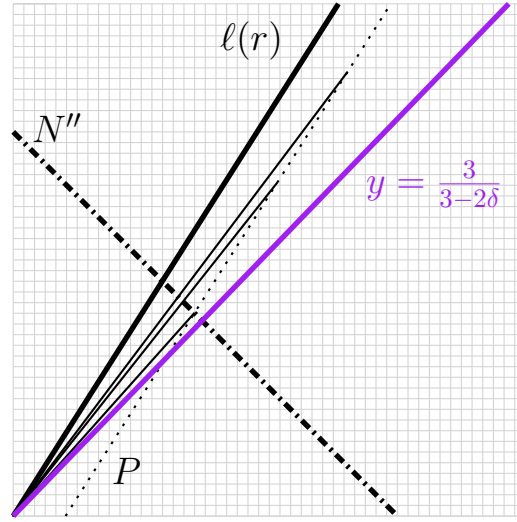


■ **Figure 10** An illustration of the consecutive grid points in Σ .

Picking the split point s . Let $x := \lceil \frac{200}{\delta^2} \rceil$, $y := x + \lceil \frac{600-182\delta}{\delta(3-2\delta)} \rceil$, and $N := x+y$. Let Σ denote the subset of consecutive grid points along diagonal N from the grid point $(x-21, y+21)$ to (x, y) . Let $\ell(t)$ and $\ell(r)$ be the Euclidean lines as previously defined for $Cone(\Sigma)$. That is t is the point in Σ with maximum slope, and r is the point in Σ with minimum slope. See Figure 10. We will first show that (3) from Lemma 5 must be satisfied. Let $N' := \lceil 20\frac{N}{21} \rceil$, and let Σ' denote the set of points between $\ell(t)$ and $\ell(r)$ on N' . Notice that the width of $Cone(\Sigma)$ increases by 1 every $\frac{N}{21}$ diagonals. That is to say that $ConeWidth(\Sigma, \frac{20}{21}N) = ConeWidth(\Sigma, N) - 1 = 20$. It follows that $ConeWidth(\Sigma, N') \geq 20$ which implies from Observation 6 that $|\Sigma'| \geq 20$.

We will lower bound the number of points in Σ' that must extend to a point in Σ . Let p be a point in Σ' . There are three cases for why p does not extend to a point in Σ . The first is that p extends to a point above $\ell(t)$, the second is p extends to a point below $\ell(r)$, and the third is that p does not extend to N . Let us consider the first case. Let p extend to a point q , such that $q \notin \Sigma$, $D(q) = N$, and q is above $\ell(t)$. Consider the point $w := (x-22, y+22)$. Notice that w^\searrow is on $\ell(t)$ and $D(w) = N$. It follows that $M(q) \geq M(w)$.

We would like to know the difference in x-coordinates when $\ell(w)$ and $\ell(w^\searrow)$ intersect N' . Recall that $N' = \lceil \frac{20}{21}N \rceil$, which implies that $N' \geq \frac{20}{21}N$. We know that the difference in x-coordinates of w^\searrow and w is 1. It then follows that the difference in x-coordinates of $Inter(\ell(w^\searrow), N')$ and $Inter(\ell(w), N')$ is at least $\frac{20}{21}$. Therefore error of $R(q)$ is at least the difference between p_x and the x-coordinate of $Inter(\ell(w^\searrow), N')$ plus $\frac{20}{21}$. It follows that p can only be the point with the largest slope in Σ' , otherwise error for $R(q)$ is greater than $1 + \frac{20}{21}$. See Figure 11. We now analyze case two with a similar argument. Let $l := (x+1, y-1)$. If p extends to a point on N that is below $\ell(r)$, then error of $R(q)$ is at least the difference between the x-coordinate of $Inter(\ell(l^\swarrow), N')$ and p_x plus $\frac{20}{21}$. It then follows that p can only be the point with the smallest slope in Σ' , otherwise error for $R(q)$ is greater than $1 + \frac{20}{21}$. Therefore there are only two points in Σ' that can extend to a point on N that is not in Σ . It follows that there are at least $20 - 2 = 18$ points in Σ' that either extend to a point in Σ or do not extend to N . If a point $p \in \Sigma'$ does not extend to N then from Lemma 4 it must be that p^\swarrow and p^\searrow extend to N . It then follows that the number of points that must extend to Σ is at least $\frac{18}{2} = 9$.

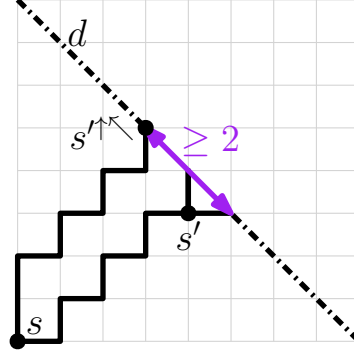


■ **Figure 13** An illustration of P under $\ell(r)$. The Euclidean line with slope $\frac{3}{3-2\delta}$ is represented by the purple line on the bottom.

Next we want to show that $N' \leq \frac{382}{\delta^2 - \frac{2}{3}\delta^3}$ by showing that $\lceil \frac{20}{21}N \rceil \leq \frac{382}{\delta^2 - \frac{2}{3}\delta^3}$. We have $N = 2\lceil \frac{200}{\delta^2} \rceil + \lceil \frac{600-182\delta}{\delta(3-2\delta)} \rceil \leq \frac{400}{\delta^2 - \frac{2}{3}\delta^3}$ for all $\delta \in (0, 0.1]$, and $\lceil \frac{20}{21}(\frac{400}{\delta^2 - \frac{2}{3}\delta^3}) \rceil \leq \frac{382}{\delta^2 - \frac{2}{3}\delta^3}$. Thus $\lceil \frac{20}{21}N \rceil \leq \frac{382}{\delta^2 - \frac{2}{3}\delta^3}$ which implies $N' \leq \frac{382}{\delta^2 - \frac{2}{3}\delta^3}$. This concludes the proof of (3) from Lemma 5.

Proving s satisfies (1). While it is true that both s^\uparrow and s^\rightarrow must extend to at least one grid point in Σ , it is possible that they also extend to grid points outside of Σ on N . However, any $\ell(p)$, such that $p \in \text{Subtree}(s)$, must cross $D(s)$ at most $1.5 - \delta$ above or below s . Moreover, since s extends to at least two grid points in Σ it must be that s can be at most $1.5 - \delta$ above $\ell(t)$ or at most $1.5 - \delta$ below $\ell(r)$. It follows that any $\ell(p)$, such that $p \in \text{Subtree}(s)$, must cross $D(s)$ at most $3 - 2\delta$ above $\ell(t)$ or at most $3 - 2\delta$ below $\ell(r)$. Recall that we just showed that (3) from Lemma 5 is true. Therefore to prove (1) from Lemma 5 we will show that $M(p) < 1 + \frac{4\delta}{3-2\delta}$ for every point p that is 3 above $\ell(t)$ such that $N'' \leq D(p) < N'$, and that $M(p) > 1 + \frac{2\delta}{3-2\delta}$ for every point p that is 3 below $\ell(r)$ such that $N'' \leq D(p) < N'$.

We will begin by first showing that $M(p) > 1 + \frac{2\delta}{3-2\delta} = \frac{3}{3-2\delta}$ for every point p that is 3 below $\ell(r)$ such that $N'' \leq D(p) < N'$. Let $p \in P$ be the set of all points such that p is 3 below $\ell(r)$ and $N'' \leq D(p) < N'$. Notice that the point $p \in P$ with minimum slope is such that $D(p) = N''$. See Figure 13. Recall that $M(r) = \frac{y}{x} = \frac{\lceil \frac{200}{\delta^2} \rceil + \lceil \frac{600-182\delta}{3\delta-2\delta^2} \rceil}{\lceil \frac{200}{\delta^2} \rceil} \geq \frac{3+\delta-0.91\delta^2}{3-2\delta+0.015\delta^2-0.01\delta^3}$. From Observation 7 we can obtain the x-coordinate of the intersection of $\ell(r)$ and our bound on N'' with $(\frac{63}{\delta^2}) \cdot (\frac{3-2\delta+0.015\delta^2-0.01\delta^3}{6-\delta-0.89\delta^2-0.01\delta^3}) \leq \frac{189-125.055\delta}{6\delta^2-1.905\delta^3}$ since δ is at most 0.1. Similarly, we can obtain the x-coordinate of the intersection of the line with slope $\frac{3}{3-2\delta}$ and our bound on N'' with $(\frac{63}{\delta^2}) \cdot (\frac{3-2\delta}{6-2\delta}) = \frac{189-126\delta}{6\delta^2-2\delta^3}$. If $\frac{189-126\delta}{6\delta^2-2\delta^3} - \frac{189-125.055\delta}{6\delta^2-1.905\delta^3} > 3$ is true, then $M(p) > \frac{3}{3-2\delta}$ for all $p \in P$. We have $\frac{189-126\delta}{6\delta^2-2\delta^3} - \frac{189-125.055\delta}{6\delta^2-1.905\delta^3} > 3$ which is true if $\frac{24.57-236.16\delta+140.58\delta^2+22.86\delta^3}{72\delta-46.86\delta^2-7.62\delta^3} > 0$. The denominator is always positive since δ is at most 0.1. It follows from $\delta \leq 0.1$ that the numerator is always positive. Indeed, $24.57-236.16(0.1)+140.58(0.1)^2+22.86(0.1)^3 = 2.38266$. Thus $M(p) > \frac{3}{3-2\delta}$ for all $p \in P$ which implies that for every $p \in \text{Subtree}(s)$ it must be that $M(p) > \frac{3}{3-2\delta}$.



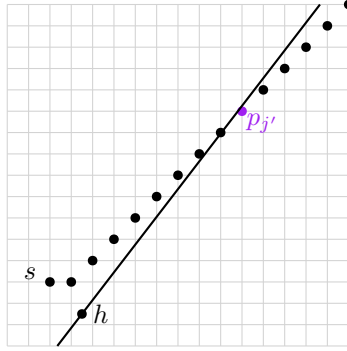
■ **Figure 14** An illustration of the first case.

Next we will show that $M(p) < 1 + \frac{4\delta}{3-2\delta} = \frac{3+2\delta}{3-2\delta}$ for every point p that is 3 above $\ell(t)$ such that $N'' \leq D(p) < N'$. Let $p \in P'$ be the set of all points such that p is 3 above $\ell(t)$ and $N'' \leq D(p) < N'$. Recall that $M(t) = \frac{y+21}{x-21} = \frac{\lceil \frac{200}{\delta^2} \rceil + \lceil \frac{600-182\delta}{3\delta-2\delta^2} \rceil + 21}{\lceil \frac{200}{\delta^2} \rceil - 21} \leq \frac{3+\delta}{3-2.5\delta}$. From Observation 7 we can obtain the x-coordinate of the intersection of $\ell(t)$ and our bound on N'' with $(\frac{63}{\delta^2}) \cdot (\frac{3-2.5\delta}{6-1.5\delta}) = \frac{189-157.5\delta}{6\delta^2-1.5\delta^3}$. Similarly, we can obtain the x-coordinate of the intersection of the line with slope $\frac{3+2\delta}{3-2\delta}$ and our bound on N'' with $(\frac{63}{\delta^2}) \cdot (\frac{3-2\delta}{6}) = \frac{189-126\delta}{6\delta^2}$. If $\frac{189-157.5\delta}{6\delta^2-1.5\delta^3} - \frac{189-126\delta}{6\delta^2} > 3$ is true then $M(p) < \frac{3+2\delta}{3-2\delta}$ for all $p \in P'$. We then have $\frac{189-157.5\delta}{6\delta^2-1.5\delta^3} - \frac{189-126\delta}{6\delta^2} > 3$ is true if $\frac{186-594\delta+54\delta^2}{72\delta-18\delta^2} > 0$. The denominator is always positive since δ is at most 0.1. The numerator is positive since $\delta \leq 0.1$. Indeed, when $\delta = 0.1$ we have $186 - 594(0.1) + 54(0.1)^2 = 127.14$. Thus $M(p) < \frac{3+2\delta}{3-2\delta}$ for all $p \in P'$ which implies that for every $p \in \text{Subtree}(s)$ it must be that $M(p) < \frac{3+2\delta}{3-2\delta}$. Therefore it must be that for every $p \in \text{Subtree}(s)$ that $1 + \frac{2\delta}{3-2\delta} < M(p) < 1 + \frac{4\delta}{3-2\delta}$. This concludes the proof of (1) from Lemma 5.

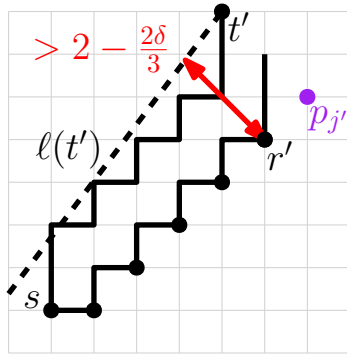
Proving s satisfies (2). We will first show that $N - D(s) > \frac{7+\delta}{\delta}$, and then use this bound to show that (2) must be true. Recall that $N = x + y = 2\lceil \frac{200}{\delta^2} \rceil + \lceil \frac{600-182\delta}{\delta(3-2\delta)} \rceil \geq \frac{1200-200\delta-182\delta^2}{3\delta^2-2\delta^3}$. Next, recall that $D(s) < N' \leq \frac{382}{\delta^2-\frac{2}{3}\delta^3}$. The difference between these two bounds is $\frac{1200-200\delta-182\delta^2}{3\delta^2-2\delta^3} - \frac{382}{\delta^2-\frac{2}{3}\delta^3} = \frac{54-200\delta-182\delta^2}{3\delta^2-2\delta^3}$. Clearly we can see that $\frac{54-200\delta-182\delta^2}{3\delta^2-2\delta^3} > \frac{7+\delta}{\delta}$ is true. It follows that $N - D(s) > \frac{7+\delta}{\delta}$. Let $d = D(s) + \frac{7+\delta}{\delta}$. Recall that s^{\nearrow} extends to at least one point in Σ and s^{\searrow} must also extend to at least one point in Σ . It then follows from $N - D(s) > \frac{7+\delta}{\delta}$ that s^{\nearrow} extends to at least one point on d and s^{\searrow} must also extend to at least one point on d . We will show that this implies that $\text{ConeWidth}(\text{Subtree}(s), d) > 2 - \frac{2\delta}{3}$.

There are two cases to consider. The first is there is a split point in the path from s^{\searrow} before d , and the second is there is no split point in the path from s^{\searrow} before d . Let us consider the first case. Let s' be a split point in the path from s^{\searrow} . We will show that $\text{ConeWidth}(\text{Subtree}(s), D(s') + 1) \geq 2$. By assumption $D(s') < d$ which implies that $D(s') + 1 \leq d$. Trivially, $s^{\searrow} \in R(s'^{\searrow})$ and $s^{\nearrow} \in R(s'^{\nearrow})$. Furthermore, since s^{\nearrow} must extend to d it then trivially extends to $D(s') + 1$. We also know from (S3) that any point that s^{\nearrow} extends to on $D(s') + 1$ must be above s'^{\nearrow} . Notice that the diagonal distance between s'^{\searrow} and s'^{\nearrow} is 2. It then follows that $\text{ConeWidth}(\text{Subtree}(s), D(s') + 1) \geq 2$ which implies $\text{ConeWidth}(\text{Subtree}(s), d) \geq 2$. See Figure 14.

In the second case we consider the point $h := (s_x + 1.5, s_y - 1.5)$. Recall that the error is less than 1.5. This implies that the bottom edge of $\text{Cone}(\text{Subtree}(s))$ must intersect $D(s)$ above h . See Figure 15. It follows that all points in $\text{Subtree}(s^{\searrow})$ have slope greater than



■ **Figure 15** An illustration of the points of the form p_j and the point h .



■ **Figure 16** An illustration of how two consecutive movements on the path from s^{\rightarrow} to d implies that $ConeWidth(Subtree(s), D(r')) > 2 - \frac{2\delta}{3}$.

$M(h)$ otherwise the error is at least 1.5. Let the points p_j be of the form $p_j := (s_x^{\rightarrow} + j, s_y^{\rightarrow} + j)$ for any $j \geq 1$. If s^{\rightarrow} extends to a p_j then it must be that the path from s^{\rightarrow} to diagonal $D(p_j)$ has j vertical and j horizontal movements. We would like to know the maximum j for which s^{\rightarrow} can extend to p_j such that $M(p_j) > M(h)$. Notice that $s^{\rightarrow} = (h_x - 0.5, h_y + 1.5)$ which implies $M(p_j) = \frac{h_y + 1.5 + j}{h_x - 0.5 + j}$. We will now solve for which j satisfies $\frac{h_y + 1.5 + j}{h_x - 0.5 + j} \leq M(h)$. Notice this is equivalent as asking for what j satisfies $\frac{h_y + 1.5 + j}{h_x - 0.5 + j} \leq \frac{h_y}{h_x}$. We then have $h_y h_x + h_x(1.5 + j) \leq h_y h_x + h_y(j - 0.5) \implies \frac{1.5 + j}{j - 0.5} \leq \frac{h_y}{h_x}$. Thus we would like to know for which j satisfies $\frac{1.5 + j}{j - 0.5} \leq M(h)$.

Given a fixed $M(s)$, we can see that $M(h)$ grows as $D(s)$ becomes larger. Recall that $M(s) > \frac{3}{3-2\delta}$ and $D(s) \geq N''$. It follows that $M(h)$ is at least the slope of the point, q , that is 1.5 diagonal distance below $Inter(\ell(3 - 2\delta, 3), N'')$. From Observation 7 we then have $q_x = \frac{3-2\delta}{6-2\delta} N'' + 1.5 \leq \frac{189-117\delta}{6\delta^2-2\delta^3}$ and $q_y = N'' - \frac{189-117\delta}{6\delta^2-2\delta^3} = \frac{189-9\delta}{6\delta^2-2\delta^3}$. We can then conclude that $M(h) \geq \frac{189-9\delta}{189-117\delta}$.

It follows that if we find for which j is $\frac{1.5+j}{j-0.5} \leq \frac{189-9\delta}{189-117\delta}$ true, it will also be true for $\frac{1.5+j}{j-0.5} \leq M(h)$. We then have $\frac{1.5+j}{j-0.5} \leq \frac{189-9\delta}{189-117\delta}$ is true for all $j \geq \frac{378-180\delta}{108\delta}$. Therefore the maximum j for which s^{\rightarrow} can extend to p_j is $j = \lceil \frac{378-180\delta}{108\delta} \rceil - 1$. Let $j' := \lceil \frac{378-180\delta}{108\delta} \rceil$, that is the maximum j plus 1. Let us then consider what point s^{\rightarrow} must extend to on $D(p_{j'})$. Because $M(p_{j'}) < M(h)$ it must be that s^{\rightarrow} extends to a point who's slope is at minimum $M(p_{j'})$. Recall that $p_{j'}$ is j' vertical and j' horizontal movements away from s^{\rightarrow} . It then follows that $p_{j'}^{\leftarrow}$ is $j' + 1$ vertical and $j' - 1$ horizontal movements away from s^{\rightarrow} . Therefore

26:20 Optimal Bounds for Weak Consistent Digital Rays in 2D

the path from s^\rightarrow to $D(p_{j'})$ must have at least two more vertical than horizontal movements. It then follows from pigeonhole principle that the path from s^\rightarrow to $D(p_{j'})$ must contain at least two consecutive vertical movements. See Figure 16.

Notice that $D(p_{j'}) = D(s) + 2j' + 1 = D(s) + 2 \cdot \lceil \frac{378 - 180\delta}{108\delta} \rceil + 1 < D(s) + 2 \cdot (\frac{378 - 180\delta}{108\delta} + 1) + 1 \leq D(s) + \frac{7}{\delta}$. Recall that $d = D(s) + \frac{7 + \delta}{\delta}$. Clearly $D(p_{j'}) < d$. Therefore the path from s^\rightarrow to d must contain at least two consecutive vertical movements. We will now show these consecutive vertical movements imply $ConeWidth(Subtree(s), d) > 2 - \frac{2\delta}{3}$. Let r' be a point such that $D(r') \leq d - 2$ and s^\rightarrow extends to r' , r'^\uparrow , and $r'^{\uparrow\uparrow}$. Let $t' := r'^{\uparrow\uparrow\swarrow}$. Recall that s^\uparrow must extend to N which implies s^\uparrow also extends to $D(t')$. We also know from (S3) that any point that s^\uparrow extends to on $D(t')$ must be above $r'^{\uparrow\uparrow}$. This implies that $ConeWidth(Subtree(s), D(r'))$ is at least the diagonal distance between r' and $Inter(\ell(t'), D(r'))$. As we showed in our proof of (1) from Lemma 5 all points s extends to must have slope less than $\frac{3+2\delta}{3-2\delta}$ which implies that $M(t') < \frac{3+2\delta}{3-2\delta}$, otherwise there is no point s^\uparrow can extend to on $D(t')$. Let z be the point that is $2 - \frac{2\delta}{3}$ above r' . Let \bar{t}' be the Euclidean line that intersects both t' and z . If $Inter(\ell(t'), D(r'))$ is above $Inter(\bar{t}', D(r'))$ then $ConeWidth(Subtree(s), D(r')) > 2 - \frac{2\delta}{3}$. We then have $M(\bar{t}') = \frac{t'_y - z_y}{t'_x - z_x} = \frac{t'_y - (t'_y - (1 + \frac{2\delta}{3}))}{t'_x - (t'_x - (1 - \frac{2\delta}{3}))} = \frac{3+2\delta}{3-2\delta}$. It then follows that $M(t') < M(\bar{t}')$ which implies that $Inter(\ell(t'), D(r'))$ is above $Inter(\bar{t}', D(r'))$. Thus $ConeWidth(Subtree(s), D(r')) > 2 - \frac{2\delta}{3}$ which implies that $ConeWidth(Subtree(s), d) > 2 - \frac{2\delta}{3}$. This concludes the proof of (2) from Lemma 5, which then concludes the entire proof of Lemma 5.

Matroid-Constrained Maximum Vertex Cover: Approximate Kernels and Streaming Algorithms

Chien-Chung Huang ✉

CNRS, DI ENS, École normale supérieure, Université PSL, Paris, France

François Sellier ✉

Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

MINES ParisTech, Université PSL, F-75006, Paris, France

Abstract

Given a graph with weights on the edges and a matroid imposed on the vertices, our problem is to choose a subset of vertices that is independent in the matroid, with the objective of maximizing the total weight of covered edges. This problem is a generalization of the much studied MAX k -VERTEX COVER problem, where the matroid is the simple uniform matroid, and it is also a special case of maximizing a monotone submodular function under a matroid constraint.

In this work, we give a Fixed Parameter Tractable Approximation Scheme (FPT-AS) when the given matroid is a partition matroid, a laminar matroid, or a transversal matroid. Precisely, if k is the rank of the matroid, we obtain $(1 - \varepsilon)$ approximation using $(\frac{1}{\varepsilon})^{O(k)} n^{O(1)}$ time for partition and laminar matroids and using $(\frac{1}{\varepsilon} + k)^{O(k)} n^{O(1)}$ time for transversal matroids. This extends a result of Manurangsi for uniform matroids [26]. We also show that these ideas can be applied in the context of (single-pass) streaming algorithms.

Our FPT-AS introduces a new technique based on matroid union, which may be of independent interest in extremal combinatorics.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms

Keywords and phrases Maximum vertex cover, matroid, approximate kernel, streaming

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.27

Funding This work was funded by the grants ANR-19-CE48-0016 and ANR-18-CE40-0025-01 from the French National Research Agency (ANR).

Acknowledgements The authors thank the anonymous reviewers for their helpful comments. One of them especially pointed out a mistake on a counter-example for gammoids in the submitted version.

1 Introduction

Let $G = (V, E)$ be a graph. A weight $w(e)$ is associated with each edge $e \in E$. By convention we set $n = |V|$ and $m = |E|$. For a vertex $v \in V$ we define $\delta(v)$ the set of edges that are incident to v . The *degree* of a vertex $v \in V$, denoted $\deg(v)$, is the size of $\delta(v)$, and we define the *weighted degree* of a vertex $v \in V$ as the sum $\deg_w(v) = \sum_{e \in \delta(v)} w(e)$. For two sets of vertices $S, T \subseteq V$ in a graph G , we denote $E_G(S, T) = \sum_{e \in E, e \cap S \neq \emptyset, e \cap T \neq \emptyset} w(e)$ the sum of the weights of the edges that have one endpoint in S and one endpoint in T . Then $E_G(S, S)$, abbreviated $E_G(S)$, denotes the sum of the weights of the edges that are covered by S (*i.e.* having at least one of its endpoints in S).

Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid on the ground set V . Recall that $\mathcal{M} = (V, \mathcal{I})$ is a matroid if the following three conditions hold: (1) $\emptyset \in \mathcal{I}$, (2) if $X \subseteq Y \in \mathcal{I}$, then $X \in \mathcal{I}$, and (3) if $X, Y \in \mathcal{I}, |Y| > |X|$, there exists an element $e \in Y \setminus X$ so that $X \cup \{e\} \in \mathcal{I}$. The sets in \mathcal{I}



© Chien-Chung Huang and François Sellier;
licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 27; pp. 27:1–27:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are the *independent sets* and the *rank* k of the matroid \mathcal{M} is defined as $\max_{X \in \mathcal{I}} |X|$. For more details about matroids, we refer the reader to [33]. In this paper, given a set $S \subseteq V$ and $v \in V$, we will denote $S \cup \{v\}$ by $S + v$ and $S \setminus \{v\}$ by $S - v$ for conciseness.

The problem that we consider in this paper is to choose an independent set of vertices $S \in \mathcal{I}$, with the objective of maximizing $E_G(S)$, namely, the total weight of the edges covered by S .

Let us put our problem in a larger picture. When the given matroid \mathcal{M} is a uniform matroid (see below for a formal definition), our problem reduces to the MAX k -VERTEX-COVER problem, where we want to choose k arbitrary vertices so as to maximize the total weight of covered edges. This is a classical problem with a long history: the greedy heuristics is known to give $1 - 1/e$ approximation as shown by Hochbaum and Pathria [23]. Ageev and Sviridenko [1] propose an LP-based approach and the technique of pipage rounding to obtain $3/4$ approximation. Using SDP, Feige and Langberg [14] improve this ratio to $3/4 + \delta$ for some small constant $\delta > 0$. The current best approximation ratio is 0.92, achieved by Manurangsi [26]. For some special cases of the problem, different ratios are also obtained, e.g. see [4, 21, 22]. On the hardness side, to our knowledge, the best inapproximability ratio is due to Austrin and Stankovic [2], which is 0.929.

The MAX k -VERTEX-COVER has also been studied through the lens of fixed-parameterized-tracability. Guo et al. [19] show the problem to be $W[1]$ -hard with k as parameter, thus showing the unlikelihood of getting an exact solution in FPT time. Nonetheless, Marx [28] shows that it is possible to get a near-optimal solution in FPT time. Precisely, he gives an FPT approximation scheme (FPT-AS), that delivers a $(1 - \varepsilon)$ -approximate solution in $(k/\varepsilon)^{O(k^3/\varepsilon)} n^{O(1)}$ time. This running time is later improved by Gupta et al. [20] and Manurangsi [26].

Here we recall the definition of an FPT-AS [28]:

► **Definition 1.** *Given a parameter function κ associating a natural number to each instance $x \in I$ of a given problem, a Fixed-Parameter Tractable Approximation Scheme (FPT-AS) is an algorithm that can provide a $(1 - \varepsilon)$ approximation in $f(\varepsilon, \kappa(x)) \cdot |x|^{O(1)}$ time.*

In our case, the instances are made of a graph and a matroid, and the parameter of an instance is the rank k of its matroid.

Regarding the more general case of an arbitrary matroid of rank k , one can obtain $3/4$ approximation in polynomial time by combining known techniques.¹ This is also a special case of maximizing a submodular function (more precisely, a coverage function) under matroid constraint, for which a $1 - 1/e$ approximation can be achieved in polynomial time [6, 18]. In this work, we try to do better than this ratio for some special cases of matroids, in the context of fixed-parameter algorithms. We also show that the ideas developed here can be applied in the streaming setting [31]. In streaming, maximizing of a submodular function under a general matroid constraint has received much attention recently [8, 10, 16].

¹ Ageev and Sviridenko [1] show that, for the case of a uniform matroid, the optimal fractional solution x^* of the LP has at least $3/4$ of the optimal value. They then use the pipage rounding to transform it into an integral solution with value no less than x^* . The same LP approach can be generalized for arbitrary matroids. The optimal fractional solution can be obtained by Ellipsoid algorithm: even though the linear program to describe the independent sets of an arbitrary matroid may use exponentially many constraints, we can design a separation oracle using an algorithm of Cunningham [12]. What remains is just the pipage rounding with a general matroid – this is already known to be do-able by Calinescu et al. [7]. We thank Pasin Manurangsi for communicating to us this method.

1.1 Our Contribution

Let us recall some definitions. A *uniform matroid* of rank k is a matroid where the independent sets are the sets S of cardinality at most k . A *partition matroid* is a matroid where we are given a partition V_1, \dots, V_r of the ground set V and bounds k_1, \dots, k_r such that a set S is independent if for all $1 \leq i \leq r$, $|S \cap V_i| \leq k_i$. A *laminar matroid* is given as a laminar family V_1, \dots, V_r of V , i.e. given $V_i \neq V_j$, then either $V_i \cap V_j = \emptyset$, or $V_i \subset V_j$, or $V_j \subset V_i$, along with bounds k_1, \dots, k_r . A set $S \subseteq V$ is independent if for all $1 \leq i \leq r$, $|S \cap V_i| \leq k_i$. Finally, a *transversal matroid* is given in a family $V_1, \dots, V_k \subseteq V$, where V_i s are not necessarily disjoint, and a set $S = \{u_1, \dots, u_t\}$ is independent if and only if for each element u_i , there exists a distinct $\phi(i)$ so that $u_i \in V_{\phi(i)}$. These simple types of matroid have been extensively studied in a large variety of contexts.

A uniform matroid is a special case of a partition matroid, which is again a special case of a laminar or a transversal matroid. However, laminar matroids and transversal matroids are not inclusive of each other [32]. Transversal matroids were introduced in the 60s, by Edmonds and Fulkerson [13] and by Mirsky and Perfect [30]. They unified many results in transversal theory and are generally considered an important class of matroids, e.g. see [35]. Laminar matroids receive much attention recently in the community of theoretical computer science, especially in the context of matroid secretary problem, e.g., see [3, 9, 15, 24, 34]. Our results involve these kinds of matroids.

► **Theorem 2.** *For every $\varepsilon > 0$, we can extract an approximate kernel $V' \subseteq V$ in polynomial time so that a $(1 - \varepsilon)$ -approximate solution is contained in V' . The size of the kernel V' depends on the type of the given matroid \mathcal{M} .*

- (i) $|V'| \leq \frac{k}{\varepsilon}$ when \mathcal{M} is a partition matroid;
- (ii) $|V'| \leq \frac{2k}{\varepsilon}$ when \mathcal{M} is a laminar matroid;
- (iii) $|V'| \leq \frac{k}{\varepsilon} + k(k - 1)$ when \mathcal{M} is a transversal matroid.

Furthermore, by a brute force enumeration, we can find the desired $1 - \varepsilon$ approximation in $(\frac{1}{\varepsilon})^{O(k)} n^{O(1)}$ time for partition and laminar matroids and $(\frac{1}{\varepsilon} + k)^{O(k)} n^{O(1)}$ time for transversal matroids.

In addition, by a straightforward modification of our proofs in Section 2 (see Appendix A), we can show the following corollary.

► **Corollary 3.** *Suppose that we are given a hypergraph $G = (V, E)$ with edge size bounded by a constant $\eta \geq 2$. We can compute a $(1 - (\eta - 1) \cdot \varepsilon)$ approximation using $(\frac{1}{\varepsilon})^{O(k)} n^{O(1)}$ time for partition and laminar matroids and $(\frac{1}{\varepsilon} + k)^{O(k)} n^{O(1)}$ time for transversal matroids.*

Put slightly differently, when G is a hypergraph with edge size at most η , we can obtain $1 - \varepsilon$ approximation in $(\frac{\eta}{\varepsilon})^{O(k)} n^{O(1)}$ or $(\frac{\eta}{\varepsilon} + k)^{O(k)} n^{O(1)}$ time, depending on the type of matroid. To see the interest of this corollary, we recall that recently Manurangsi [27] showed that if η is unbounded, one cannot obtain an approximation ratio better than $1 - 1/e + \varepsilon$, assuming GAP-ETH, in FPT time (where the matroid rank k is the parameter). This result holds even for the simplest uniform matroid. Thus Corollary 3 implies that one can circumvent this lower bound by introducing another parameter η , even for more general matroids.

Our algorithm is inspired by that of Manurangsi [26] for the case of uniform matroid. So let us briefly summarize his approach: an *approximate kernel*² V' is first extracted from V , where V' is simply made of the k/ε vertices with the largest weighted degrees. Let O be an optimal solution. Apparently, a vertex of O is either part of the kernel V' , or its weighted degree is dominated by all vertices in $V' \setminus O$. To recover the optimal value, we can potentially use the vertices in $V' \setminus O$ to replace the vertices in $O \setminus V'$. However, there is a risk in doing this: an edge among the vertices in $V' \setminus O$ can be double-counted, if both of its endpoints are chosen to replace the vertices in $O \setminus V'$. To circumvent this issue, Manurangsi uses a random sampling argument to show that in expectation such double counting is negligible. Therefore, by the averaging principle, there exists a $(1 - \varepsilon)$ -approximate solution in the kernel V' , which can be found using brute force.

To generalize the approach of Manurangsi for more general matroids, one has to answer the quintessential question: how does one guarantee that the sampled vertices, along with $O \cap V'$, are independent in \mathcal{M} ? To overcome this difficulty, we introduce a new technique. We take the union of some number τ of matroids \mathcal{M} . Such a union is still a matroid, which we denote as $\tau\mathcal{M}$. We then apply a greedy algorithm on $\tau\mathcal{M}$ (based on non-increasing weighted degrees) to construct an independent set V' in $\tau\mathcal{M}$. We show that such a set V' is “robust” (see Definition 4) in the sense that we can sample vertices from V' so that they, along with $O \cap V'$, are *always* independent and in expectation cover edges of weight at least $1 - \varepsilon$ times that of O .

We note that the value of τ automatically gives an upper bound on the kernel size V' , which is τk . Theorem 6 shows the required scale of τ , depending on the type of the given matroid. We leave as an open question whether for matroids more general than considered in the paper, a larger τ can always yield the kernel.

In the last part of this work, we consider the problem in the semi-streaming model [31]. In that context, the edges in E arrive over time but we have only limited space (for instance, $O(n \cdot \text{polylog}(n)) = o(m)$) and cannot afford to store all edges in E . In this context we can also obtain a $(1 - \varepsilon)$ approximation using $O(\frac{nk}{\varepsilon})$ space in a single pass.³ The idea of using (parameterized) kernels for streaming algorithms has recently been introduced, for instance in [11, 29]. We also show that a FPT-streaming algorithm can be derived from our ideas to get a $(1 - \varepsilon)$ approximation for a special form of maximization of a coverage function with bounded frequency (see Theorem 14, Remark 15 and Appendix B for details).

2 Kernelization Framework

In this section, we give a general framework to construct the kernel by a greedy procedure and show how such a kernel contains a $(1 - \varepsilon)$ -approximate solution.

► **Definition 4.** Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid with weights $\omega : V \rightarrow \mathbb{R}^+$. We say $V' \subseteq V$ is t -robust if given any base $O \in \mathcal{I}$, there is a bijection from the elements $u_1, \dots, u_t \in O \setminus V'$ to subsets $U_{u_1}, \dots, U_{u_t} \subseteq V' \setminus O$ so that

- (i) the U_{u_i} s are mutually disjoint and $|U_{u_i}| = t$,
- (ii) all elements in U_{u_i} have weights no less than u_i ,
- (iii) by taking an arbitrary element $u'_i \in U_{u_i}$ for all i , $(V' \cap O) \cup \{u'_i\}_{i=1}^t$ is a base in \mathcal{M} .

² In the rest of the paper, we will just say kernel, dropping the adjective. The interest for this kind of kernel has risen recently in the community [17, 25].

³ Here we assume that the matroid is given in the form of oracle, in which the algorithm has access freely – this is a standard assumption in the streaming setting when matroids are involved.

We next recall the definition of matroid union.

► **Definition 5.** *Suppose that $\mathcal{M} = (V, \mathcal{I})$ is a matroid. Then we can define $\tau\mathcal{M} = (V, \mathcal{I}_\tau)$ as the union of τ matroids \mathcal{M} , as follows: $S \in \mathcal{I}_\tau$ if S can be partitioned into $S_1 \cup \dots \cup S_\tau$ so that each $S_i \in \mathcal{I}$.*

Recall that the union of matroids is still a matroid and here the rank of $\tau\mathcal{M}$ is at most τ times the rank of \mathcal{M} . e.g. see [33, Chapter 42]. We can now state our main theorem.

► **Theorem 6.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid with weights $\omega : V \rightarrow \mathbb{R}^+$ and rank k . Consider the following greedy procedure on $\tau\mathcal{M} = (V, \mathcal{I}_\tau)$ to construct V' : initially $V' = \emptyset$. Process the elements in V by non-increasing weights ω . For each element u , if $V' + u \in \mathcal{I}_\tau$, add u into V' , otherwise, ignore it. The final V' is t -robust*

- (i) if \mathcal{M} is a partition matroid and $\tau \geq t$,
- (ii) if \mathcal{M} is a laminar matroid and $\tau \geq 2t$,
- (iii) if \mathcal{M} is a transversal matroid and $\tau \geq t + k - 1$.

Notice that the rank of the matroid $\tau\mathcal{M}$ gives an upper-bound on the size of V' . The next section will give the proof of this theorem for each type of matroid considered. In the following we show how it can be used to construct the $1 - \varepsilon$ approximation.

Let the weight $\omega : V \rightarrow \mathbb{R}^+$ be the weighted degrees in the graph $G = (V, E)$, that is, $\omega(u) = \deg_w(u)$. Apply Theorem 6 by setting $t = \frac{1}{\varepsilon}$. Then V' is $\frac{1}{\varepsilon}$ -robust. Note that we suppose that $\frac{1}{\varepsilon}$ is an integer, otherwise we could take $t = \lceil \frac{1}{\varepsilon} \rceil$.

Based on V' , we create a new graph $G' = (V', E')$, where an original edge $e = \{u, v\}$ is retained in E' if both of its endpoints are in V' . In case only one endpoint, say u is in V' , we add a self-loop to u in E' to represent this edge.

► **Lemma 7.** *Suppose that V' is the constructed set that is $\frac{1}{\varepsilon}$ -robust. Then V' contains a set S such that $S \in \mathcal{I}$ and $E_G(S) \geq (1 - \varepsilon)E_G(O)$ where O denotes an optimal solution of the problem.*

Proof. Let $O \in \mathcal{I}$ be an optimal solution. We denote $O^{in} = O \cap V'$, $O^{out} = O \setminus O^{in}$. Then by $\frac{1}{\varepsilon}$ -robustness, we have mutually disjoint sets $U_v \subseteq V' \setminus O$ for each $v \in O^{out}$, each of size $\frac{1}{\varepsilon}$. We set $U = \cup_{v \in O^{out}} U_v$. We construct a set $S \subseteq V'$ as follows: S is initialized as O^{in} . Then from each set U_v , for all $v \in O^{out}$, pick an element at random and add it into S . By definition of $\frac{1}{\varepsilon}$ -robustness, S is independent in \mathcal{M} .

Next we will show that

$$\mathbb{E}[E_G(S)] \geq (1 - \varepsilon) \cdot \mathbb{E}[E_G(O)].$$

Let $U^* = S \setminus O^{in}$, i.e. those elements that are added into S randomly. First, we have that:

$$E_G(S) = E_G(O^{in}) + E_G(U^*) - E_G(O^{in}, U^*).$$

We bound $\mathbb{E}[E_G(O^{in}, U^*)]$ as follows. By construction, $\mathbb{P}[u \in U^*] = \varepsilon$ for all $u \in U$. Then,

$$\mathbb{E}[E_G(O^{in}, U^*)] = \sum_{u \in U} \sum_{v \in O^{in}} w(\{u, v\}) \cdot \mathbb{P}[u \in U^*] = \varepsilon \sum_{u \in U} \sum_{v \in O^{in}} w(\{u, v\}) \leq \varepsilon \cdot E_G(O^{in}).$$

Furthermore, the value $\mathbb{E}[E_G(U^*)]$ can be rearranged as follows:

$$\mathbb{E}[E_G(U^*)] = \mathbb{E} \left[\sum_{u \in U^*} \left(\deg_w(u) - \frac{1}{2} \sum_{v \in U^* \setminus \{u\}} w(\{u, v\}) \right) \right]$$

$$\begin{aligned}
 &= \sum_{u \in U} \left(\deg_w(u) \cdot \mathbb{P}[u \in U] - \frac{1}{2} \sum_{v \in U \setminus \{u\}} w(\{u, v\}) \cdot \mathbb{P}[u \in U^* \wedge v \in U^*] \right) \\
 &\geq \sum_{u \in U} \left(\deg_w(u) \cdot \varepsilon - \frac{1}{2} \sum_{v \in U \setminus \{u\}} w(\{u, v\}) \cdot \varepsilon^2 \right) \geq \varepsilon(1 - \varepsilon/2) \left(\sum_{u \in U} \deg_w(u) \right),
 \end{aligned}$$

where the first inequality comes from the fact that $\mathbb{P}[u \in U^* \wedge v \in U^*] \leq \mathbb{P}[u \in U^*] \cdot \mathbb{P}[v \in U^*]$.

Recall that by robustness, for all $u \in O^{out}$, the elements of U_u have weighted degrees no less than that of u . Therefore,

$$\begin{aligned}
 \mathbb{E}[E_G(U^*)] &\geq \varepsilon(1 - \varepsilon/2) \left(\sum_{u \in O^{out}} \sum_{v \in U_u} \deg_w(v) \right) \geq \varepsilon(1 - \varepsilon/2) \left(\sum_{u \in O^{out}} \frac{1}{\varepsilon} \cdot \deg_w(u) \right) \\
 &\geq (1 - \varepsilon/2) \cdot E_G(O^{out}).
 \end{aligned}$$

As a result, we get:

$$\mathbb{E}[E_G(S)] \geq E_G(O^{in}) + (1 - \varepsilon/2) \cdot E_G(O^{out}) - \varepsilon \cdot E_G(O^{in}) \geq (1 - \varepsilon) \cdot E_G(O).$$

By averaging principle, there exists $S \subseteq V'$ such that $S \in \mathcal{I}$ and $E_G(S) \geq (1 - \varepsilon) \cdot E_G(O)$. ◀

3 Proof of Theorem 6

3.1 Partition Matroids

Consider a partition matroid $\mathcal{M} = (V, \mathcal{I})$ defined by a partition V_1, \dots, V_r of V and bounds k_1, \dots, k_r . Given $t \in \mathbb{N}$, we take in each set V_i of the partition the $\min\{|V_i|, t \cdot k_i\}$ elements having the largest weighted degrees. We denote these extracted sets as V'_i and their union as V' . Clearly, this is the same as the greedy algorithm stated in Theorem 6 applied on $t\mathcal{M}$.

To see that V' is t -robust, let $O \in \mathcal{I}$ be a base. We denote $O^{in} = O \cap V'$, $O^{out} = O \setminus O^{in}$, $O_i = O \cap V_i$, $O_i^{in} = O \cap V'_i$, $O_i^{out} = O_i \setminus O_i^{in}$, and we set $\bar{U}_i \subseteq V'_i \setminus O_i^{in}$ as an arbitrary subset of cardinality $t \cdot |O_i^{out}|$, for $1 \leq i \leq r$ (it is possible as $O_i \neq O_i^{in}$ implies that $|V'_i| = t \cdot k_i$). We then partition \bar{U}_i into $U_i^1, \dots, U_i^{|O_i^{out}|}$, each one of size t . It is easy to verify that the generated sets $\{U_i^j\}_{i,j}$ satisfy the three conditions stated in Definition 4.

3.2 Laminar Matroids

Recall that a laminar matroid $\mathcal{M} = (V, \mathcal{I})$ is given as a laminar family V_1, \dots, V_r along with bounds k_1, \dots, k_r . Without loss of generality, we can assume that $V = V_0$ with bound $k_0 = k$ (being the rank of \mathcal{M}) is a member in the family. Furthermore, we can also assume that each vertex $v \in V$ by itself is also a member $V_i = \{v\}$ with bound $k_i = 1$ in this family.

Such a laminar matroid \mathcal{M} can be naturally associated with a laminar tree T where the each tree node $T_i = (V_i, k_i)$ corresponds to V_i , and the structure of the tree reflects the inclusion relationship of the members V_i in the laminar family. In such a tree $T_0 = (V, k)$ corresponds to the root of the tree.

For ease of our study, we will assume that such a tree T is binary (so in total T contains $2n - 1$ nodes, with $n = |V|$). Such an assumption can be easily justified by adding more sets V_i into the laminar family with the appropriately defined bounds k_i .

In the following, the elements of $\{T_i = (V_i, k_i)\}_{0 \leq i \leq 2n-2}$ will be referred as “nodes”, whereas the elements of V will be referred as “vertices”. We are given $t \in \mathbb{N}$. To choose the vertices that are to be added into the kernel, we employ the following greedy procedure. We

process the vertices in non-increasing order with respect to their weighted degrees. At the beginning, V' is empty. When we consider a new vertex v , if for all i such that $v \in V_i$, we have $|V' \cap V_i| < 2t \cdot k_i$, then v is added to V' , otherwise v is simply ignored. This procedure is equivalent to the greedy algorithm described in Theorem 6 applied on $2t\mathcal{M}$.

A node T_j of the tree $\{T_i = (V_i, k_i)\}_{0 \leq i \leq 2n-2}$ is called *saturated* if $|V' \cap V_j| = 2t \cdot k_j$. Let O be an arbitrary solution. As in the previous subsection we will use the notations $V'_i = V_i \cap V'$, $O^{in} = V' \cap O$, and $O^{out} = O \setminus V'$. In the following, we say that a vertex or a set of vertices is “contained” in a tree node T_i if they are part of V'_i (equivalently, the leaves corresponding to these elements of V' are in the subtree of root T_i).

For every element $v \in V$, there exists a leaf T_{i_v} in the laminar tree such that $V_{i_v} = \{v\}$, and we have a unique path from the root T_0 to T_{i_v} . If a vertex $v \in O$ is not in V' , it means that some node along the path from T_0 to T_{i_v} was already saturated when v was processed: the *blocking node* of v is the deepest saturated node along this path. For each node T_i , we denote by B_i the set of vertices of O that are blocked by the node T_i , and we set $b_i = |B_i|$. Then, $b_i = 0$ when T_i is not saturated. Moreover, the B_i s are mutually disjoint and $\bigcup B_i = O^{out}$.

Then for each vertex $v \in O^{out}$, we construct a set \bar{U}_v of at least t vertices drawn from V' . Then an arbitrary subset $U_v \subseteq \bar{U}_v$ of t vertices is retained. We will argue that the generated sets $U_{v,s}$ ensure the robustness.

Constructing the sets \bar{U}_v s for all $v \in O^{out}$

We want the constructed sets \bar{U}_v s to satisfy the following three properties.

- (i) The sets \bar{U}_v s are mutually disjoint and are drawn from $V' \setminus O^{in}$.
- (ii) For each $v \in O^{out}$ and each $u \in \bar{U}_v$, $\deg_w(u) \geq \deg_w(v)$.
- (iii) Choosing an arbitrary $\bar{v} \in \bar{U}_v$ for each $v \in O^{out}$, the set $S = O^{in} \cup \{\bar{v}\}_{v \in O^{out}}$ is independent in the laminar matroid \mathcal{M} .

The formal algorithm for constructing the sets $\{\bar{U}_v\}_{v \in O^{out}}$ is given in Algorithm 1. Here we give the intuition behind it.

To guarantee Property (i), we first mark all elements in O^{in} as unusable. Then, each \bar{U}_v is chosen among the usable vertices. Once a set \bar{U}_v is allocated, all its vertices will be marked as unusable.

To guarantee Property (ii), first recall that each vertex $v \in O^{out}$ has a corresponding blocking node T_i (and $v \in B_i$). By our greedy procedure to build the kernel V' , we know that there exist $2t \cdot k_i$ vertices u in the set V'_i , all of whom contained in T_i and $\deg_w(u) \geq \deg_w(v)$. What we do is to choose a deepest blocking node T_i and to process one of its vertex $v \in B_i$ (Lines 6-7). As we will show later (Claim 10), such a blocking node must contain at least $2t$ usable vertices. We climb down the tree from the blocking node T_i until we reach a node T_j neither of whose child nodes contains more than t usable vertices (Lines 9-10). Recall that our tree is binary, as a result, the number of usable vertices contained in T_j is between t and $2t - 2$. All these usable vertices constitute a new set \bar{U}_v and then are marked as unusable.

How to guarantee Property (iii) is the most tricky part of our algorithm. Recall that we will choose an arbitrary vertex from \bar{U}_v to construct a solution S stated in (iii). Apparently we have no control over the choice of the arbitrary vertex from \bar{U}_v , nonetheless, we need to ensure that S does not violate any of the rank constraints k_i . What we do is to associate a variable s_i with each tree node T_i . This variable indicates how many vertices contained in T_i will *certainly* be part of S , according to O^{in} and the sets \bar{U}_v s that have been constructed so far. Once s_i is set to k_i , it is a warning that we should not use any more remaining usable vertices contained in T_i to construct the future sets \bar{U}_v .

Initially, $s_i = |V'_i \cap O^{in}|$. Each time that we have decided on a tree node T_j to form a new set U_v (Lines 9-12), we increase the value of s_j from T_j all the way up to the root (Lines 13-14). If any node T_j has its variable $s_j = k_j$, we say such a node is full-booked and we mark all its (remaining) usable vertices as unusable (Lines 15-16).

■ **Algorithm 1** Algorithm constructing the sets \overline{U}_v .

```

1:  $\forall v \in O \setminus V', \overline{U}_v \leftarrow \emptyset$ 
2:  $\forall 1 \leq i \leq 2n - 2, s_i \leftarrow |V'_i \cap O^{in}|$ 
3: the elements of  $V' \setminus O^{in}$  are marked as usable
4: the elements of  $O^{in}$  are marked as unusable
5: while there exists a set  $B_i$  which is not empty do
6:   let  $T_i$  be one of the deepest nodes such that  $B_i \neq \emptyset$ 
7:   let  $v \in B_i$  be an arbitrary vertex
8:    $B_i \leftarrow B_i - v$ 
9:   while  $T_i$  has a child  $T_j$  containing at least  $t$  usable vertices in  $V'_j$  do
10:     $i \leftarrow j$ 
11:   set  $\overline{U}_v$  as the set of usable elements in  $V'_i$ 
12:   mark all the elements of  $\overline{U}_v$  as unusable
13:   for all nodes  $T_j$  on the path from  $T_i$  to the root of the tree do
14:      $s_j \leftarrow s_j + 1$ 
15:     if  $s_j = k_j$  then ▷ in that case, we say that  $T_j$  is fully-booked
16:       mark all the elements of  $V'_j$  as unusable

```

We want to show that Algorithm 1 manages to build the sets \overline{U}_v s of size at least t satisfying the aforementioned properties.

▷ **Claim 8.** At any time during the execution of Algorithm 1, for a saturated node T_i that is not a descendant of a fully-booked node (no node above it is fully-booked), the number of usable vertices in V'_i is at least $2t \cdot (k_i - s_i)$.

Proof. As T_i is saturated, V'_i contains exactly $2t \cdot k_i$ vertices. The unusable vertices in V'_i fall into three categories:

- (i) the vertices in $V'_i \cap O^{in}$ (see Line 4), but that are not contained in any fully booked descendent node of T_i ,
- (ii) the vertices made unusable during the allocation of a set \overline{U}_v (see Line 12), but that are not contained in any fully booked descendent node of T_i ,
- (iii) the vertices that are contained in a fully-booked descendent node of T_i (see Line 16).

The vertices v_1, \dots, v_{l_1} in the first category each has a contribution of $+1$ in the value of s_i . Then, we can observe that an allocated set \overline{U}_v is either entirely contained in a fully-booked node or has no element at all in any fully-booked node. Let us denote $\overline{U}_{v_1}, \dots, \overline{U}_{v_{l_2}}$ the allocated sets contained in T_i that are not contained in any fully-booked node. In addition, by construction, each set \overline{U}_{v_j} contains at most $2t - 2$ vertices (otherwise we would be able to go deeper in the binary tree for the allocation, see Lines 9-10, because at least one child would contain at least t usable elements). Each set \overline{U}_{v_j} has a contribution of $+1$ in the value of s_i . Finally, among the fully-booked nodes in the subtree of root T_i , we consider the nodes $T_{i_1}, \dots, T_{i_{l_3}}$ that are inclusion-wise maximal (*i.e.* the roots of the fully-booked parts of the subtree). A fully-booked subtree of root T_j has to bring a $+k_j$ contribution to s_i (otherwise it would not be fully-booked, and observe that a set \overline{U}_v is included in at most one such fully-booked maximal node), and is making at most $2t \cdot k_j$ vertices of V'_i

unusable (the worst case being that T_j was also a saturated node). We also have the equality $s_i = l_1 + l_2 + \sum_{j=1}^{l_3} k_{i_j}$. As a result, we have at most $l_1 + (2t - 2) \cdot l_2 + \sum_{j=1}^{l_3} 2t \cdot k_{i_j} \leq 2t \cdot s_i$ unusable vertices in V'_i . \triangleleft

▷ **Claim 9.** At any time during the execution of the algorithm, for all $i \in \llbracket 1, 2n - 2 \rrbracket$, we have $k_i \geq s_i + b_i + \sum_{T_j \text{ below } T_i} b_j$ as an invariant.

Proof. As $O \in \mathcal{I}$, these inequalities hold at the beginning of Algorithm 1. To see this, note that $s_i = |V_i \cap O^{in}|$ and $b_i + \sum_{T_j \text{ below } T_i} b_j \leq |V_i \cap O^{out}|$. For the induction step, observe that Line 6 guarantees that the node T_i selected is the only one with a non-zero b_i value in the subtree of root T_i . As a result, when the set \bar{U}_v is allocated, for the nodes T_j between T_i and the allocated node, the augmentation by one of the value s is not an issue because these nodes are chosen to be non-fully-booked, *i.e.* $k_j > s_j$. For the nodes T_j above T_i , the value s_j are increased by one but as b_i was decreased by one, the total value $s_j + b_j + \sum_{T_{j'} \text{ below } T_j} b_{j'}$ remains unchanged. \triangleleft

▷ **Claim 10.** If $B_i \neq \emptyset$, T_i contains at least $2t$ usable vertices. Consequently, Algorithm 1 (Lines 6-11) always build the set \bar{U}_v of size at least t .

Proof. In fact, as $B_i \neq \emptyset$, b_i is still non-zero, and by Claim 9 we get $k_i - s_i \geq b_i$, so V'_i contains at least $2t$ usable vertices because of Claim 8. Then the set \bar{U}_v can be built as required, containing at least t elements. \triangleleft

The above claim lower-bounds the size of each \bar{U}_v . The fact that the constructed \bar{U}_v s are mutually disjoint follows from the algorithm (Line 12). Now we want to show that the \bar{U}_v s have the desired properties regarding independence and weighted degrees.

▷ **Claim 11.** At any time during the execution of Algorithm 1, if we build a set S by taking the elements of O^{in} and one arbitrary element in each set \bar{U}_v that has already been constructed, then S is independent. Moreover, for a node T_i that does not contain only unusable vertices in V'_i , any arbitrary choice of elements in the \bar{U}_v s will lead to the equality $|S \cap V'_i| = s_i$.

Proof. We proceed by induction. These properties are clearly satisfied at the beginning of the algorithm (because then $S = O^{in}$). Now suppose that these properties hold at some time, and then we allocate a new set $\bar{U}_{v'}$ for some $v' \in O^{out}$. Let S be made of O^{in} and an arbitrary choice for the \bar{U}_v s that were constructed so far (excluding $\bar{U}_{v'}$). By the induction hypothesis, $S \in \mathcal{I}$. The vertices of $\bar{U}_{v'}$ are supposed to be usable, so the nodes containing them are not fully-booked and these nodes contain usable vertices. By induction on the second part of the claim, a usable element in such a node T_j can be selected, as any choice for the other \bar{U}_v s will use exactly $s_j < k_j$ vertices of the laminar constraint of that node. Therefore any vertex $u \in V_{v'}$ added to S does not cause any constraint to be violated, and $S \cup \{u\} \in \mathcal{I}$. Let T_i be the node used for the allocation at Line 11 of Algorithm 1. All the nodes in the subtree of root T_i will be subsequently ignored by the algorithm, as all the nodes inside it are marked as unusable. The values s_j of the nodes T_j in that subtree are not updated by the algorithm, but it is not an issue given that the second part of the claim does not affect them. The nodes above T_i are updated, and it is true that for any vertex chosen in $\bar{U}_{v'}$, that vertex will count in the laminar inequalities for these nodes as a +1. This concludes the induction. \triangleleft

▷ **Claim 12.** For all $v \in O \setminus V'$, for all $u \in \bar{U}_v$, it holds that $\deg_w(u) \geq \deg_w(v)$.

Proof. By construction, $\overline{U}_v \subseteq V'_i$ where T_i is the blocking node of v . As T_i is the blocking node of v , all the elements in V'_i have a larger weighted degree than v . \triangleleft

Finally we construct the sets U_v s by choosing arbitrarily t vertices from \overline{U}_v . By Claims 10, 11, and 12, they satisfy the properties of robustness in Definition 4.

3.3 Transversal Matroids

Recall that a transversal matroid $\mathcal{M} = (V, \mathcal{I})$ can be represented as a bipartite graph $G = (A \cup V, E)$ and $A = \{A_1, \dots, A_k\}$. A subset $V' \subseteq V$ is independent in \mathcal{M} if and only if there is a matching where all of V' are matched to some subset of A . Let $t \in \mathbb{N}$. The matroid union $(t+k-1)\mathcal{M}$ can be regarded as making the capacity of each vertex A_i in A increased to $t+k-1$ (equivalently, create $t+k-1$ copies of each A_i and modify the edge set E accordingly). Our algorithm is as follows. Again process the vertices in non-increasing order of their weighted degrees. We start with an empty matching, and we maintain a matching throughout the execution of the algorithm. For each new vertex $v \in V$, try to find an augmenting path so that it can be matched. If we cannot find such a path, v is discarded. At any time during the execution of the algorithm, the current kernel $V' \subseteq V$ is simply the set of vertices in V that are matched in the current matching. We can observe that a vertex in V' cannot be evicted once it belongs to V' . In the following, we write $V'_i \subseteq V'$ to denote the vertices in V that are matched to A_i in the current kernel V' .

First we argue that our procedure is the same as the greedy described in Theorem 6 applied on $(t+k-1)\mathcal{M}$. We need to show that a vertex v , if discarded, is spanned by vertices in V' that arrived earlier than it. To see this, observe that, at the moment v arrives, V' is independent in $(t+k-1)\mathcal{M}$. Moreover, as we cannot find an augmenting path when v is added to V' , it means that $V' + v$ is not independent, *i.e.* v is spanned by V' and this holds until the end of the algorithm.

Now let us consider the robustness. Let $O = \{o_1, \dots, o_k\}$ be an arbitrary base in \mathcal{M} . We can assume that o_i is assigned to A_i for all i in the corresponding matching. For an element $o_i \in O \setminus V'$, when it was discarded, some $V'_i \subseteq V'$ elements (exactly $t+k-1$) that arrived earlier were already assigned to A_i . As no augmenting path could go through A_i when o_i was discarded, the set of elements assigned to A_i would not have changed till the end: otherwise, that would mean that at some point an augmenting path passed through A_i , which is not possible as o_i was discarded because no augmenting path passing through A_i was found at that point. As a result the $t+k-1$ elements of V'_i assigned to A_i are all of weighted degrees larger than that of o_i . As $|V'_i \cap O| \leq |V' \cap O| < k$ there remains at least t elements of V'_i that can be used to build a set U_{o_i} of cardinality t as stated in Definition 4.

4 Steaming Algorithms

In this section, we turn our algorithms into streaming form.

First, we show that it is easy to compute a $(1 - \varepsilon)$ approximation in two passes, using $O(n + \tau^2)$ space (τ depends on the type of matroids involved, as defined in Theorem 6). In the first pass, we compute the weighted degrees $\deg_w(v)$ of all vertices v to define the kernel $V' \subseteq V$. This requires $O(n)$ space. In the second pass, we retain a subset of edges $E' \subseteq E$, those both of whose end-points are in V' . Easily $|E'| = O(\tau^2)$. Using E' , we can compute the exact value $E_G(S) = \sum_{v \in S} \deg_w(v) - \sum_{e \in (S \times S) \cap E'} w(e)$ for each feasible independent set $S \subseteq V'$. Then an enumeration of all such sets gives the desired $(1 - \varepsilon)$ approximation.

We now explain how to achieve the same goal in one pass, at the expense of higher space requirement.

► **Theorem 13.** *In the edge arrival streaming model (each edge appearing exactly once in the stream), one can extract a $(1 - \varepsilon)$ -approximate solution of the matroid-constrained maximum vertex cover using $O(\frac{nk}{\varepsilon})$ variables for uniform, partition, laminar, and transversal matroids.*

Proof. Let $\varepsilon > 0$. During the streaming phase, we keep track of the weighted degrees of all the vertices, as well as for each vertex v the set of the $\frac{2k}{\varepsilon}$ edges incident to v that have the largest weight. We denote the set of memorized edges as E' .

Then, we can choose, depending on the type of matroid, the value τ corresponding to the right type of matroid (as prescribed in Theorem 6) for the parameter $\frac{\varepsilon}{2}$ and we build the kernel V' that is supposed to contain a $(1 - \frac{\varepsilon}{2})$ approximation of the maximum cover. However, we do not know all the edges between the elements in V' , as only the $\frac{2k}{\varepsilon}$ heaviest incident edges are known for each vertex.

We will compute the value of S *pretending* that the edges in $((S \times S) \cap E) \setminus E'$ are not present. Precisely, for each set $S \subseteq V$, we define

$$\tilde{E}_G(S) = \sum_{v \in S} \deg_w(v) - \sum_{e \in ((S \times S) \cap E) \setminus E'} w(e) = E_G(S) + \sum_{e \in ((S \times S) \cap E) \setminus E'} w(e).$$

Notice that $\tilde{E}_G(S) \geq E_G(S)$. Let $S^* \subseteq V'$ be the independent set reaching the maximum $\tilde{E}_G(S^*)$. This set S^* will be our final output. We next lower-bound its real value $E_G(S^*)$.

Let O denote the original optimal solution (with respect to the entire graph), and S' denote the optimal vertex cover in the kernel V' (also with respect to the entire graph), so that $S' \subseteq V'$, $S' \in \mathcal{I}$, and $E_G(S') \geq (1 - \frac{\varepsilon}{2}) \cdot E_G(O)$. Then

$$\tilde{E}_G(S^*) \geq \tilde{E}_G(S') \geq E_G(S') \geq \left(1 - \frac{\varepsilon}{2}\right) \cdot E_G(O).$$

To compare the real value of $E_G(S^*)$ with $\tilde{E}_G(S^*)$, we just need to compute the total weight of the edges in $((S^* \times S^*) \cap E) \setminus E'$:

$$\begin{aligned} \sum_{(u,v) \in ((S^* \times S^*) \cap E) \setminus E'} w(u,v) &= \frac{1}{2} \sum_{v \in S^*} \left(\sum_{u \in S^*: (u,v) \in E \setminus E'} w(u,v) \right) \\ &\leq \frac{1}{2} \sum_{v \in S^*} k \cdot \deg_w(v) \cdot \frac{\varepsilon}{2k} \\ &= \frac{\varepsilon}{4} \sum_{v \in S^*} \deg_w(v) \leq \frac{\varepsilon}{2} \cdot \tilde{E}_G(S^*), \end{aligned}$$

where the first inequality comes from the fact that the edges that are not among the $\frac{2k}{\varepsilon}$ heaviest edges incident on v must be of weight at most $\deg_w(v) \cdot \frac{\varepsilon}{2k}$. Therefore the real value $E_G(S^*)$ is at least $(1 - \frac{\varepsilon}{2}) \cdot \tilde{E}_G(S^*) \geq (1 - \frac{\varepsilon}{2})^2 \cdot E_G(O) \geq (1 - \varepsilon) \cdot E_G(O)$. ◀

Next we consider a particular kind of stream of edges, where each edge appears twice: given an arbitrary order of the vertices, for each vertex, all its incident edges are given in a row. For this *incidence streaming model* [5] (sometimes called *adjacency list model* [29]), the next theorem shows that we can use much less space with just a single pass.

► **Theorem 14.** *In the incidence streaming model, one can extract a $(1 - \varepsilon)$ -approximate solution using $O((\frac{k}{\varepsilon})^2)$ variables for uniform, partition, laminar, and $O((\frac{k}{\varepsilon} + k)^2)$ for transversal matroids.*

Proof. Let $\varepsilon > 0$. Given the type of the matroid \mathcal{M} , choose the corresponding value of τ as prescribed in Theorem 6. Start with an empty kernel $V' = \emptyset$. Through the execution of the algorithm, V' will contain the largest independent set in $\tau\mathcal{M}$ with respect to the sum of the weighted degrees. When we process a vertex v (*i.e.* its set of incident edges) we can compute its weighted degree $\deg_w(v)$ and store the edges linking v to elements of V' . If $V' + v$ is not independent in $\tau\mathcal{M}$, consider the element with the smallest weighted degree u in the circuit formed in $V' + v$. Then, set $V' \leftarrow (V' + v) - u$. If v is added into V' , we keep in memory all the edges linking v to other vertices of V' . When an element is discarded or evicted from V' , all its incident edges are deleted. As a result, at any time during the execution of the algorithm, only $O(\tau^2)$ edges are stored, so the overall memory consumption is $O(\tau^2)$.

In the end, we obtain exactly the approximate kernels described in the previous sections, and because we know all the values of the weighted degrees of V' as well as the weights of the edges between them we can find the largest vertex cover in that kernel using bruteforce. ◀

► **Remark 15.** This model has an interesting interpretation in the context of coverage function⁴ maximization in the streaming setting – here the sets arrive over time in such a way that the values of singletons $f(\{v\})$, for $v \in V$, are revealed one by one. In case where a coverage function has bounded frequency larger than 2, we also present in Appendix B a streaming algorithm.

5 Conclusion and Open Questions

Theorem 6 allows us to generalize the cardinality constraint to some special cases of matroid constraints, and these ideas could be useful for other kernelization algorithms. Regarding the bounds of Theorem 6, tight examples can be built to show that the values of τ provided are the best possible for partition and laminar matroids. For transversal matroids it is less clear whether the bound for τ can be improved or not. The most important open question is obviously whether Theorem 6 can be generalized to other types of matroids.

References

- 1 Alexander A. Ageev and Maxim I. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *IPCO 1999*, 1999.
- 2 Per Austrin and Aleksa Stankovic. Global cardinality constraints make approximating some max-2-csps harder. In *APPROX/RANDOM 2019*, pages 24:1–24:17, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.24.
- 3 M. Babaioff, N. Immorlica, and R. Kleinberg. Matroids, secretary problems, and online mechanisms. In *SODA*, pages 434–443, 2007.
- 4 Édouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Georgios Stamoulis. Purely combinatorial approximation algorithms for maximum k-vertex cover in bipartite graphs. *Discrete Optimization*, 27:26–56, 2018.

⁴ A coverage function f over a ground set $\{1, \dots, m\}$, associated with a universe U of weighted elements and m sets A_1, \dots, A_m , where $A_i \subseteq U$ for all i , is defined over all $S \subseteq \{1, \dots, m\}$ so that $f(S)$ is the sum of the weight of the elements in $\cup_{i \in S} A_i$. The frequency of an element of the universe is the number of sets A_i it appears in. Here in our problem of maximum vertex cover, the vertices correspond to the ground set and the edges to the universe U . Note that for a vertex cover the frequency (the maximum number of sets where an element of the universe appears in) is exactly 2, as an edge has only two endpoints.

- 5 Vladimir Braverman, Zaoxing Liu, Tejasvam Singh, N. V. Vinodchandran, and Lin F. Yang. New bounds for the CLIQUE-GAP problem using graph decomposition theory. *Algorithmica*, 80(2):652–667, 2018. doi:10.1007/s00453-017-0277-5.
- 6 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *Proc. 12th IPCO*, pages 182–196, 2007. doi:10.1007/978-3-540-72792-7_15.
- 7 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.
- 8 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids and more. In *Proc. 17th IPCO*, pages 210–221, 2014.
- 9 S. Chakraborty and O. Lachish. Improved competitive ratio for the matroid secretary problem. In *SODA*, pages 1702–1712, 2012.
- 10 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Proc. 42nd ICALP*, pages 318–330, 2015.
- 11 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *SODA 2016*, pages 1326–1344, 2016. doi:10.1137/1.9781611974331.ch92.
- 12 William H Cunningham. Testing membership in matroid polyhedra. *Journal of Combinatorial Theory, Series B*, 36(2):161–188, 1984.
- 13 J. Edmonds and D.R. Fulkerson. Transversals and matroid partition. *Journal of Research National Bureau of Standards Section B*, 69:147–153, 1965.
- 14 Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211, 2001.
- 15 M. Feldman, O. Svensson, and R. Zenklusen. A simple $o \log \log(\text{rank})$ -competitive algorithm for the matroid secretary problem. In *SODA*, pages 1189–1201, 2015.
- 16 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. Streaming submodular maximization with matroid and matching constraints. *CoRR*, abs/2107.07183, 2021. arXiv:2107.07183.
- 17 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.
- 18 Yuval Filmus and Justin Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *Proc. 53rd FOCS*, 2012.
- 19 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of generalized vertex cover problems. In *Algorithms and Data Structures*, pages 36–48, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 20 Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for k-cut. In *FOCS 2018*, pages 113–123. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00020.
- 21 Qiaoming Han, Yinyu Ye, Hantao Zhang, and Jiawei Zhang. On approximation of max-vertex-cover. *Eur. J. Oper. Res.*, 143(2):342–355, 2002. doi:10.1016/S0377-2217(02)00330-2.
- 22 Qiaoming Han, Yinyu Ye, and Jiawei Zhang. An improved rounding method and semidefinite programming relaxation for graph partition. *Math. Program.*, 92(3):509–535, 2002. doi:10.1007/s101070100288.
- 23 Dorit S Hochbaum and Anu Pathria. Analysis of the greedy approach in covering problems. *Naval Research Quarterly*, 45:615–627, 1998.
- 24 S. Im and Y. Wang. Secretary problems: laminar matroids and interval scheduling. In *SODA*, pages 1265–1274, 2011.
- 25 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *STOC 2017*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

- 26 Pasin Manurangsi. A Note on Max k -Vertex Cover: Faster FPT-AS, Smaller Approximate Kernel and Improved Approximation. In *SOSA 2019*, pages 15:1–15:21, 2018.
- 27 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k -coverage, unique set cover and related problems (via t -wise agreement testing theorem). In *SODA*, pages 62–81, 2020.
- 28 Dániel Marx. Parameterized Complexity and Approximation Algorithms. *The Computer Journal*, 51(1):60–78, July 2008.
- 29 Andrew McGregor, David Tench, and Hoa T. Vu. Maximum coverage in the data stream model: Parameterized and generalized. In *ICDT 2021*, volume 186, pages 12:1–12:20, 2021. doi:10.4230/LIPIcs.ICDT.2021.12.
- 30 L. Mirsky and H. Perfect. Applications of the notion of independence to problems of combinatorial analysis. *Journal of Combinatorial Theory*, 2:327–357, 1965.
- 31 S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc, January 2005.
- 32 András Recski. *Matroid Theory and its Applications in Electric Network Theory and in Statics*. Springer-Verlag, 1989.
- 33 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003.
- 34 J. Soto. Matroid secretary problem in the random assignment model. *SIAM Journal on Computing*, 42:178–211, 2013.
- 35 D.J.A Welsh. Transversal theory and matroids. *Canadian Journal of Mathematics*, 21:1323–1302, 1969.

A An FPT-AS for Hypergraphs

Suppose that we are given a hypergraph $G = (V, E)$ with edge size bounded by a constant $\eta \geq 2$. We proceed as in Section 2. First, notice that:

$$E_G(S) = E_G(O^{in}) + E_G(U^*) - E_G(O^{in}, U^*).$$

We bound $\mathbb{E}[E_G(O^{in}, U^*)]$ as follows. By construction, $\mathbb{P}[u \in U^*] = \varepsilon$ for all $u \in U$. Then,

$$\begin{aligned} \mathbb{E}[E_G(O^{in}, U^*)] &= \sum_{e \in E, e \cap O^{in} \neq \emptyset} w(e) \cdot \mathbb{1}[e \cap U^* \neq \emptyset] \leq \sum_{e \in E, e \cap O^{in} \neq \emptyset} w(e) \cdot (\eta - 1) \cdot \varepsilon \\ &= \varepsilon \cdot (\eta - 1) \cdot \mathbb{E}[E_G(O^{in})]. \end{aligned}$$

using union bound and the fact that at most $\eta - 1$ endpoints can be in U . Furthermore, the value $\mathbb{E}[E_G(U^*)]$ can be rearranged as follows:

$$\begin{aligned} \mathbb{E}[E_G(U^*)] &= \mathbb{E} \left[\sum_{u \in U^*} \left(\deg_w(u) - \sum_{e \in \delta(u)} w(e) \cdot \frac{|e \cap U^*| - 1}{|e \cap U^*|} \right) \right] \\ &\geq \mathbb{E} \left[\sum_{u \in U^*} \left(\deg_w(u) - \sum_{e \in \delta(u)} w(e) \cdot \frac{\eta - 1}{\eta} \cdot \mathbb{1}[e \cap U^* \setminus \{u\} \neq \emptyset] \right) \right] \\ &\geq \mathbb{E} \left[\sum_{u \in U} \left(\deg_w(u) \cdot \mathbb{1}[u \in U^*] - \sum_{e \in \delta(u)} w(e) \cdot \frac{\eta - 1}{\eta} \cdot \mathbb{1}[u \in U^* \wedge e \cap U^* \setminus \{u\} \neq \emptyset] \right) \right] \\ &\geq \mathbb{E} \left[\sum_{u \in U} \left(\deg_w(u) \cdot \varepsilon - \frac{\eta - 1}{\eta} \sum_{e \in \delta(u)} w(e) \cdot (\eta - 1) \cdot \varepsilon^2 \right) \right] \end{aligned}$$

$$\geq \varepsilon \cdot (1 - \varepsilon \cdot (\eta - 1)) \left(\sum_{u \in U} \deg_w(u) \right).$$

Recall that by robustness, for all $u \in O^{out}$, the elements of U_u have weighted degree no less than the one of u . Therefore,

$$\begin{aligned} \mathbb{E}[E_G(U^*)] &\geq \varepsilon(1 - \varepsilon \cdot (\eta - 1)) \left(\sum_{u \in O^{out}} \sum_{v \in U_u} \deg_w(v) \right) \\ &\geq \varepsilon(1 - \varepsilon \cdot (\eta - 1)) \left(\sum_{u \in O^{out}} \frac{1}{\varepsilon} \cdot \deg_w(u) \right) \\ &\geq (1 - \varepsilon \cdot (\eta - 1)) \cdot E_G(O^{out}). \end{aligned}$$

As a result, we get:

$$\begin{aligned} \mathbb{E}[E_G(S)] &\geq E_G(O^{in}) + (1 - \varepsilon \cdot (\eta - 1)) \cdot E_G(O^{out}) - \varepsilon \cdot (\eta - 1) \cdot E_G(O^{in}) \\ &\geq (1 - \varepsilon \cdot (\eta - 1)) \cdot E_G(O). \end{aligned}$$

By averaging principle, there exists a set $S \subseteq V'$ such that $S \in \mathcal{I}$ and such that $E_G(S) \geq (1 - (\eta - 1) \cdot \varepsilon) \cdot E_G(O)$.

B Streaming Algorithm for Hypergraphs

Here we suppose that we are given a hypergraph $G = (V, E)$ with edge size bounded by a constant $\eta \geq 2$ as an adjacency list stream. Using the idea of Theorem 14 and the result of Appendix A, one can get in the incidence streaming model a $(1 - (\eta - 1) \cdot \varepsilon)$ approximation using $O(\tau^\eta)$ memory, where τ depends on the type of matroid, as prescribed in Theorem 6. In fact, we can maintain through the execution of the algorithm for each subset of at most η elements $S \subseteq V'$ the sum of the weight of the hyper-edges e such that $e \cap V' = S$ (just like in the proof of Theorem 14 where we keep track of these values for pairs in V'). For instance, for a uniform, a partition, or a laminar matroid, we could get a $(1 - \varepsilon)$ approximation using $O((\frac{2\eta \cdot k}{\varepsilon})^\eta)$ variables. This shows that for the special matroids that we studied of rank k , a weighted coverage function with bounded frequency η can be $(1 - \varepsilon)$ approximated in streaming. This extends a result of [29] to matroids.

Non-Uniform k -Center and Greedy Clustering

Tanmay Inamdar ✉ 

Department of Informatics, University of Bergen, Norway

Kasturi Varadarajan ✉

Department of Computer Science, University of Iowa, Iowa City, IA, USA

Abstract

In the Non-Uniform k -Center (NU k C) problem, a generalization of the famous k -center clustering problem, we want to cover the given set of points in a metric space by finding a placement of balls with specified radii. In t -NU k C, we assume that the number of distinct radii is equal to t , and we are allowed to use k_i balls of radius r_i , for $1 \leq i \leq t$. This problem was introduced by Chakrabarty et al. [ACM Trans. Alg. 16(4):46:1-46:19], who showed that a constant approximation for t -NU k C is not possible if t is unbounded, assuming $P \neq NP$. On the other hand, they gave a bicriteria approximation that violates the number of allowed balls as well as the given radii by a constant factor. They also conjectured that a constant approximation for t -NU k C should be possible if t is a fixed constant. Since then, there has been steady progress towards resolving this conjecture – currently, a constant approximation for 3-NU k C is known via the results of Chakrabarty and Negahbani [IPCO 2021], and Jia et al. [SOSA 2022]. We push the horizon by giving an $O(1)$ -approximation for the Non-Uniform k -Center for 4 distinct types of radii. Our result is obtained via a novel combination of tools and techniques from the k -center literature, which also demonstrates that the different generalizations of k -center involving non-uniform radii, and multiple coverage constraints (i.e., *colorful k -center*), are closely interlinked with each other. We hope that our ideas will contribute towards a deeper understanding of the t -NU k C problem, eventually bringing us closer to the resolution of the CGK conjecture.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Rounding techniques

Keywords and phrases k -center, approximation algorithms, non-uniform k -center, clustering

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.28

Related Version *Full Version*: <https://arxiv.org/abs/2111.06362>

Funding *Tanmay Inamdar*: Supported by the European Research Council (ERC) via grant LOPPRE, reference 819416.

Kasturi Varadarajan: Supported by National Science Foundation (NSF) Award CCF-1615845.

1 Introduction

The k -center problem is one of the most fundamental problems in clustering. The input to the k -center problem consists of a finite metric space (X, d) , where X is a set of n points, and $d : X \times X \rightarrow \mathbb{R}^+$ is the associated distance function satisfying triangle inequality. We are also given a parameter k , where $1 \leq k \leq n$. A solution to the k -center problem consists of a set $C \subseteq X$ of size at most k , and the cost of this solution is $\max_{p \in X} d(p, C)$, i.e., the maximum distance of a point to its nearest center in C . Alternatively, a solution can be thought of as a set of k balls of radius $\max_{p \in X} d(p, C)$, centered around points in C , that covers the entire set of points X . The goal is to find a solution of smallest radius. We say that a solution C' is an α -approximation, if the cost of C' is at most α times the optimal radius. Several 2-approximations are known for the k -center problem [10, 9]. A simple reduction from the Minimum Dominating Set problem shows that the k -center problem is NP-hard. In fact, the same reduction also shows that it is NP-hard to get a $(2 - \epsilon)$ -approximation for any $\epsilon > 0$.



© Tanmay Inamdar and Kasturi Varadarajan;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 28; pp. 28:1–28:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Several generalizations of the vanilla k -center problem have been considered in the literature, given its fundamental nature in the domain of clustering and approximation algorithms. One natural generalization is the *Robust k -center* or *k -center with outliers* problem, where we are additionally given a parameter m , and the goal is to find a solution that covers at least m points of X . Note that the remaining at most $n - m$ points can be thought of as outliers with respect to the clustering computed. Charikar et al. [7], who introduced this problem, showed that a simple greedy algorithm gives a 3-approximation for the problem. Subsequently, the approximation guarantee was improved by [4, 8], who gave a 2-approximation, which is optimal in light of the aforementioned $(2 - \epsilon)$ -hardness result.

The focus of our paper is the Non-Uniform k -Center (NUkC), which was introduced by Chakrabarty et al. [4]. A formal definition follows.

► **Definition 1** (*t*-NUkC). *The input is an instance $\mathcal{I} = ((X, d), (k_1, k_2, \dots, k_t), (r_1, r_1, \dots, r_t))$, where $r_1 \geq r_2 \geq \dots \geq r_t \geq 0$, and the k_i are positive integers. The goal is to find sets $C_i \subseteq X$ for $1 \leq i \leq t$, such that $|C_i| \leq k_i$, and the union of balls of radius αr_i around the centers in C_i , over $1 \leq i \leq t$, covers the entire set of points X . The objective is to minimize the value of the dilation factor α .*

In the Robust *t*-NUkC problem, we are required to cover at least m points of X using such a solution. We note that the special case of (Robust) *t*-NUkC with $t = 1$ corresponds to the (Robust) k -center problem. Chakrabarty et al. [4] gave a bicriteria approximation for *t*-NUkC for arbitrary t , i.e., they give a solution containing $O(k_i)$ balls of radius $O(\alpha^*)r_i$ for $1 \leq i \leq t$, where α^* is the optimal dilation.¹ They also give a $(1 + \sqrt{5})$ -approximation for 2-NUkC. Furthermore, they conjectured that there exists a polynomial-time $O(1)$ -approximation for *t*-NUkC for constant t . Subsequently, Chakrabarty and Negahbani [6] made some progress by giving a 10-approximation for Robust 2-NUkC. Very recently, Jia et al. [12] showed an approximate equivalence between $(t + 1)$ -NUkC and Robust *t*-NUkC, thereby observing that the previous result of [6] readily implies a 23-approximation for 3-NUkC. We note that the techniques from Inamdar and Varadarajan [11] implicitly give an $O(1)$ -approximation for *t*-NUkC for any $t \geq 1$, in $k^{O(k)} \cdot n^{O(1)}$ time, where $k = \sum_t k_t$. That is, one gets an *FPT approximation*. Finally, we also note that Bandyapadhyay [2] gave an exact algorithm for perturbation resilient instances of NUkC in polynomial time.

Another related variant of k -center is the *Colorful k -center* problem. Here, the set of points X is partitioned into ℓ color classes, $X_1 \sqcup \dots \sqcup X_\ell$. Each color class X_j has a coverage requirement m_j , and the goal is to find a set of k balls of smallest radius that satisfy the coverage requirements of all the color classes. Note that this is a generalization of Robust k -center to multiple types of coverage constraints. Bandyapadhyay et al. [3] introduced this problem, and gave a *pseudo-approximation*, i.e., their algorithm returns an 2-approximate solution using at most $k + \ell - 1$ centers. Furthermore, they managed to improve this to a true $O(1)$ -approximation in the Euclidean plane for constant number of color classes. Subsequently, Jia et al. [13] and Anegg et al. [1] independently gave (true) 3 and 4-approximations respectively for this problem (with constant ℓ) in arbitrary metrics.

¹ Chakrabarty et al. [4] use a slightly different, but equivalent, formulation of NUkC: the input contains a sequence of allowed radii $r_1 \geq r_2 \geq \dots \geq r_k$, and we are allowed to use one ball of radius αr_i , for each i . In this setting, they design an approximation algorithm that returns a solution containing $O(1)$ balls of radius $O(\alpha^*)r_i$, for each i . When there are only t distinct classes of radii, this is equivalent to the result as formulated above.

Our Results and Techniques. Our main result is an $O(1)$ -approximation for 4-NUkC. We obtain this result via a sequence of reductions; some of these reductions are from prior work while some are developed here and constitute our main contribution. Along the way, we combine various tools and techniques from the aforementioned literature of Robust, Colorful, and Non-Uniform versions of k -center.

First, we reduce the 4-NUkC problem to the Robust 3-NUkC problem, following Jia et al. [12]. Next, we reduce the Robust 3-NUkC to *well-separated* Robust 3-NUkC, by adapting the approach of Chakrabarty and Negahbani [6].² In a well-separated instance, we are given a set of potential centers for the balls of radius r_1 , such that the distance between any two of these centers is at least $c \cdot r_1$, for a parameter $c \geq 2$.

To solve Well-Separated Robust 3-NUkC, we give a sequence of reductions, which constitute the technical core of our paper. First, we show that any instance of Robust t -NUkC can be transformed to an instance of “Colorful” $(t - 1)$ -NUkC, where we want to cover certain number of red and blue points using the specified number of balls of $t - 1$ distinct radii. Thus, this reduction reduces the number of radii classes from t to $t - 1$ at the expense of increasing the number of coverage constraints from 1 to 2. In our next reduction, we show that Colorful $(t - 1)$ -NUkC can be reduced to Colorful $(t - 1)$ -NUkC with an additional “self-coverage” property, i.e., the radius r_{t-1} can be assumed to be 0. Just like the aforementioned reduction from [12], these two reductions are generic, and hold for any value of $t \geq 2$. These reductions crucially appeal to the classical greedy algorithm and its analysis from Charikar et al. [7], which is a tool that has not been exploited in the NUkC literature thus far. We believe that these connections between Colorful and Robust versions of NUkC are interesting in their own right, and may be helpful toward obtaining a true $O(1)$ -approximation for t -NUkC for any fixed t . Indeed, one possible avenue to this result is to find suitable generalizations of some of our reductions.

We apply these two new reductions to transform Well-Separated Robust 3-NUkC to *Well-Separated Colorful* 2-NUkC, with $r_2 = 0$. The latter problem can be solved in polynomial time using dynamic programming in a straightforward way – the details can be found in Section 5. Since each of our reductions preserves the approximation factor up to a constant, this implies an $O(1)$ -approximation for 4-NUkC.

Our overall algorithm for 4-NUkC is combinatorial, except for the step where we reduce Robust 3-NUkC to Well-Separated Robust 3-NUkC using the round-or-cut approach of [6]. Thus, we avoid an additional “inner loop” of round-or-cut that is employed in recent work [6, 12].³

We conclude this section by explaining the bottleneck in employing the techniques used in this paper to obtain an $O_t(1)$ approximation for t -NUkC. For concreteness, we focus on 5-NUkC. Using the general reductions from prior work and this paper, we can reduce 5-NUkC to Well-Separated Colorful 3-NUkC, with $r_3 = 0$ (self-coverage at level 3). However, we do not know how to solve this problem exactly via DP, or even obtain an $O(1)$ approximation. The difficulty pertains to the balls of radius $r_2 > 0$, which are not constrained in any way. It is not straightforward to use our techniques to also obtain self-coverage at level 2 ($r_2 = 0$); doing so would result in a problem that can be solved exactly using DP.

² In this discussion, “reduction” refers to a polynomial time (possibly Turing) reduction from problem A to problem B , such that (i) a feasible instance of A yields (possibly polynomially many) instance(s) of B , and (ii) a constant approximation for B implies a constant approximation for A .

³ A by-product of one of our reductions is a purely combinatorial approximation algorithm for colorful k -center, in contrast with the LP-based approaches in [3, 1, 13].

2 Definitions, Main Result, and Greedy Clustering

2.1 Problem Definitions

In the following, we set up the basic notation and define the problems we will consider in the paper. We consider a finite metric space (X, d) , where X is a finite set of (usually n) points, and d is a distance function satisfying triangle inequality. If Y is a subset of X , then by slightly abusing the notation, we use (Y, d) to denote the metric space where the distance function d is restricted to the points of Y . Let $p \in X$, $Y \subseteq X$, and $r \geq 0$. Then, we use $d(p, Y) := \min_{y \in Y} d(p, y)$, and denote by $B(p, r)$ the ball of radius r centered at p , i.e., $B(p, r) := \{q \in X : d(p, q) \leq r\}$. We say that a ball $B(p, r)$ covers a point q iff $q \in B(p, r)$; a set of balls \mathcal{B} (resp. a tuple of sets of balls $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t)$) covers q if there exists a ball in \mathcal{B} that covers q (resp. $\bigcup_{1 \leq i \leq t} \mathcal{B}_i$ that covers q). Analogously, a set of points $Y \subseteq X$ is covered iff every point in Y is covered. For a function $f : S \rightarrow \mathbb{R}^+$ or $f : S \rightarrow \mathbb{N}$, and $R \subseteq S$, we define $f(R) := \sum_{r \in R} f(r)$.

► **Definition 2** (Decision Version of t -NUkC).

The input is an instance $\mathcal{I} = ((X, d), (k_1, k_2, \dots, k_t), (r_1, r_2, \dots, r_t))$, where $r_1 \geq r_2 \geq \dots \geq r_t \geq 0$, and each k_i is a non-negative integer. The goal is to determine whether there exists a solution $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t)$, where for each $1 \leq i \leq t$, \mathcal{B}_i is a set with at most k_i balls of radius r_i , that covers the entire set of points X . Such a solution is called a feasible solution, and if the instance \mathcal{I} has a feasible solution, then \mathcal{I} is said to be feasible.

An algorithm is said to be an α -approximation algorithm (with $\alpha \geq 1$), if given a feasible instance \mathcal{I} , it returns a solution $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t)$, where for each $1 \leq i \leq t$, \mathcal{B}_i is a collection of at most k_i balls of radius αr_i , such that the solution covers X .

Next, we define the robust version of t -NUkC.

► **Definition 3** (Decision Version of Robust t -NUkC).

The input is an instance $\mathcal{I} = ((X, d), (\omega, m), (k_1, k_2, \dots, k_t), (r_1, r_2, \dots, r_t))$. The setup is the same as in t -NUkC, except for the following: $\omega : X \rightarrow \mathbb{Z}^+$ is a weight function, and $1 \leq m \leq \omega(X)$ is a parameter. The goal is to determine whether there exists a feasible solution, i.e., $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t)$ of appropriate sizes and radii (as defined above), such that the total weight of the points covered is at least m . An α -approximate solution covers points of weight at least m while using at most k_i balls of radius αr_i for each $1 \leq i \leq t$.

We will frequently consider the *unweighted* version of Robust t -NUkC, i.e., where the weight of every point in X is unit. Let $\mathbb{1}$ denote this unit weight function. Now we define the Colorful t -NUkC problem, which generalizes Robust t -NUkC.

► **Definition 4** (Decision Version of Colorful t -NUkC).

The input is an instance $\mathcal{I} = ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, k_2, \dots, k_t), (r_1, r_2, \dots, r_t))$. The setup is similar as in Robust t -NUkC, except that we have two weight functions $\omega_r, \omega_b : X \rightarrow \mathbb{Z}^+$ (corresponding to red and blue weight respectively). A feasible solution covers a set of points with red weight at least m_r , and blue weight at least m_b . The notion of approximation is the same as above.

We note that the preceding definition naturally extends to an arbitrary number $\chi \geq 2$ of colors (i.e., χ different weight functions over X). However, we will not need that level of generality in this paper.

2.2 Main Algorithm for 4-NUkC

Let $\mathcal{I} = ((X, d), (k_1, \dots, k_4), (r_1, \dots, r_4))$ be the given instance of 4-NUkC, which we assume is feasible. First, using a reduction from [12] (or a variant described in Section 6), we reduce \mathcal{I} to an instance $\mathcal{I}' = ((X, d), (\mathbb{1}, m)(r'_1, r'_2, r'_3), (k_1, k_2, k_3))$ of Robust 3-NUkC. The reduction has the property that \mathcal{I}' is feasible, and furthermore an $O(1)$ -approximation for \mathcal{I}' implies an $O(1)$ -approximation for \mathcal{I} .

Next, we perform a Turing reduction from Robust 3-NUkC to *Well-Separated* Robust 3-NUkC, by adapting a very similar reduction of [6]. In a well-separated instance, we are given a set of potential centers for the balls of radius r'_1 , such that the distance between any two potential centers is at least $3r'_1$. This Turing reduction is described in Sections A and B of the appendix, and uses the round-or-cut methodology on the instance \mathcal{I}' . At a high level, we run the ellipsoid algorithm, and each iteration of the ellipsoid algorithm returns a candidate LP solution such that, (1) it can be rounded to obtain an $O(1)$ -approximate solution for \mathcal{I}' , or (2) one can obtain polynomially many instances of *well-separated* Robust 3-NUkC, at least one of which is feasible, or (3) If none of the obtained instances is feasible, then one can obtain a hyperplane separating the LP solution from the integer hull of coverages.

Solving a Well-Separated Instance. For the sake of simplicity let \mathcal{J} be one of the instances of Well-Separated Robust 3-NUkC, along with a well-separated set Y that is a candidate set for the centers of balls of radius r'_1 . Furthermore, let us assume that \mathcal{J} is feasible. First, the reduction in Section 3 (Theorem 11), given the instance \mathcal{J} , produces $O(n)$ instances $\mathcal{J}(\ell)$ of Colorful 2-NUkC, such that at least one of the instances is feasible. Then, we apply the reduction from Section 4 (Theorem 17) on each of these instances to ensure the *self-coverage* property, i.e., we obtain an instance $\mathcal{J}'(\ell)$ of Colorful 2-NUkC with $r''_1 = r'_1 + c_2 r'_2 + c_3 r'_3$ and, crucially, $r''_2 = 0$. Finally, assuming that the resulting instance $\mathcal{J}'(\ell)$ is feasible, it is possible to find a feasible solution using straightforward dynamic programming, as described in Section 5. This dynamic programming only requires that the instance is *Well-Separated* w.r.t. a smaller separation factor of 2. We argue in the next paragraph that this property holds in each of the instances $\mathcal{J}'(\ell)$.

In the reductions in Sections 3 and Section 4, the solution transformations preserve the centers for the balls of the largest radius class. In order to show that the set Y of candidate centers is well-separated w.r.t. the new top level radius r''_1 , we need to show that $3r'_1 \geq 2r''_1$, i.e., $r'_1 \geq 2c_2 r'_2 + 2c_3 r'_3 \geq \beta \cdot r'_2$ for some sufficiently large constant β . This assumption is without loss of generality, since, if two consecutive radii classes in instance \mathcal{J} are within a β factor, it is possible to combine them into a single radius class, at the expense of an $O(\beta)$ factor in the approximation guarantee.

Assuming the instance \mathcal{J} is feasible, a feasible solution to an instance $\mathcal{J}'(\ell)$ can be mapped back to an $O(1)$ -approximate solution to \mathcal{J} , and then to \mathcal{I} , since each reduction preserves the approximation guarantee up to an $O(1)$ factor.

► **Theorem 5.** *There exists a polynomial time $O(1)$ -approximation algorithm for 4-NUkC.*

We have overviewed how the various sections of the paper come together in deriving Theorem 5. Before proceeding to these sections, we describe a greedy clustering procedure that we need.

2.3 Greedy Clustering

Assume we are given (i) a metric space (X, d) , where X is finite, (ii) a radius $r \geq 0$, (iii) an expansion parameter $\gamma \geq 1$, (iv) a subset $Y \subseteq X$ and a weight function $\omega : Y \rightarrow \mathbb{Z}^+$. The weight $\omega(y)$ can be thought of as the multiplicity of $y \in Y$, or how many points are co-located

28:6 Non-Uniform k -Center and Greedy Clustering

at y . We describe a greedy clustering procedure, from Charikar et al. [7], that is used to partition the point set Y into clusters, each of which is contained in a ball of radius γr . This clustering procedure, together with its properties, is a crucial ingredient of our approach.

■ **Algorithm 1** GREEDYCLUSTERING($Y, X, r \geq 0, \gamma \geq 1, \omega : Y \rightarrow \mathbb{Z}^+$).

We require that $Y \subseteq X$

- 1: Let $U \leftarrow Y, M \leftarrow \emptyset$
- 2: **while** $U \neq \emptyset$ **do**
- 3: $p = \arg \max_{q \in X} \omega(U \cap B(q, r))$
- 4: $C(p) := U \cap B(p, \gamma r); wt(p) := \omega(C(p))$
- 5: $U \leftarrow U \setminus C(p)$
- 6: $M \leftarrow M \cup \{p\}$ ▷ We refer to $C(p)$ as a cluster whose center is p
- 7: **end while**
- 8: **return** $(M, \{C(p)\}_{p \in M}, \{wt(p)\}_{p \in M})$

In line 3, we only consider $q \in X$ such that $U \cap B(q, r) \neq \emptyset$. Notice that it is possible that $\omega(U \cap B(q, r)) = 0$ if $\omega(y) = 0$ for each $y \in U$. Furthermore, notice that we do not require that $q \in U$ for it to be an eligible point in line 3.

We summarize some of the key properties of this algorithm in the following observations.

- **Observation 6. 1.** For any $p \in M$, $C(p) \subseteq B(p, \gamma r)$,
- 2. Point $y \in Y$ belongs to the cluster $C(p)$, such that p is the first among all $q \in M$ satisfying $d(y, q) \leq \gamma r$.
- 3. The sets $\{C(p)\}_{p \in M}$ partition Y , which implies that
- 4. $\sum_{p \in M} wt(p) = \omega(Y)$, where $\omega(Z) = \sum_{z \in Z} \omega(z)$ for any $Z \subseteq Y$.
- 5. If p_i and p_j are the points added to M in iterations $i \leq j$, then $wt(p_i) \geq wt(p_j)$.
- 6. For any two distinct $p, q \in M$, $d(p, q) > (\gamma - 1)r$.

Proof. The first five properties are immediate from the description of the algorithm. Now, we prove the sixth property. Suppose for contradiction that there exist $p, q \in M$ with $d(p, q) \leq (\gamma - 1)r$, and without loss of generality, p was added to M before q . Then, note that at the end of this iteration, $B(q, r) \cap U = \emptyset$. Therefore, q will subsequently never be a candidate for being added to M in line 3. ◀

A key property of this greedy clustering, established by Charikar et al. [7], is that for any $k \geq 1$ balls of radius r , the weight of the points in the first k clusters is at least as large as the weight of the points covered by the k balls.

► **Lemma 7.** Suppose that the parameter γ used in Algorithm 1 is at least 3. Let \mathcal{B} be any collection of $k \geq 1$ balls of radius r , each centered at a point in X . Let M' consist of the first k' points of M chosen by the algorithm, where $k' = \min\{k, |M|\}$. We have

$$\sum_{p \in M'} wt(p) = \omega \left(\bigcup_{p \in M'} C(p) \right) \geq \omega \left(Y \cap \bigcup_{B \in \mathcal{B}} B \right).$$

The equality follows from the definition of $wt(p)$ and the fact that the clusters partition Y , as stated in Observation 6.

3 From Robust t -NUkC to Colorful $(t - 1)$ -NUkC

Let $\mathcal{I} = ((X, d), (\omega, m), (k_1, k_2, \dots, k_t), (r_1, r_2, \dots, r_t))$ be an instance of Robust t -NUkC. The reduction to Colorful $(t - 1)$ -NUkC consists of two phases. In the first phase, we use Algorithm 1 to reduce the instance \mathcal{I} to an instance \mathcal{I}' focused on the cluster centers output by the greedy algorithm. A key property of this reduction is that we may set $r_t = 0$ in the instance \mathcal{I}' – each ball at level t is allowed to cover at most one point.

In the second phase, we transform \mathcal{I}' to $O(n)$ instances of Colorful $(t - 1)$ -NUkC. Assuming there exists a feasible solution for \mathcal{I}' , at least one of the instances \mathcal{I}'' of Colorful $(t - 1)$ -NUkC has a feasible solution, and any approximate solution to \mathcal{I}'' can be used to obtain an approximate solution to \mathcal{I}' (and thus to \mathcal{I}).

Phase 1. Let $\mathcal{I} = ((X, d), (\omega, m), (k_1, k_2, \dots, k_t), (r_1, r_2, \dots, r_t))$ be an instance of Robust t -NUkC. We call the algorithm $\text{GREEDYCLUSTERING}(X, X, r_t, 3, \omega)$, and obtain a set of points M with the corresponding clusters $C(p)$ for $p \in M$. The greedy algorithm also returns a weight $wt(p) = \omega(C(p))$ for each $p \in M$. Let us number the points of M as p_i , where i is the iteration in which p_i was added to the set M by $\text{GREEDYCLUSTERING}(X, X, r_t, 3, \omega)$. This gives an ordering $\sigma = \langle p_1, p_2, \dots, p_{|M|} \rangle$ of the points in M . Note that $wt(p_i) \geq wt(p_j)$ for $i \geq j$.

We define a weight function $\lambda : X \rightarrow \mathbb{Z}^+$. Let $\lambda(p) = wt(p)$ for $p \in M$ and $\lambda(p) = 0$ for $p \in X \setminus M$. Note that for $p \in M$, $\lambda(p) = wt(p) = \omega(C(p))$. Thus, for each $p \in M$, we are moving the weight from points in cluster $C(p)$ to the cluster center p . Clearly, $\omega(X) = \lambda(X)$.

The output of Phase 1 is the instance $\mathcal{I}' = ((X, d), (\lambda, m), (k_1, k_2, \dots, k_t), (r'_1, r'_2, \dots, r'_{t-1}, 0))$ of Robust t -NUkC, where $r'_i = r_i + 3r_t$. Note that in the instance \mathcal{I}' , we have $r'_t = 0$, whereas the other radii in \mathcal{I} have been increased by an additive factor of $3r_t$. The following claim relates instances \mathcal{I} and \mathcal{I}' .

► **Lemma 8.** (a) If instance \mathcal{I} has a feasible solution, then so does the instance \mathcal{I}' . (b) Given a solution $(\mathcal{B}'_i)_{i \in [t]}$ for \mathcal{I}' that uses at most k_i balls of radius $\alpha r'_i$ for every $i \in [t]$, we can obtain a solution $(\mathcal{B}_i)_{i \in [t]}$ for \mathcal{I} that uses at most k_i balls of radius at most $\alpha r'_i + 3r_t \leq \alpha r_i + (3\alpha + 3)r_t$ for $1 \leq i \leq t$.

Proof. We begin with part (b). For each ball in $B(p, r)$ that is part of the solution $(\mathcal{B}'_i)_{i \in [t]}$, we replace it with the ball $B(p, r + 3r_t)$ to obtain a solution $(\mathcal{B}_i)_{i \in [t]}$ for \mathcal{I} . That is, we expand each ball by an additive $3r_t$. If $B(p, r)$ covers $q \in M$, then $B(p, r + 3r_t)$ covers $C(q)$, and $\lambda(q) = \omega(C(q))$. Let $M' \subseteq M$ denote the points covered by $(\mathcal{B}'_i)_{i \in [t]}$. The weight of the points covered by $(\mathcal{B}_i)_{i \in [t]}$ is at least

$$\sum_{p \in M'} \omega(C(p)) = \sum_{p \in M'} \lambda(p) \geq m.$$

We now establish (a). Fix a feasible solution $(\mathcal{B}_i)_{i \in [t]}$ to \mathcal{I} that covers ω -weight at least m , where \mathcal{B}_i is a set of at most k_i balls of radius r_i , for $i \in [t]$. Let $M_1 \subseteq M$ be the set of points p such that some point in $C(p)$ is covered by a ball in $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_{t-1}$.

Now let $M_2 = M \setminus M_1$ be the set of points p , such that any point in $C(p)$ is either covered by a ball from \mathcal{B}_t , or is an outlier. Let $X_i := \bigcup_{p \in M_i} C(p)$ for $i = 1, 2$. Note that $X = X_1 \sqcup X_2$.

Note that in the sequence $\sigma = \langle p_1, p_2, \dots, p_{|M|} \rangle$, the points of M_1 and M_2 may appear in an interleaved fashion. Let $p_{i_1}, p_{i_2}, \dots, p_{i_{|M_2|}}$ be the subsequence restricted to the points in M_2 . In the following lemma, we argue that the first k_t points in this subsequence are sufficient to replace the balls in \mathcal{B}_t . Let $k' = \min\{|\mathcal{B}_t|, |M_2|\} \leq k_t$.

► **Lemma 9.** *There exists a subset $M_2^+ \subseteq M_2$ of size at most k' such that $\sum_{p \in M_2^+} wt(p) \geq \omega(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_t} \text{Ball})$.*

Proof. Let $M_2^+ = \{p_{i_1}, p_{i_2}, \dots, p_{i_{k'}}\}$. That is, M_2^+ consists of the first k' points of M_2 picked by the greedy algorithm. Recall that $M_2^+ \subseteq M_2$, and thus for $p_{i_j} \in M_2^+$, it holds that $C(p_{i_j}) \subseteq X_2$.

Now imagine calling the algorithm $\text{GREEDYCLUSTERING}(X_2, X, r_t, 3, \omega)$. Observe that in the iteration $1 \leq j \leq |M_2|$, this algorithm will select point p_{i_j} (as defined above) in Line 3, and the corresponding cluster and its weight will be $C(p_{i_j})$ and $wt(p_{i_j})$ – exactly as in the execution of $\text{GREEDYCLUSTERING}(X, X, r_t, 3, \omega)$. That is, the algorithm $\text{GREEDYCLUSTERING}(X_2, X, r_t, 3, \omega)$ will output M_2 and the clusters $C(p)$ for each $p \in M_2$.

Now, \mathcal{B}_t consists of a set of $|\mathcal{B}_t|$ balls of radius r_t . The lemma now follows from Lemma 7 applied to $\text{GREEDYCLUSTERING}(X_2, X, r_t, 3, \omega)$. ◀

Using Lemma 9, we now construct a solution to instance \mathcal{I}' . Fix index $1 \leq i \leq t-1$, and \mathcal{B}'_i denote the set of balls obtained by expanding each ball in \mathcal{B}_i by an additive $3r_t$. Note that each ball in \mathcal{B}'_i has radius $r'_i = r_i + 3r_t$. For every point $p \in M_2^+$, we add a ball of radius 0 around it and let \mathcal{B}'_t be the resulting set of balls. Note that $|\mathcal{B}'_t| = |M_2^+| \leq k' \leq k_t$.

By definition, for each point $p \in M_1$, there is a ball in $(\mathcal{B}_i)_{i \in [t-1]}$ that intersects cluster $C(p)$, whose points are at distance at most $3r_t$ from p . It follows that the balls in $(\mathcal{B}'_i)_{i \in [t-1]}$ cover each point in M_1 .

Using Lemma 9, the coverage of $(\mathcal{B}'_i)_{i \in [t]}$ in instance \mathcal{I}' is at least

$$\sum_{p \in M_1} wt(p) + \sum_{p \in M_2^+} wt(p) \geq \omega(X_1) + \omega\left(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_t} \text{Ball}\right) \geq m.$$

The final inequality follows because any point covered by solution $(\mathcal{B}_i)_{i \in [t]}$ for \mathcal{I} either belongs to X_1 or to $X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_t} \text{Ball}$. Thus, we have shown that \mathcal{I}' has a feasible solution. ◀

Phase 2. Now we describe the second phase of the algorithm. We have the instance $\mathcal{I}' = ((X, d), (\lambda, m), (k_1, k_2, \dots, k_t), (r'_1, r'_2, \dots, r'_{t-1}, 0))$ of Robust t -NUkC that is output by Phase 1. Phase 2 takes \mathcal{I}' as input and generates an instance $\mathcal{I}(\ell)$, for each $0 \leq \ell \leq |X|$, of the Colorful $(t-1)$ -NUkC problem. Note that the number of generated instances is $|X| + 1 = O(n)$. If \mathcal{I}' is feasible, at least one of these $|X| + 1$ instances will be feasible.

Let $\sigma = \langle p_1, p_2, \dots, p_{|X|} \rangle$ be an ordering of the points in X by non-increasing λ . That is, $\lambda(p_i) \geq \lambda(p_j)$ for $i \leq j$.

Fix an index $0 \leq \ell \leq |X|$. We now describe the instance $\mathcal{I}(\ell)$ of colorful $(t-1)$ -NUkC. Let $R = \{p_1, p_2, \dots, p_\ell\}$ denote the set of *red* points, and $B = \{p_{\ell+1}, p_{\ell+2}, \dots, p_{|X|}\}$ denote the set of *blue* points. For each $p \in B$, define its blue weight as $\omega_b(p) := \lambda(p)$; for each $p \in R$, define its blue weight as $\omega_b(p) := 0$. Define the blue coverage m_b for instance $\mathcal{I}(\ell)$ as $m_b := m - \lambda(R)$. We define the red weight function ω_r in a slightly different manner. For each red point $p \in R$, let its red weight $\omega_r(p) := 1$; for each $p \in B$, let red weight $\omega_r(p) := 0$. Let $m_r := \sum_{p \in R} \omega_r(p) - k_t = |R| - k_t$ denote the red coverage for instance $\mathcal{I}(\ell)$. Note that ω_r is supported on R and ω_b on B . Let $\mathcal{I}(\ell) := ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, k_2, \dots, k_{t-1}), (r'_1, r'_2, \dots, r'_{t-1}))$ denote the resulting instance of Colorful $(t-1)$ -NUkC problem. Recall that a solution to this instance is required to cover red weight that adds up to at least m_r , and blue weight that adds up to at least m_b . (In instance $\mathcal{I}(\ell)$, the point sets R and B , the red and blue weights, and total coverage requirements m_r and m_b all depend on the index ℓ . This dependence is not made explicit in the notation, so as to keep it simple.)

We now relate the instance \mathcal{I}' to the instances $\mathcal{I}(\ell)$, for $0 \leq \ell \leq |X|$.

► **Lemma 10.** (a) *If the instance $\mathcal{I}' = ((X, d), (\lambda, m), (k_1, k_2, \dots, k_t), (r'_1, r'_2, \dots, r'_{t-1}, 0))$ is feasible, then there exists an $0 \leq \ell^* \leq |X|$ such that instance $\mathcal{I}(\ell^*)$ is feasible.*

(b) *Let $\mathcal{I}(\ell) = ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, k_2, \dots, k_{t-1}), (r'_1, r'_2, \dots, r'_{t-1}))$ be a generated instance of Colorful $(t-1)$ -NUkC, and suppose $(\mathcal{B}''_i)_{i \in [t-1]}$ is a solution to this instance such that \mathcal{B}''_i contains at most k_i balls of radius $\alpha r'_i$ for $1 \leq i \leq t-1$, and covers red weight at least m_r and blue weight at least m_b . Then, we can efficiently obtain a solution to the instance \mathcal{I}' that uses at most k_i balls of radius $\alpha r'_i$ for $1 \leq i \leq t-1$, and at most k_t balls of radius 0.*

Proof. We first show part (b). In instance $\mathcal{I}(\ell)$, the red weight $\omega_r(p) = 1$ for each $p \in R$, so the solution $(\mathcal{B}''_i)_{i \in [t-1]}$ covers at least $m_r = \sum_{p \in R} \omega_r(p) - k_t = |R| - k_t$ red points. So the number of red points that are not covered is at most k_t . Construct \mathcal{B}'_t by adding a ball of radius 0 at each uncovered point in R . Thus, $|\mathcal{B}'_t| \leq k_t$.

Let $\mathcal{B}'_i = \mathcal{B}''_i$ for each $1 \leq i \leq t-1$. Now, we argue that the solution $(\mathcal{B}'_i)_{i \in [t]}$ covers weight at least m in instance \mathcal{I}' . Note that this solution covers all points in R , and a subset $C \subseteq B$ such that $\omega_b(C) \geq m_b = m - \lambda(R)$. Thus the coverage for \mathcal{I}' is at least

$$\lambda(R) + \lambda(C) = \lambda(R) + \omega_b(C) \geq \lambda(R) + m_b = m.$$

We now turn to part (a). Fix a feasible solution $(\mathcal{B}'_i)_{i \in [t]}$ to \mathcal{I}' . Let $M_1 \subseteq X$ denote the subset consisting of each point covered by a ball in \mathcal{B}'_i , for $1 \leq i \leq t-1$. Let $M_2 = X \setminus M_1$. Each point in M_2 is either an outlier or is covered by a ball in \mathcal{B}'_t . Note that in the sequence $\sigma = \langle p_1, p_2, \dots, p_{|X|} \rangle$, the points of M_1 and M_2 may appear in an interleaved fashion. Let $p_{i_1}, p_{i_2}, \dots, p_{i_{|M_2|}}$ be the subsequence restricted to the points in M_2 . Let $k' = \min\{k_t, |M_2|\}$, and let $M_2^+ = \{p_{i_1}, p_{i_2}, \dots, p_{i_{k'}}\}$. A key observation is that $\lambda(M_2^+)$ is at least as large as the total weight of the points in M_2 covered by balls in \mathcal{B}'_t . This is because each ball in \mathcal{B}'_t has radius 0 and can cover only one point in M_2 ; and the maximum coverage using such balls is obtained by placing them at the points in M_2 with the highest weights, i.e., M_2^+ . Without loss of generality, we assume that \mathcal{B}'_t consists of balls of radius 0 placed at each point in M_2^+ .

Now, let the index $\ell^* := i_{k'}$. We now argue that the instance $\mathcal{I}(\ell^*)$ of colorful $(t-1)$ -NUkC is feasible. In particular, we argue that $(\mathcal{B}'_i)_{i \in [t-1]}$ is a solution. Consider the set $R = \{p_1, p_2, \dots, p_{\ell^*}\}$ of red points in $\mathcal{I}(\ell^*)$. Each point in R is either in M_1 or in M_2^+ , and is therefore covered by $(\mathcal{B}'_i)_{i \in [t]}$. It follows that $(\mathcal{B}'_i)_{i \in [t-1]}$ covers at least $|R| - |\mathcal{B}'_t| \geq |R| - |k_t| = m_r$ points of R . In other words, the red weight in $\mathcal{I}(\ell^*)$ covered by $(\mathcal{B}'_i)_{i \in [t-1]}$ is at least m_r .

Now consider the set $B = \{p_{\ell^*+1}, p_{\ell^*+2}, \dots, p_{|X|}\}$ of blue points in $\mathcal{I}(\ell^*)$. Let $C \subseteq B$ denote the blue points covered by solution $(\mathcal{B}'_i)_{i \in [t]}$. As $(\mathcal{B}'_i)_{i \in [t]}$ covers points with weight at least m in instance \mathcal{I}' , we have $\lambda(R) + \lambda(C) \geq m$; thus, $\lambda(C) \geq m - \lambda(R) = m_b$. However, the balls in \mathcal{B}'_t do not cover any point in B . Thus, the balls in $(\mathcal{B}'_i)_{i \in [t-1]}$ cover all points in C . For any $p \in B$, we have $\lambda(p) = \omega_b(p)$. It follows that the blue weight in $\mathcal{I}(\ell^*)$ covered by $(\mathcal{B}'_i)_{i \in [t-1]}$ is at least $\omega_b(C) = \lambda(C) \geq m_b$. This concludes the proof of part (a). ◀

Combining Lemmas 8 and 10 from Phases 1 and 2, we obtain the following reduction from robust t -NUkC to colorful $(t-1)$ -NUkC.

► **Theorem 11.** *There is a polynomial-time algorithm that, given an instance $\mathcal{I} = ((X, d), (\omega, m), (k_1, \dots, k_t), (r_1, \dots, r_t))$ of Robust t -NUkC, outputs a collection of $O(n)$ instances of Colorful $(t-1)$ -NUkC with the following properties: (a) If \mathcal{I} is feasible, then at least one of the instances $\mathcal{I}(\ell) = ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, \dots, k_{t-1}), (r'_1, \dots, r'_{t-1}))$ of Colorful $(t-1)$ -NUkC is feasible; (b) given an α -approximate solution to some instance $\mathcal{I}(\ell)$, we can efficiently construct a solution to \mathcal{I} that uses at most k_i balls of radius at most $\alpha r_i + (3\alpha + 3)r_t$.*

► **Remark 12.** In part (a), the feasible solution for $\mathcal{I}(\ell)$ that is constructed from the feasible solution for \mathcal{I} has the following useful property: for any **Ball** of radius $r'_i = r_i + 3r_t$ in the feasible solution for $\mathcal{I}(\ell)$, the center of **Ball** is also the center of some ball of radius r_i in the feasible solution for \mathcal{I} .

4 Ensuring Self-Coverage in Colorful 2-NU k C

We assume that we are given as input a Colorful 2-NU k C instance $\mathcal{I} = ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, k_2), (r_1, r_2))$. Recall that $\omega_r : X \rightarrow \mathbb{Z}^+$ (resp. $\omega_b : X \rightarrow \mathbb{Z}^+$) is the red (resp. blue) weight function. The task in Colorful 2-NU k C is to find a solution $(\mathcal{B}_1, \mathcal{B}_2)$ such that (1) $|\mathcal{B}_i| \leq k_i$ for $i = 1, 2$, and (2) the point set $Y \subseteq X$ covered by the solution satisfies $\omega_r(Y) \geq m_r$ and $\omega_b(Y) \geq m_b$, (i.e., the solution covers points with total red weight at least m_r , and blue weight at least m_b .) In this section, we show that \mathcal{I} can be reduced to an instance of Colorful 2-NU k C with $r_2 = 0$. The fact that each ball of radius r_2 can only cover its center in the target instance is what we mean by the term *self-coverage*. This reduction actually generalizes to Colorful t -NU k C, but we address the case $t = 2$ to keep the notation simpler.

Our reduction proceeds in two phases. In Phase 1, we construct an intermediate instance where we can ensure blue self-coverage. Then in Phase 2, we modify the intermediate instance so as to obtain red self-coverage as well.

Phase 1. In this step, we call the greedy clustering algorithm using the *blue weight function* ω_b . In particular, we call `GREEDYCLUSTERING`($X, X, r_2, 3, \omega_b$) (See Algorithm 1). This algorithm returns a set of points $M \subseteq X$, where every $p \in M$ has a cluster $C(p)$ and weight $wt(p)$ such that (1) $\{C(p)\}_{p \in M}$ is a partition of X ; (2) for any $p \in M$, $wt(p) = \omega_b(C(p))$, the blue weight of the cluster, and (3) $d(q, p) \leq 3r_2$ for any $q \in C(p)$. Furthermore, the greedy algorithm naturally defines an ordering $\sigma = \langle p_1, p_2, \dots, p_{|M|} \rangle$ of M – this is the order in which the points were added to M .

We define a new weight function $\lambda_b : X \rightarrow \mathbb{Z}^+$ as follows: $\lambda_b(p) := wt(p)$ if $p \in M$ and $\lambda_b(p) := 0$ if $p \in X \setminus M$. Note that for $p \in M$, we have $wt(p) = \omega_b(C(p))$. So the new weight function λ_b is obtained from ω_b by moving weight from each cluster $C(p)$ to its center p .

Phase 1 outputs the intermediate instance $\mathcal{I}' = ((X, d), (\omega_r, \lambda_b, m_r, m_b), (k_1, k_2), (r'_1, r'_2))$ of Colorful 2-NU k C, where $r'_1 = r_1 + 6r_2$ and $r'_2 = 5r_2$. A solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ for \mathcal{I}' is said to be *structured* if it has the following properties.

1. It is a solution to \mathcal{I}' viewed as an instance of Colorful 2-NU k C.
2. Let $Y \subseteq X$, the set of points *self-covered* by solution $(\mathcal{B}'_1, \mathcal{B}'_2)$, consist of points $p \in X$ such that either (a) p is covered by \mathcal{B}'_1 , or (b) p is the center of some ball in \mathcal{B}'_2 . We require that $\lambda_b(Y) \geq m_b$.

Thus, a structured solution covers red weight in the usual way; whereas a ball in \mathcal{B}'_2 can only contribute blue coverage for its center. The following lemma relates instances \mathcal{I} and \mathcal{I}' .

► **Lemma 13.** (a) *If instance \mathcal{I} has a feasible solution, then the instance \mathcal{I}' has a feasible solution that is also structured.* (b) *Given a solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ for \mathcal{I}' that uses at most k_i balls of radius $\alpha r'_i$ for every $i \in \{1, 2\}$, we can obtain a solution $(\mathcal{B}_1, \mathcal{B}_2)$ for \mathcal{I} that uses at most k_i balls of radius $\alpha r'_i + 3r_2 \leq \alpha r_i + (6\alpha + 3)r_2$ for $i \in \{1, 2\}$.*

Part (b) is straightforward as the red weights are unchanged in going from \mathcal{I} to \mathcal{I}' , and the blue weights are moved by at most $3r_2$. (Note that we don't require in part (b) that the solution to \mathcal{I}' be structured.)

In the rest of this section, we establish (a). Fix a feasible solution $(\mathcal{B}_1, \mathcal{B}_2)$ to \mathcal{I} . Thus, (1) $|\mathcal{B}_i| \leq k_i$ for $i = 1, 2$, and (2) the point set $Y \subseteq X$ covered by the solution satisfies $\omega_r(Y) \geq m_r$ and $\omega_b(Y) \geq m_b$, (i.e., the solution covers points with total red weight at least m_r , and blue weight at least m_b .)

Let $M_1 \subseteq M$ be the set of points p such that some point in $C(p)$ is covered by a ball in \mathcal{B}_1 . Now let $M_2 = M \setminus M_1$ be the set of points p such that any point in $C(p)$ is either covered by a ball from \mathcal{B}_2 , or is an outlier. Let $X_i := \bigcup_{p \in M_i} C(p)$ for $i = 1, 2$. Note that $X = X_1 \sqcup X_2$.

We construct a solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ for instance \mathcal{I}' as follows. The set \mathcal{B}'_1 is obtained by expanding each ball in \mathcal{B}_1 by an additive factor of $6r_2$. Thus, the balls in \mathcal{B}'_1 cover X_1 . As in the proof of Lemma 8, we construct a subset $N \subseteq M_2$ of size at most $|\mathcal{B}_2|$. We let \mathcal{B}'_2 consist of the balls of radius $r'_2 = 5r_2$, each centered at a point in N . The set N will have the following properties:

$$\omega_r(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}'_2} \text{Ball}) \geq \omega_r(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_2} \text{Ball}) \quad (1)$$

$$\sum_{p \in N} wt(p) \geq \omega_b(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_2} \text{Ball}) \quad (2)$$

It is easy to verify that these two guarantees imply that $(\mathcal{B}'_1, \mathcal{B}'_2)$ is a structured, feasible solution to \mathcal{I}' :

The red weight covered by $(\mathcal{B}'_1, \mathcal{B}'_2)$ is at least

$$\omega_r(X_1) + \omega_r(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}'_2} \text{Ball}) \geq \omega_r(X_1) + \omega_r(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_2} \text{Ball}) \geq m_r.$$

The set $M_1 \cup N$ is self-covered by $(\mathcal{B}'_1, \mathcal{B}'_2)$. We have

$$\lambda_b(M_1) + \lambda_b(N) = \omega_b(X_1) + \sum_{p \in N} wt(p) \geq \omega_b(X_1) + \omega_b(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_2} \text{Ball}) \geq m_b.$$

We now describe the construction of N and establish properties (1) and (2). At a high level, this is similar to what we did for M_2^+ in Lemma 8; but it is more involved as we need to ensure that both properties hold.

Let $\widehat{\mathcal{B}}_2 = \{\text{Ball} \in \mathcal{B}_2 \mid \text{Ball} \cap X_2 \neq \emptyset\}$. The set N is obtained via MAPPING PROCEDURE, given in Algorithm 2. In particular, we invoke MAPPING PROCEDURE($M_2, \sigma, \widehat{\mathcal{B}}_2, \{C(p)\}_{p \in M_2}$). We describe Algorithm 2 at a high level. First, we map every ball in $\widehat{\mathcal{B}}_2$ to the *first* (according to σ) point q in M_2 whose cluster $C(q)$ has a non-empty intersection with the ball – this is the definition of φ . Now, some points $q \in M_2$ may get mapped by more than one ball. Then, we create a “grouping procedure” that creates pairs (N_ℓ, D_ℓ) as follows. We start from the *first* (according to σ) point q_i that is mapped by at least one ball. We add q_i to N_ℓ , and the balls that were mapped to q_i to the set D_ℓ . Now, if $|\varphi^{-1}(q_i)| > 1$, then we aim to find $|\varphi^{-1}(q_i)| - 1$ additional points after q_i to be added to N_ℓ . Furthermore, it is important in the analysis that these points be *consecutive* according to $\sigma|_{M_2}$. The variable **pending** keeps track of how many additional distinct points need to be added to N_ℓ to match the number of distinct balls in D_ℓ at the current time. Thus, if $|\varphi^{-1}(q_i)| > 1$, we add q_{i+1} to N_ℓ as well. At this stage, it may happen that $\varphi^{-1}(q_{i+1}) \neq \emptyset$. Then, we add $\varphi^{-1}(q_{i+1})$ to D_ℓ , and update the variable **pending** appropriately. If the variable **pending** becomes 0, then $|N_\ell| = |D_\ell|$, at which point the inner while loop terminates. By construction, the points added to N_ℓ form a contiguous sub-sequence of $\sigma|_{M_2}$. We add the pair (N_ℓ, D_ℓ) to \mathcal{T} . At this point, if there still exists a ball of $\widehat{\mathcal{B}}_2$ that does not belong to any D_j with $j \leq \ell$, we

28:12 Non-Uniform k -Center and Greedy Clustering

■ **Algorithm 2** MAPPING PROCEDURE($\widehat{M}, \sigma, \widehat{\mathcal{B}}, \{C(p)\}_{p \in \widehat{M}}$).

```

1: Index the points of  $\widehat{M}$  as  $q_1, q_2, \dots$  according to the ordering  $\sigma$ 
2: For every  $\text{Ball} \in \widehat{\mathcal{B}}, \varphi(\text{Ball}) := q_i$ , where  $q_i \in \widehat{M}$  is the first point  $q$  s.t.  $\text{Ball} \cap C(q) \neq \emptyset$ 
3:  $\ell = 0; \mathcal{T} \leftarrow \emptyset$ 
4: while there exists a  $\text{Ball} \in \widehat{\mathcal{B}}$  that does not belong to any  $D_j$  with  $j \leq \ell$  do
5:    $\ell \leftarrow \ell + 1$ 
6:    $q_i \in \widehat{M} \setminus \bigcup_{j=1}^{\ell-1} N_j$  be the first point  $q$  with  $|\varphi^{-1}(q)| > 0$ 
7:   pending  $\leftarrow |\varphi^{-1}(q_i)| - 1$ 
8:    $N_\ell \leftarrow \{q_i\}, D_\ell \leftarrow \varphi^{-1}(q_i)$ 
9:   while pending  $> 0$  and  $i + 1 \leq |\widehat{M}|$  do
10:     $i \leftarrow i + 1$ 
11:    pending  $\leftarrow$  pending  $+$   $|\varphi^{-1}(q_i)| - 1$ 
12:     $N_\ell \leftarrow N_\ell \cup \{q_i\}, D_\ell \leftarrow D_\ell \cup \varphi^{-1}(q_i)$ 
13:   end while
14:   Add  $(N_\ell, D_\ell)$  to  $\mathcal{T}$ 
15: end while
16: Return  $\mathcal{T}$ 

```

start the construction of the next pair $(N_{\ell+1}, D_{\ell+1})$. Note that in all but the last iteration of the outer while loop, it holds that $|N_\ell| = |D_\ell|$. However, in the last iteration t , the loop may terminate with $|N_t| \leq |D_t|$.

The invocation of MAPPING PROCEDURE($M_2, \sigma, \widehat{\mathcal{B}}_2, \{C(p)\}_{p \in M_2}$) returns $\mathcal{T} = \{(N_1, D_1), (N_2, D_2), \dots, (N_t, D_t)\}$. In the following observation, we summarize a few key properties of this collection of pairs.

► **Observation 14.** $\mathcal{T} = \{(N_1, D_1), (N_2, D_2), \dots, (N_t, D_t)\}$ satisfies the following properties.

1. For each $1 \leq \ell \leq t$, we have $\emptyset \neq N_\ell \subseteq M_2$; Furthermore, the points of N_ℓ form a contiguous subsequence of M_2 ordered according to σ . The sets N_1, N_2, \dots, N_t are pairwise disjoint.
2. For each $1 \leq \ell \leq t$, we have $\emptyset \neq D_\ell \subseteq \widehat{\mathcal{B}}_2$. The sets D_1, \dots, D_t form a partition of $\widehat{\mathcal{B}}_2$.
3. $|N_\ell| = |D_\ell|$ for $\ell < t$, and $|N_t| \leq |D_t|$.

Now we prove the following key lemma.

► **Lemma 15.** For any $1 \leq \ell \leq t$, the following properties hold.

A. For any ball $B(c, r_2) \in D_\ell$, there exists a $q \in N_\ell$ such that $B(c, r_2) \subseteq B(q, 5r_2)$.

B. $\omega_b \left(X_2 \cap \bigcup_{B(c, r_2) \in D_\ell} B(c, r_2) \right) \leq \sum_{p \in N_\ell} wt(p)$.

Proof. For any $\text{Ball} = B(c, r_2) \in D_\ell$, $q_i = \varphi(\text{Ball}) \in N_\ell$. By the definition of q_i , it holds that $C(q_i) \cap \text{Ball} \neq \emptyset$. Therefore, for any point $p \in \text{Ball}$, it holds that $d(p, q_i) \leq d(p, c) + d(c, p') + d(p', q_i) \leq r_2 + r_2 + 3r_2 = 5r_2$, where $p' \in C(q_i) \cap \text{Ball}$. This proves property A.

Let $\mathcal{X}_\ell := X_2 \cap \left(\left(\bigcup_{q \in N_\ell} C(q) \right) \cup \left(\bigcup_{\text{Ball} \in D_\ell} \text{Ball} \right) \right)$. That is, \mathcal{X}_ℓ denotes the set of those points in X_2 that belong to the clusters of all the points in N_ℓ , as well as those in the balls in D_ℓ . Now, imagine calling GREEDYCLUSTERING($\mathcal{X}_\ell, X, r_2, 3, \omega_b$). As in the proof of Lemma 9, the main observation is that the set of clusters computed in the first $|N_\ell|$ iterations is exactly $\{C(q)\}_{q \in N_\ell}$. Thus, property B in the lemma follows from Lemma 7 applied to GREEDYCLUSTERING($\mathcal{X}_\ell, X, r_2, 3, \omega_b$). ◀

We now set $N = \bigcup_{1 \leq \ell \leq t} N_\ell$. Note that

$$|N| = \sum_{\ell} |N_\ell| \leq \sum_{\ell} |D_\ell| = |\widehat{\mathcal{B}}_2| \leq |\mathcal{B}_2|.$$

Recall that for instance \mathcal{I}' , we set $\mathcal{B}'_2 = \{B(q, 5r_2) \mid q \in N\}$. We now argue that N satisfies properties (1) and (2).

By Property A of Lemma 15, we have that for any $\text{Ball} \in \mathcal{B}_2$, there is a $\text{Ball}' \in \mathcal{B}'_2$ such that $X_2 \cap \text{Ball} \subseteq X_2 \cap \text{Ball}'$. Thus, $(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_2} \text{Ball}) \subseteq (X_2 \cap \bigcup_{\text{Ball}' \in \mathcal{B}'_2} \text{Ball}')$, which implies property (1).

Using Property B of Lemma 15, we can prove property (1) as follows:

$$\sum_{p \in N} wt(p) = \sum_{\ell} \sum_{p \in N_\ell} wt(p) \geq \sum_{\ell} \omega_b \left(X_2 \cap \bigcup_{\text{Ball} \in D_\ell} \text{Ball} \right) \geq \omega_b \left(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}_2} \text{Ball} \right),$$

Phase 2. Phase 1 outputs an instance $\mathcal{I}' = ((X, d), (\lambda_r, \lambda_b, m_r, m_b), (k_1, k_2), (r'_1, r'_2))$ of Colorful 2-NUkC. In Phase 2, we transform this into an instance $\mathcal{I}'' = ((X, d), (\chi_r, \chi_b, m_r, m_b), (k_1, k_2), (r''_1, 0))$ of Colorful 2-NUkC where the radius at the second level is 0.

In this step, we call the greedy clustering algorithm (Algorithm 1) using the *red weight function* λ_r . In particular, we will call $\text{GREEDYCLUSTERING}(X, X, r'_2, 3, \lambda_r)$. This algorithm returns a set of points $M \subseteq X$, where every $p \in M$ has a cluster $C(p)$ and weight $wt(p)$ such that (1) $\{C(p)\}_{p \in M}$ is a partition of X , (2) For any $p \in M$, $wt(p) = \lambda_r(C(p))$, the red weight of the cluster, and (3) $d(q, p) \leq 3r'_2$ for any $q \in C(p)$. Furthermore, the greedy algorithm naturally defines an ordering $\sigma = \langle p_1, p_2, \dots, p_{|M|} \rangle$ of M – this is the order in which the points were added to M .

We define the red weight function χ_r for \mathcal{I}'' as follows: $\chi_r(p) := \lambda_r(C(p))$ for $p \in M$, and $\chi_r(p) := 0$ for $p \in X \setminus M$. We also define $\phi : X \rightarrow M$ as follows: $\phi(p)$ is the first point in M (according to σ) such that $B(p, r'_2) \cap C(p) \neq \emptyset$. Note that $\phi(p)$ exists and $d(p, \phi(p)) \leq 4r'_2$. We define the blue weight function χ_b for \mathcal{I}'' as follows: $\chi_b(p) := \sum_{q \in \phi^{-1}(p)} \lambda_b(q)$ for $p \in M$, and $\chi_b(p) := 0$ for $p \in X \setminus M$.

Finally, we let $r''_1 = r'_1 + 4r'_2$, and obtain the instance $\mathcal{I}'' = ((X, d), (\chi_r, \chi_b, m_r, m_b), (k_1, k_2), (r''_1, 0))$ of Colorful 2-NUkC. The following lemma relates instances \mathcal{I}' and \mathcal{I}'' .

► **Lemma 16.** (a) *If instance \mathcal{I}' has a feasible solution that is structured, then the instance \mathcal{I}'' has a feasible solution.* (b) *Given a solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ for \mathcal{I}' that uses at most k_i balls of radius $\alpha r'_i$ for each $i \in \{1, 2\}$, we can obtain a solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ for \mathcal{I}' that uses at most k_i balls of radius $\alpha r''_i + 4r'_2 \leq \alpha r'_i + (4\alpha + 4)r'_2$ for $i \in \{1, 2\}$.*

Again, part (b) follows from the fact that in constructing \mathcal{I}'' from \mathcal{I}' , we move weights by a distance of at most $4r'_2$. Note that we do not claim that the solution to \mathcal{I}' constructed in part (b) is structured.

In the rest of this section, we establish part (a). Fix a feasible solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ for \mathcal{I}' that is also structured. Our construction of a feasible solution for \mathcal{I}'' is analogous to what we did in Phase 1.

Let $M_1 \subseteq M$ be the set of points p such that there exists some point x satisfying (i) x is covered by a ball in \mathcal{B}'_1 , and (ii) $d(x, p) \leq 4r'_1$. Note that M_1 includes any $p \in M$ such $C(p)$ contains a point covered by a ball in \mathcal{B}'_1 . Now let $M_2 = M \setminus M_1$; note that for $p \in M_2$, any point in $C(p)$ is either covered by a ball from \mathcal{B}'_2 , or is an outlier. Let $X_i := \bigcup_{p \in M_i} C(p)$ for $i = 1, 2$. Note that $X = X_1 \sqcup X_2$.

Let $\widehat{\mathcal{B}}'_2 = \{\text{Ball} \in \mathcal{B}'_2 \mid \text{Ball} \cap X_2 \neq \emptyset\}$. We invoke `MAPPING PROCEDURE`($M_2, \sigma, \widehat{\mathcal{B}}'_2, \{C(p)\}_{p \in M_2}$) and $\mathcal{T} = \{(N_1, D_1), (N_2, D_2), \dots, (N_t, D_t)\}$. We let $N = \bigcup_{1 \leq \ell \leq t} N_\ell$.

As in phase 1, we have that $|N| \leq |\widehat{\mathcal{B}}'_2| \leq |\mathcal{B}'_2|$. The set N also satisfies the following property, which is the analog of Property 2.

$$\sum_{p \in N} wt(p) \geq \lambda_r(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}'_2} \text{Ball}) \quad (3)$$

We now construct a solution $(\mathcal{B}''_1, \mathcal{B}''_2)$ for \mathcal{I}'' . The set \mathcal{B}''_1 is obtained by expanding each ball in \mathcal{B}'_1 by an additive $4r'_2$; each ball in \mathcal{B}''_1 has radius r''_1 . Note that by definition of M_1 , the balls in \mathcal{B}''_1 cover M_1 . The set \mathcal{B}''_2 is obtained by including in it a ball of radius 0 at each point in N . Note that $|\mathcal{B}''_2| = |N| \leq |\mathcal{B}'_2|$.

We now argue that $(\mathcal{B}''_1, \mathcal{B}''_2)$ provides adequate coverage. Red coverage is analogous to blue coverage in phase 1, using property 3:

$$\chi_r(M_1) + \chi_r(N) = \lambda_r(X_1) + \sum_{p \in N} wt(p) \geq \lambda_r(X_1) + \lambda_r(X_2 \cap \bigcup_{\text{Ball} \in \mathcal{B}'_2} \text{Ball}) \geq m_r.$$

For blue coverage, let $Y \subseteq X$ denote the set of points *self-covered* by the structured, feasible solution $(\mathcal{B}'_1, \mathcal{B}'_2)$ with $\lambda_b(Y) \geq m_b$. We argue that for each $y \in Y$, we have $\phi(y) \in M_1 \cup N$. If y is covered by a ball in \mathcal{B}'_1 , then as $d(y, \phi(y)) \leq 4r'_2$, we conclude that $\phi(y) \in M_1$ using the definition of M_1 . Otherwise, y is the center of some ball in $B(y, r'_2) \in \mathcal{B}'_2$. Assume $\phi(y) \notin M_1$. Then by the definition of ϕ , $\phi(y)$ is the first point $p \in M_2$ such that $B(y, r'_2)$ intersects $C(p)$. But this means $\phi(y)$ is the same as $\varphi(B(y, r'_2))$ computed in `MAPPING PROCEDURE`($M_2, \sigma, \widehat{\mathcal{B}}'_2, \{C(p)\}_{p \in M_2}$). Thus, $B(y, r'_2) \in D_\ell$ and $\phi(y) \in N_\ell$ for some pair (N_ℓ, D_ℓ) in \mathcal{T} . We conclude $\phi(y) \in N = \bigcup_\ell N_\ell$.

Thus, the blue coverage of $(\mathcal{B}''_1, \mathcal{B}''_2)$ is at least

$$\chi_b(M_1) + \chi_b(N) \geq \sum_{p \in M_1 \cup N} \phi^{-1}(p) \geq \sum_{y \in Y} \lambda_b(y) \geq m_b.$$

This completes the proof of Lemma 16 and concludes our description of Phase 2. Combining Phase 1 and Phase 2, we conclude with the main result of this section.

► **Theorem 17.** *There is a polynomial-time algorithm that transforms a Colorful 2-NUkC instance $\mathcal{I} = ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, k_2), (r_1, r_2))$ into an instance $\mathcal{I}'' = ((X, d), (\chi_r, \chi_b, m_r, m_b), (k_1, k_2), (r''_1, 0))$ of Colorful 2-NUkC with $r''_1 = r_1 + 26r_2$, and has the following properties: (a) If \mathcal{I} has a feasible solution, then so does \mathcal{I}'' ; (b) Given an α -approximate solution to \mathcal{I}'' , we can construct, in polynomial time, a $c \cdot \alpha$ -approximate solution to \mathcal{I} , where $c > 0$ is an absolute constant.*

► **Remark 18.** In part (a), the feasible solution $(\mathcal{B}''_1, \mathcal{B}''_2)$ to \mathcal{I}'' that is constructed from feasible solution $(\mathcal{B}_1, \mathcal{B}_2)$ to \mathcal{I} has the following useful property: for any $\text{Ball} \in \mathcal{B}''_1$, the center of Ball is also the center of some ball in \mathcal{B}'_1 .

5 Solving Well-Separated Colorful 2-NUkC

We assume that we are given a *well-separated* instance $\mathcal{I} = ((X, d), (\omega_r, \omega_b, m_r, m_b), (k_1, k_2), (r_1, 0))$ of Colorful 2-NUkC. The *well-separatedness* of the instance comes with the following additional input and restriction – we are given an additional set $Y \subseteq X$ as an input. The set Y is *well-separated*, i.e., for any $u, v \in Y$, $d(u, v) > 2r_1$. The additional restriction is that, the set of centers of balls of radius r_1 must be chosen from the set Y . We sketch how to solve such an instance optimally in polynomial time using dynamic programming.

Let $z := |Y|$, and let $Y = \{y_1, y_2, \dots, y_z\}$. For $1 \leq i \leq z$, let $X_i := B(y_i, r_1) \cap X$, and let $X_{z+1} := X \setminus \left(\bigcup_{1 \leq i \leq z} X_i\right)$. Note that $\{X_i\}_{1 \leq i \leq z+1}$ is a partition of X .

For any $X' \subseteq X$ and non-negative integers k, n_r, n_b , let $F(X', k, n_r, n_b)$ be **true** if there exists a subset $X'' \subseteq X'$ of size at most k , and (red, blue) weight at least (n_r, n_b) ; and **false** otherwise.⁴ For a particular subset X' , the value of $F(X', k, n_r, n_b)$ can be found in polynomial time using dynamic programming, since the values k, n_r, n_b are at most n .

For $(1, 0, 0, 0, 0) \leq (i, k'_1, k'_2, n_r, n_b) \leq (z+1, k_1, k_2, m_r, m_b)$, let $G(i, k'_1, k'_2, n_r, n_b)$ be **true** if it is possible to obtain (red, blue) coverage of at least (n_r, n_b) from the set of points $\bigcup_{1 \leq j \leq i} X_j$, using at most k'_1 balls of radius r_1 and k'_2 balls of radius 0; and **false** otherwise. Note that if $G(i-1, k'_1, k'_2, n_r, n_b) = \mathbf{true}$, then $G(i, k'_1, k'_2, n_r, n_b)$ is trivially **true**. Otherwise, suppose some points in X_i are covered. We consider two possibilities: either (A) X_i is covered using a ball of radius r_1 (note that for $i \leq z$ this is possible by definition; for $i = z+1$ we omit this case), and the remaining (red, blue) coverage comes from $\bigcup_{1 \leq j \leq i-1} X_j$, or (B) We use some $1 \leq t \leq \min\{k'_2, |X_i|\}$ balls of radius 0 to achieve the (red, blue) coverage of (n'_r, n'_b) from within X_i , and the remaining (red, blue) coverage comes from $\bigcup_{1 \leq j \leq i-1} X_j$. Note that in case (B), for a fixed guess of (t, n'_r, n'_b) , the subproblem for X_i corresponds to $F(X_i, t, n'_r, n'_b)$ as defined in the previous paragraph, and can be solved in polynomial time. It is straightforward to convert this recursive argument to compute $G(z+1, k_1, k_2, m_r, m_b)$ into a dynamic programming algorithm that also finds a feasible solution, and it can be implemented in polynomial time. We omit the details.

6 From $(t+1)$ -NUkC to Robust t -NUkC

In this section, we show an approximate equivalence of $t+1$ -NUkC and Robust t -NUkC. Note that Jia et al. [12] recently showed a very similar result. However, our proof is slightly different from theirs, and we describe it here for the sake of completeness.

► Lemma 19.

1. Suppose there exists an α -approximation algorithm for $(t+1)$ -NUkC. Then, there exists an α -approximation algorithm for unweighted Robust t -NUkC.
2. Suppose there exists a β -approximation algorithm for unweighted Robust t -NUkC. Then there exists a $3\beta + 2$ -approximation algorithm for $(t+1)$ -NUkC.

Proof. Note that the first claim is trivial, since an instance of Robust t -NUkC is a special case of NUkC, as follows. Let $\mathcal{I} = ((X, d), (\mathbb{1}, m), (r_1, r_2, \dots, r_t), (k_1, k_2, \dots, k_t))$ be an instance of unweighted t -Robust-NUkC, where m is the coverage requirement. Then, observe that it is equivalent to the instance $\mathcal{I}' = ((X, d), (r_1, r_2, \dots, r_t, 0), (k_1, k_2, \dots, k_t, n-m))$ of $t+1$ -NUkC. An α -approximate solution to \mathcal{I}' immediately gives an α -approximate solution to \mathcal{I} . We now proceed to the second claim.

Consider an instance $\mathcal{I} = ((X, d), (r_1, r_2, \dots, r_t, r_{t+1}), (k_1, k_2, \dots, k_t, k_{t+1}))$ of $(t+1)$ -NUkC. Note that we have to cover all points of X in the instance \mathcal{I} . First, we compute a $2r_{t+1}$ -net Y of X . That is compute $Y \subseteq X$ with the following properties: (i) $d(u, v) > 2r_{t+1}$ for any $u, v \in Y$, and (ii) for any $u \in X \setminus Y$, there exists a $v \in Y$ such that $d(u, v) \leq 2r_{t+1}$. Let $\varphi : X \rightarrow Y$ be a mapping that assigns every point in X to its nearest point in Y (breaking ties arbitrarily). Our reduction constructs the instance $\mathcal{I}' = ((Y, d), (\mathbb{1}, |Y| - k_{t+1}), (k_1, k_2, \dots, k_t), (r'_1, r'_2, \dots, r'_t))$ of t -Robust-NUkC with at most k_{t+1} outliers, where $r'_i = r_i + 2r_{t+1}$ for $1 \leq i \leq t$.

⁴ We use X'' has (red, blue) weight at least (n_r, n_b) as shorthand for $\omega_r(X'') \geq n_r$ and $\omega_b(X'') \geq n_b$.

We now argue that if \mathcal{I} is feasible, then so is \mathcal{I}' . Fix a solution $(\mathcal{B}_i)_{i \in [t+1]}$ for the original instance \mathcal{I} , where \mathcal{B}_i is a set of at most k_i balls of radius r_i . Let $Y' \subseteq Y$ be the set of points in Y covered by $(\mathcal{B}_i)_{i \in [t]}$, the balls of the t largest radii types. For each ball $B(c_i, r_i) \in \mathcal{B}_i$, we add $B(\varphi(c_i), r'_i)$ to obtain the set \mathcal{B}'_i of balls; recall $r'_i = r_i + 2r_t$. Note that the resulting solution $(\mathcal{B}'_i)_{i \in [t]}$ covers the set of points Y' . Now, let $Y'' = Y \setminus Y'$ be the set of points covered by \mathcal{B}_{t+1} , the balls of radius r_{t+1} . The distance between any two points of Y , and thus Y'' , is greater than $2r_{t+1}$. Therefore, a ball of radius r_{t+1} covers at most one point of Y'' , which implies that $|Y''| \leq |\mathcal{B}_{t+1}| \leq k_{t+1}$. Thus $(\mathcal{B}'_i)_{i \in [t]}$ is a feasible solution for instance \mathcal{I}' , with the points in Y'' being the set of outliers of size at most k_{t+1} .

We now argue that from a β -approximate solution to \mathcal{I}' , we can efficiently construct a $(3\beta + 2)$ -approximate solution to \mathcal{I} . Fix a solution $(\mathcal{B}'_i)_{i \in [t]}$ for the instance \mathcal{I}' that covers at least $|Y| - k_{t+1}$ points of Y , where \mathcal{B}'_i consists of k_i balls of radius $\beta r'_i$, for $1 \leq i \leq t$. To obtain a solution for the original instance \mathcal{I} , we proceed as follows. We expand the radius of every ball in \mathcal{B}'_i by an additive factor of $2r_{t+1}$ to obtain \mathcal{B}_i . Note that the resulting radius for each ball in \mathcal{B}_i is $\beta r_i + 2\beta r_{t+1} + 2r_{t+1} \leq (3\beta + 2) \cdot r_i$. Note that if a ball in solution $(\mathcal{B}'_i)_{i \in [t]}$ covers $y \in Y$, then the additively expanded version of the ball covers every point $x \in \varphi^{-1}(y)$. For every outlier point $y \in Y$ not covered by $(\mathcal{B}'_i)_{i \in [t]}$, we add a ball of radius $2r_{t+1}$ centered at y to \mathcal{B}_{t+1} ; this ball covers all points $x \in \varphi^{-1}(y)$. As the number of outliers is at most k_{t+1} , we have $|\mathcal{B}_{t+1}| \leq k_{t+1}$. The resulting solution $(\mathcal{B}_i)_{i \in [t+1]}$ covers all the points of X , and has approximation guarantee $3\beta + 2$. \blacktriangleleft

References

- 1 Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k -center with covering constraints. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*, volume 12125 of *Lecture Notes in Computer Science*, pages 52–65. Springer, 2020. doi:10.1007/978-3-030-45771-6_5.
- 2 Sayan Bandyapadhyay. On perturbation resilience of non-uniform k -center. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPICs*, pages 31:1–31:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.APPROX/RANDOM.2020.31.
- 3 Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi R. Varadarajan. A constant approximation for colorful k -center. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.12.
- 4 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k -center problem. *ACM Trans. Algorithms*, 16(4):46:1–46:19, 2020. doi:10.1145/3392720.
- 5 Deeparnab Chakrabarty and Maryam Negahbani. Generalized center problems with outliers. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 6 Deeparnab Chakrabarty and Maryam Negahbani. Robust k -center with two types of radii. In Mohit Singh and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2021. doi:10.1007/978-3-030-73879-2_19.
- 7 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 642–651. Society for Industrial and Applied Mathematics, 2001.

- 8 David G Harris, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. A lottery model for center-type problems with outliers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 9 Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.
- 10 Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.
- 11 Tanmay Inamdar and Kasturi Varadarajan. Capacitated sum-of-radii clustering: An fpt approximation. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 12 Xinrui Jia, Lars Rohwedder, Kshiteej Sheth, and Ola Svensson. Towards non-uniform k -center with constant types of radii. In Karl Bringmann and Timothy Chan, editors, *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, pages 228–237. SIAM, 2022. doi:10.1137/1.9781611977066.16.
- 13 Xinrui Jia, Kshiteej Sheth, and Ola Svensson. Fair colorful k -center clustering. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*, volume 12125 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2020. doi:10.1007/978-3-030-45771-6_17.

A Setup for Robust t -NU k C

Let $\mathcal{I} = ((X, d), (\mathbb{1}, m)(k_1, \dots, k_t), (r_1, \dots, r_t))$ be an instance of Robust t -NU k C. First we state the natural LP relaxation for \mathcal{I} . Recall that the goal is to cover at least m points.

$$\begin{aligned}
\sum_{v \in X} \text{cov}(v) &\geq m \\
\sum_{u \in X} x_{i,u} &\leq k_i && \forall 1 \leq i \leq t \\
\text{cov}_i(v) &= \sum_{u \in B(v, r_i)} x_{i,u} && \forall 1 \leq i \leq t, \forall v \in X \\
\text{cov}(v) &= \min \left\{ \sum_{i=1}^t \text{cov}_i(v), 1 \right\} && \forall v \in X \\
x_{i,u} &\geq 0 && \forall 1 \leq i \leq t, \forall u \in X.
\end{aligned}$$

Let \mathcal{F} denote the set of all tuples of subsets (S_1, \dots, S_t) , where $|S_i| \leq k_i$ for $1 \leq i \leq t$. For $v \in X$, and $1 \leq i \leq t$, we say that $(S_1, \dots, S_t) \in \mathcal{F}$ covers v with radius r_i , if $d(v, S_i) \leq r_i$. Let $\mathcal{F}_i(v) \subseteq \mathcal{F}$ denote the subset of solutions that cover v with radius r_i – where, the sets $\mathcal{F}_i(v)$ of solutions are assumed to be disjoint by including a solution in $\mathcal{F}_i(v)$ of the smallest index i , if it appears in multiple such sets.

If the instance \mathcal{I} is feasible, then the integer hull of the coverages, $\mathcal{P}_{\text{cov}}^{\mathcal{I}}$ as given below, must be non-empty.

$$\begin{aligned}
\mathcal{P}_{\text{cov}}^{\mathcal{I}} : \quad &\sum_{v \in X} \sum_{i \in [t]} \text{cov}_i(v) \geq m \\
&\sum_{S \in \mathcal{F}_i(v)} z_S = \text{cov}_i(v) && \forall i \in [t], \forall v \in X \\
&\sum_{S \in \mathcal{F}} z_S = 1 \\
&z_S \geq 0 && \forall S \in \mathcal{F}
\end{aligned}$$

Next, we give a few definitions from [6], generalized to arbitrary $t \geq 2$, for the sake of completeness. These definitions are used in the round-or-cut framework that reduces an instance of Robust t -NU k C to Well-Separated Robust t -NU k C, as described in Section B.

t -Firefighter Problem. The input is a collection of height- t trees, where L_1 is the set of roots, and for any $v \in L_i$ with $i \geq 1$, $a_j(v)$ represents the ancestor of v that belongs to L_j , where $1 \leq j \leq i$ ($a_i(v) = v$). Furthermore, let $w : L_t \rightarrow \mathbb{N}$ be a weight function on the leaves. For a root $u \in L_1$, we use $\text{Leaf}(u)$ to denote the set of leaves, i.e., nodes in L_t in the tree rooted at u . Note that the $\{\text{Leaf}(u) : u \in L_1\}$ partitions L_t . Thus, $((L_1, \dots, L_t), (a_1, a_2, \dots, a_t), \text{Leaf}, w)$ completely describes the structure of the tree, where $a_i(v) : \bigcup_{i \leq j \leq t} L_j \rightarrow L_i$ is an ancestor function as defined above. Now we define the t -FF problem.

► **Definition 20** (t -FF Problem). *Given height- t trees $(\mathcal{T} = (L_1, \dots, L_t), (a_1, \dots, a_t), \text{Leaf}, w)$, along with budgets (k_1, \dots, k_t) , we say that $T = (T_1, \dots, T_t)$, with $T_i \subseteq L_i$ is a feasible solution, if $|T_i| \leq k_i$ for $1 \leq i \leq t$. Let $\mathcal{C}(T) = \{v \in L_t : a_i(v) \in T_i \text{ for some } 1 \leq i \leq t\}$ be the set of leaves covered by the solution. Then, the objective is to find a feasible solution maximizing the weight of the leaves covered. This instance is represented as $\mathcal{I} = (\mathcal{T} = ((L_1, \dots, L_t), (a_1, \dots, a_t), \text{Leaf}, w), (k_1, \dots, k_t))$,*

► **Definition 21** (The solution y). *Given cov , and a collection \mathcal{T} of rooted trees, let L_1 denote the set of roots, and let L_i , $i > 1$ denote the set of vertices at i -th level. Furthermore, for any node $v \in L_i$ with $i > 1$, let $a_j(v)$ denote the ancestor of v that belongs to L_j , where $1 \leq j < i$. Then, the solution y is defined as follows.*

$$y(v) = \begin{cases} \text{cov}_1(v) & \text{if } v \in L_1 \\ \min \left\{ \text{cov}_i(v), 1 - \sum_{j < i} \text{cov}_j(a_j(v)) \right\} & \text{if } v \in L_i, i > 1 \end{cases}$$

► **Definition 22** (The Sparse LP).

$$\begin{aligned} \max \quad & \sum_{v \in L_t} w(v)Y(v) \\ & \sum_{u \in L_1} y_u \leq k_1 - t \\ & \sum_{u \in L_i} y_u \leq k_i \quad \forall 2 \leq i \leq t \\ & Y(v) := y_v + \sum_{i=1}^{t-1} y_{a_i(v)} \quad \forall v \in L_t \end{aligned}$$

We now describe two subroutines that are used in the Reduction from Robust t -NU k C to Well-Separated Robust t -NU k C. We use the same notation and convention as in [6]. These two algorithms (Algorithm 3 and Algorithm 4) are named after Hochbaum, and Shmoys [10]; and Chakrabarty, Goyal, and Krishnaswamy [4], respectively.

We construct the t -FF instance based on the sets L_i 's constructed, as follows. Consider some $1 \leq i \leq t-1$, and some $u \in L_i$. Then, for every $v \in \text{Child}_i(u)$, we make v a child of u in a tree T . Note that L_1 is the set of roots of the trees constructed in this way. Then, we define $\text{Leaf}(u) = \{v \in L_t : v \text{ is a leaf in the tree rooted at } v\}$, and let $a_i : L_t \rightarrow \bigcup_{i \leq j \leq t} L_j$ be the ancestor function as defined above. Finally, for every $u \in L_t$, let $w(u) = |\text{Child}_t(u)|$. Then, we return the t -FF instance $\mathcal{I} = (\mathcal{T} = ((L_1, \dots, L_t), (a_1, \dots, a_t), \text{Leaf}, w), (k_1, \dots, k_t))$.

■ **Algorithm 3** HS(Metric space (X, d) , $r \geq 0$, assignment $\text{cov} : X \rightarrow \mathbb{R}^+$).

```

1:  $R \leftarrow 0$ 
2: while  $U \neq \emptyset$  do
3:    $u \leftarrow \arg \max_{v \in U} \text{cov}(v)$ 
4:    $R \leftarrow R \cup \{u\}$ 
5:    $\text{Child}(u) \leftarrow \{v \in U : d(u, v) \leq r\}$ 
6:    $U \leftarrow U \setminus \text{Child}(u)$ 
7: end while
8: return  $R, \{\text{Child}(u) : u \in R\}$ .
```

■ **Algorithm 4** CGK.

Input: Robust t -NUkC instance $\mathcal{I} = ((X, d), (\omega, m), (r_1, \dots, r_t), (k_1, \dots, k_t)), (\alpha_1, \dots, \alpha_t)$, where $\alpha_i > 0$ for $1 \leq i \leq t$,
 $\text{cov} = (\text{cov}_1, \dots, \text{cov}_t)$, where each $\text{cov}_i : X \rightarrow \mathbb{R}^+$

```

1: for  $i = t$  downto 1 do
2:    $(L_i, \{\text{Child}_i(v) : v \in L_i\}) \leftarrow \text{HS}((X, d), \alpha_i r_i, \text{cov}'_i := \sum_{j=1}^i \text{cov}_j)$ 
3: end for
4: Construct and Return a  $t$ -FF instance using  $\{L_i, \text{Child}_i\}_{1 \leq i \leq t}$  as described below.
```

B From Robust t -NUkC to Well-Separated Robust t -NUkC

In this section, we use the round-or-cut framework of [6] to give a Turing reduction from Robust t -NUkC to (polynomially many instances of) *Well-Separated* Robust t -NUkC. Furthermore, c -approximation for a feasible instance of the latter problem will imply an $O(c)$ -approximation for the original instance of Robust t -NUkC.

Round-or-Cut Framework. Let $\mathcal{I} = ((X, d), (\mathbb{1}, m), (k_1, k_2, \dots, k_t), (r_1, r_2, \dots, r_t))$ be the given instance of Robust t -NUkC (we assume that we are working with unit-weight instance, where we want to cover at least m points of X). We adopt the round-or-cut framework of [6] (also [5]) to separate an LP solution from the integer hull of coverages (see Section A in the appendix for the definitions thereof). Even though [6] discuss this for $t = 2$, it easily generalizes to arbitrary $t \geq 2$. Thus, we only sketch the high level idea.

Let $\text{cov} = (\text{cov}_1, \text{cov}_2, \dots, \text{cov}_t : \forall v \in X)$ be a candidate solution returned by the ellipsoid algorithm. First, we check whether $\text{cov}(X) \geq m$, and report as the separating hyperplane if this does not hold. Now, we call CGK Algorithm (see Section A) with $\alpha_1 = 6$, and $\alpha_i = 2$ for all $2 \leq i \leq t$ to get a t -FF instance $(\mathcal{T} = ((L_1, \dots, L_t), (a_1, \dots, a_t), \text{Leaf}, w), (k_1, \dots, k_t))$. Here, for any $i \in [t]$, any distinct $p, q \in L_i$ satisfy that $d(p, q) > 3r_i$. Then, we let $\{y_v : v \in \bigcup_i L_i\}$ be the solution as defined in Section A, see Definition 21. Now we check if $\text{cov}_i(L_i) \leq k_i$ for $i \in [t]$, and report if any of these t inequalities is not satisfied. Finally, the algorithm checks the value of $y(L_1)$, and branches into the following two cases.

In the first case, if $y(L_1) \leq k_1 - t$, then as argued by [6], it can be shown that a sparse LP that is related to the t -FF problem (see Definitions 20 and 22) admits an *almost-integral* solution. That is, a basic feasible solution to the sparse LP contains at most t strictly fractional variables. By rounding up all such variables to 1, one can obtain an $O(1)$ -approximation for the original instance \mathcal{I} . Note that here we need the assumption that the ratio between the values of consecutive radii is at least β – otherwise we can merge the two consecutive radii classes into a single class.

28:20 Non-Uniform k -Center and Greedy Clustering

In the second case, $y(L_1) > k_1 - t$. In this case, we use a generalization of an argument from [6] as follows. We enumerate every subset $Q \subseteq X$ of size at most $t - 1$, and add a ball of radius r_1 around each point in Q . Let X' be the set of points covered by balls of radius r_1 around Q . Then, we modify the weight of the points of X' to be 0, and let $\mathbb{1}_{X \setminus X'}$ be the resulting weight function. Let $\mathcal{I}(Q) = ((X, d), (\mathbb{1}_{X \setminus X'}, m - |X'|), (2r_1, r_2, \dots, r_t), (k_1 - |Q|, k_2, \dots, k_t))$ be the resulting residual instance of *Well-Separated t -NUkC*, where the *well-separatedness* property imposes that the $2r_1$ centers must be chosen from $Y := L_1 \setminus Q$ – note that the distance between any two distinct points in L_1 , and thus Y , is at least $6r_1 = 3 \cdot 2r_1$, i.e., the set Y is well-separated w.r.t. the new radius r_1 . An argument from [6] implies that if \mathcal{I} is feasible, then either (a) at least one of the well-separated instances $\mathcal{I}(Q)$ is feasible for some $Q \subseteq X$ of size at most $t - 1$, or (b) the hyperplane $y(L_1) \leq k_1 - t$ separates the LP solution cov from the integer hull of coverages. Furthermore, an argument from [6] implies that a constant approximation to any of the instances implies a constant approximation to \mathcal{I} .

Note that the ellipsoid algorithm terminates in polynomially many iterations, and each iteration produces at most n^t instances of *Well-Separated Robust t -NUkC*. Thus, we get the following theorem.

► **Theorem 23.** *Suppose there exists an algorithm that, given an instance \mathcal{J} of Well-Separated Robust t -NUkC, in time $f(n, t)$, either finds an α -approximation to \mathcal{J} , or correctly determines that \mathcal{J} is not feasible. Then, there exists an algorithm to obtain an $c \cdot \alpha$ -approximation for any instance of Robust t -NUkC, running in time $n^{O(t)} \cdot f(n, t)$.*

Most Classic Problems Remain NP-Hard on Relative Neighborhood Graphs and Their Relatives

Pascal Kunz ✉ 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Till Fluschnik ✉ 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Rolf Niedermeier 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Malte Renken ✉ 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

Proximity graphs have been studied for several decades, motivated by applications in computational geometry, geography, data mining, and many other fields. However, the computational complexity of classic graph problems on proximity graphs mostly remained open. We study 3-COLORABILITY, DOMINATING SET, FEEDBACK VERTEX SET, HAMILTONIAN CYCLE, and INDEPENDENT SET on the following classes of proximity graphs: relative neighborhood graphs, Gabriel graphs, and relatively closest graphs. We prove that all of the aforementioned problems remain NP-hard on these graphs, except for 3-COLORABILITY and HAMILTONIAN CYCLE on relatively closest graphs, where the former is trivial and the latter is left open. Moreover, for every NP-hard case we additionally show that no $2^{o(n^{1/4})}$ -time algorithm exists unless the Exponential-Time Hypothesis (ETH) fails, where n denotes the number of vertices.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Problems, reductions and completeness

Keywords and phrases Proximity Graphs, Relatively Closest Graphs, Gabriel Graphs, 3-Colorability, Dominating Set, Feedback Vertex Set, Hamiltonian Cycle, Independent Set, Exponential-Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.29

Related Version *Full Version:* <https://arxiv.org/abs/2107.04321> [26]

Funding *Pascal Kunz:* Supported by DFG Research Training Group 2434 “Facets of Complexity”.

Till Fluschnik: Supported by DFG, projects TORE, NI 369/18, and MATE, NI 369/17.

Malte Renken: Supported by DFG, project MATE, NI 369/17.

Acknowledgements This work is based on the first author’s master’s thesis.

In memory of Rolf Niedermeier, our colleague, friend, and mentor, who sadly passed away before this paper was published.

1 Introduction

Proximity graphs describe the distance relationships between points in the plane or higher-dimensional structures. They are mostly studied in computational geometry, yet arise in numerous fields of science and engineering from geography to pattern recognition [24, 35]. In this paper, we study the computational complexity of classic NP-complete problems on three specific proximity graphs: relative neighborhood graphs (RNGs) [34], Gabriel graphs (GGs) [18], and relatively closest graphs (RCGs) [27]. All three are subgraphs of the



© Pascal Kunz, Till Fluschnik, Rolf Niedermeier, and Malte Renken;
licensed under Creative Commons License CC-BY 4.0

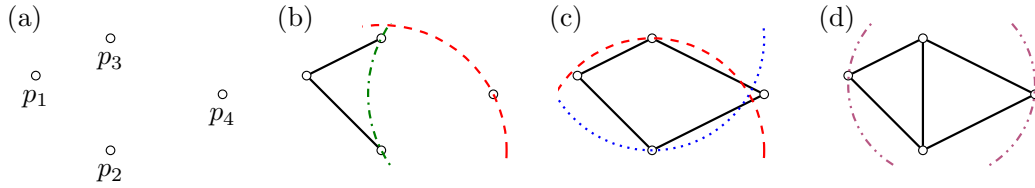
18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 29; pp. 29:1–29:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** (a) Set P of four points $p_1 = (0, 2)$, $p_2 = (2, 0)$, $p_3 = (2, 3)$, and $p_4 = (5, 3/2)$. (b) RCG on P . Point p_4 is not adjacent with p_2 , since p_3 lies in the region of influence indicated with green (dashdotted) and red (dashed). (c) RNG on P . Points p_2 and p_3 are not adjacent since p_1 is in their region of influence indicated with red (dashed) and blue (dotted). (d) GG on P . Points p_1 and p_4 are not adjacent since p_2 is in their region of influence indicated with magenta (dashdotted).

better-known Delaunay triangulation (DT). For DTs, the complexity of the restrictions of some classic NP-complete graph problems have already been resolved [12, 15] and we extend this research to these three classes.

RCGs, RNGs, and GGs are examples of empty region graphs [10]. Every pair of points is associated with a region in the plane, their region of influence, and is connected by an edge if there is no other point in that region (see Figure 1). In RNGs (RCGs), two points' region of influence is the intersection of open (closed) disks centered on each of the points with a radius equal to their distance. In a GG, two points' region of influence is a closed disk whose center is midway between them and whose diameter is their distance.

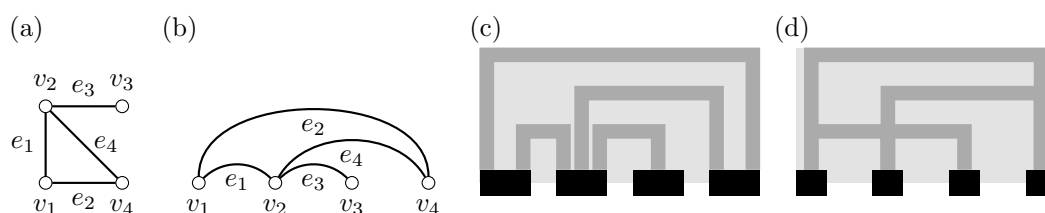
Motivation. Consider a railway network. It may make sense to build a track directly from one city to another, if there is no third city in between. If we interpret the area between two cities as a region of influence, then this makes proximity graphs such as RNGs plausible models for such networks. One might want to build as few maintenance facilities for the network as possible such that every track has a facility at one of its endpoints. This is an instance of the VERTEX COVER (VC) problem (closely related to INDEPENDENT SET (IS)). While VC is NP-hard on general graphs, one wonders whether it might be easier on proximity graphs. We will show that the problem remains NP-hard on RCGs, RNGs, and GGs. For other problems, there are similar application scenarios.

Related Work. Existing combinatorial results on the three graph classes include listing forbidden subgraphs, examples of graphs contained in each class, and bounds on the edge density [6, 13, 24, 30, 37]. Much algorithmic research on proximity graphs has focused on devising algorithms that efficiently compute the proximity graph from a point set (see [23] and [31] for an overview). On Delaunay triangulations, HAMILTONIAN CYCLE is NP-hard [15], whereas 3-COLORABILITY is polynomial-time solvable [12]. Cimikowski conjectured 3-COLORABILITY to be NP-hard on RNGs and GGs [11]. Furthermore, he proposed a heuristic for coloring GGs and a linear-time algorithm for computing a 4-coloring in RNGs [12], but the latter has some issues, which we will discuss in Section 3.

Our Contributions. Table 1 summarizes our results. We prove that 3-COLORABILITY (3-COL), DOMINATING SET (DS), FEEDBACK VERTEX SET (FVS), HAMILTONIAN CYCLE (HC), and INDEPENDENT SET (IS) remain NP-hard on RNGs and GGs, in particular confirming the aforementioned conjecture by Cimikowski [11]. On RCGs, 3-COLORABILITY is trivial, but we prove that DOMINATING SET, FEEDBACK VERTEX SET, and INDEPENDENT SET remain NP-hard. All our NP-hardness results hold true even for graphs of fairly small maximum

■ **Table 1** Overview of our results. Herein, Δ denotes the maximum vertex degree.
 † No $2^{o(n^{1/4})}$ -time algorithm exists unless the ETH fails, where n denotes the number of vertices.

| | RCGs | RNGs | GGs |
|---------------------------|-------------|--|-----|
| 3-COLORABILITY (3-COL) | trivial | NP-hard [†] , even if $\Delta = 7$ (Thm. 3.1) | |
| DOMINATING SET (DS) | | NP-hard [†] , even if $\Delta = 4$ (Thm. 6.1) | |
| FEEDBACK VERTEX SET (FVS) | | NP-hard [†] , even if $\Delta = 4$ (Thm. 4.1) | |
| HAMILTONIAN CYCLE (HC) | <i>open</i> | NP-hard [†] , even if $\Delta = 4$ (Thm. 5.1) | |
| INDEPENDENT SET (IS) | | NP-hard [†] , even if $\Delta = 4$ (Thm. 6.1) | |



■ **Figure 2** (a) A graph G with vertex set $\{v_1, \dots, v_4\}$. (b) A 1-page book embedding of G . (c) and (d) Illustration of our technique, where black rectangles correspond to vertex gadgets, thick gray lines to edge/connector gadgets, and light gray areas indicate filler gadgets.

degree (7 in the case of 3-COL, and 4 in all other cases, yielding a dichotomy between polynomial-time solvability and NP-hardness in the case of FVS). We complement each NP-hardness result with a running-time lower bound of $2^{o(n^{1/4})}$ based on the Exponential-Time Hypothesis, where n is the number of vertices. The fastest known algorithms for these problems run in time $2^{O(n^{1/2})}$ on planar graphs and it remains open whether a running time between these lower and upper bounds can be achieved on proximity graphs. Many details and proofs (marked with ★) are deferred to the full version of this paper.

Our Technique. In our NP-hardness proofs (see Table 1), we give polynomial-time many-one reductions from each problem’s restriction to planar graphs with maximum degree 3 or 4. We proceed as follows (see Figure 2 for an illustration). We exploit the fact that for any planar graph with maximum degree at most 4, we can compute in polynomial time a 2-page book embedding [3], a very structured representation of the input graph. Then, we translate the book embedding’s structure into a grid-like structure. Each reduction uses three types of gadgets: to represent vertices, to represent edges, and to fill the space between them in order to prevent the appearance of unwanted edges between the other gadgets.

2 Preliminaries

Let $\mathbb{N} := \{1, 2, 3, \dots\}$ and $\mathbb{N}_0 := \{0\} \cup \mathbb{N}$. We use basic notions from graph theory [14].

Proximity graphs. Let $d: \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ denote the Euclidean distance between two points. The *open* and *closed ball* with radius $r > 0$ and center $p \in \mathbb{R}^2$ are

$$B_r(p) := \{q \in \mathbb{R}^2 \mid d(p, q) < r\} \text{ and } \bar{B}_r(p) := \{q \in \mathbb{R}^2 \mid d(p, q) \leq r\}.$$

The Delaunay triangulation of $P \subseteq \mathbb{R}^2$ (see, e.g., [4, Ch. 9]) is denoted by $DT(P)$.

A *template region* [10] is a function $R: \binom{\mathbb{R}^2}{2} \rightarrow 2^{\mathbb{R}^2}$ that assigns a region of the plane, called the *region of influence*, to each pair of points in the plane. Given a template region R_C and points $p_1, p_2 \in \mathbb{R}^2$, a third point $p_3 \in \mathbb{R}^2 \setminus \{p_1, p_2\}$ is a C -*blocker* for $\{p_1, p_2\}$ if $p_3 \in R_C(p_1, p_2)$. For a finite set of points $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$, the C -graph induced by P is $\mathcal{C}(P) := (\{v_1, \dots, v_n\}, E_{\mathcal{C}(P)})$ with

$$E_{\mathcal{C}(P)} := \{\{v_i, v_j\} \mid (R_C(p_i, p_j) \cap P) \setminus \{p_i, p_j\} = \emptyset \text{ and } i \neq j\}.$$

The class \mathcal{C} contains a graph G if there is a finite set of points $P \subseteq \mathbb{R}^2$ with $\mathcal{C}(P) = G$.

We are interested in three template regions and the graph classes defined by them:

Relatively closest graphs (RCGs): Defined by

$$\begin{aligned} R_{\text{RCG}}(p_1, p_2) &:= \{p_3 \in \mathbb{R}^2 \mid d(p_1, p_2) \geq \max\{d(p_1, p_3), d(p_2, p_3)\}\} \\ &= \overline{B}_{d(p_1, p_2)}(p_1) \cap \overline{B}_{d(p_1, p_2)}(p_2). \end{aligned}$$

Relative neighborhood graphs (RNGs): Defined by

$$\begin{aligned} R_{\text{RNG}}(p_1, p_2) &:= \{p_3 \in \mathbb{R}^2 \mid d(p_1, p_2) > \max\{d(p_1, p_3), d(p_2, p_3)\}\} \\ &= B_{d(p_1, p_2)}(p_1) \cap B_{d(p_1, p_2)}(p_2). \end{aligned}$$

Gabriel graphs (GGs): Defined by

$$R_{\text{GG}}(p_1, p_2) := \{p_3 \in \mathbb{R}^2 \mid d(p_1, p_2)^2 \geq d(p_1, p_3)^2 + d(p_2, p_3)^2\} = \overline{B}_{d(p_1, p_2)/2}(q),$$

where q is the midpoint between p_1 and p_2 .

A C -*embedding* of a graph $G = (V, E)$ is a mapping $\text{emb}: V \rightarrow \mathbb{R}^2$ such that $\mathcal{C}(\text{emb}(V)) = G$.¹ Of course, $G \in \mathcal{C}$ if and only if G admits a C -embedding.

For any finite point set P , it holds that $E_{\text{RCG}(P)} \subseteq E_{\text{RNG}(P)} \subseteq E_{\text{GG}(P)} \subseteq \text{DT}(P)$ [13]. RCGs cannot contain K_3 as a subgraph and none of the three can contain K_4 or $K_{2,3}$ [13, 30, 37]. Moreover, $p_3 \in R_{\text{GG}}(p_1, p_2)$ if and only if the angle at p_3 formed by the lines to p_1 and p_2 is at most 90° [30]. Finally, the following lemma will be used to prove that graphs are in each graph class:

► **Lemma 2.1** ([30]). *Let P be a set of points in the plane and G the RCG, RNG, or GG induced by P . Then, the straight-line drawing of G induced by P is planar.*

Book embeddings. Our NP-hardness proofs use 2-page book embeddings. A k -page book embedding of a graph $G = (V, E)$ consists of

- (i) an edge partition $E = E_1 \uplus \dots \uplus E_k$, and
- (ii) for every $i \in \{1, \dots, k\}$, a planar embedding emb_i of (V, E_i) in $\mathbb{R} \times \mathbb{R}_{\geq 0}$, where $\text{emb}_i(v) = \text{emb}_j(v) \in \mathbb{R} \times \{0\}$ for every $v \in V$, $i, j \in \{1, \dots, k\}$.

The following result due to Bekos et al. [3] will play an important role in this work:

► **Theorem 2.2** ([3]). *Every planar graph with maximum degree at most 4 admits a 2-page book embedding. Such an embedding can be computed in quadratic time.*

The following terminology will be useful in our NP-hardness proofs. Consider a graph $G = (V, E)$ and a 2-page book embedding of G . Let v_1, \dots, v_n be the vertices of the graph ordered in such a way that $\text{emb}_i(v_j) < \text{emb}_i(v_{j+1})$ for $i \in \{1, 2\}$ and every $j \in \{1, \dots, n-1\}$. We

¹ To simplify notation, we write v instead of $\text{emb}(v)$ to refer to the point at which vertex v is embedded.

will say that v_1, \dots, v_n is the *order in which the vertices appear on the spine*. Let $r \in \{1, 2\}$. We will use $N_r(v) := \{v' \mid \{v, v'\} \in E_r\}$ to denote v 's E_r -neighborhood and $\deg_r(v) := |N_r(v)|$ to denote v 's E_r -degree. For an edge $e = \{v_i, v_j\}$, $i < j$, define its *length* as $\ell(e) := j - i$. The *interior* of $e \in E_r$ is

$$\text{int}(e) := \{e' = \{v_{i'}, v_{j'}\} \in E_r \mid i \leq i' < j' \leq j, e' \neq e\}.$$

The *height* of e is $h(e) := 1 + \max\{0, h(e') \mid e' \in \text{int}(e)\}$. Note that, because the height of any edge only depends on the height of shorter edges, edge height is well-defined. The length and height of an edge are both in $\mathcal{O}(n)$. The E_r -height of a vertex v is

$$h_r(v) := \max\{0, h(e) \mid v \in e \in E_r\}, \quad \text{where} \quad h(e) := 1 + \max\{0, h(e') \mid e' \in \text{int}(e)\}.$$

Let $h_r(G) := \max\{h_r(v_i) \mid i \in \{1, \dots, n\}\}$. For every vertex v_i , we order its incident edges in E_r as follows. If $N_r(v_i) = \{v_{j_1}, \dots, v_{j_k}\}$ with $j_1 < \dots < j_c < i < j_{c+1} < \dots < j_k$, then the order of the edges is $\{v_i, v_{j_c}\} < \dots < \{v_i, v_{j_1}\} < \{v_i, v_{j_k}\} < \dots < \{v_i, v_{j_{c+1}}\}$.

Grid structure. The graphs we build in our reductions will all have a grid-like structure. With the exception of the reduction for HAMILTONIAN CYCLE, we group their vertices into (x, y) -corners with $(x, y) \in \mathbb{Z}^2$ and there will be a corner for every (x, y) within certain bounds, which depend on the problem in question as well as the size and structure of the input graph. In the embedding, the vertices forming the (x, y) -corner will be in $\overline{B}_r(x, y)$ for a suitable $r > 0$. Some vertices are not part of any corner and are called *intermediate vertices*. They are usually located midway between two corners. Each corner can have one or multiple dedicated right, top, left, and bottom *connecting vertices*. If a corner consists of a single vertex, that vertex always simultaneously acts as the right, top, left, and bottom connecting vertex of that corner. The connecting vertices of a corner are the only ones that may have neighbors outside of that corner. For any $(x, y) \in \mathbb{Z}^2$, we say that the vertices in the (x, y) , $(x + 1, y)$, $(x, y + 1)$, and $(x + 1, y + 1)$ -corners along with any intermediate vertices that are adjacent to vertices in two of the aforementioned corners jointly form a *grid face*.

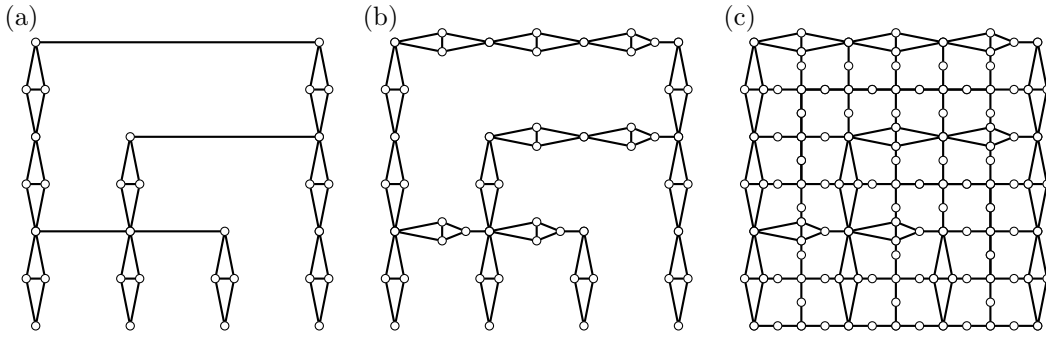
Exponential-Time Hypothesis. The *Exponential-Time Hypothesis (ETH)*, introduced by Impagliazzo and Paturi [22], has become a useful tool to give more precise running-time lower bounds than the more classical dichotomy between polynomial-time solvable and NP-hard problems (for an overview, we refer to [28]). This conjecture states:

► **Hypothesis 2.3** (Exponential-Time Hypothesis [22]). *There is some fixed $c > 0$ such that 3-CNF-SAT is not solvable in $2^{cn} \cdot (n + m)^{\mathcal{O}(1)}$ time, where n and m denote the numbers of variables and clauses, respectively.*

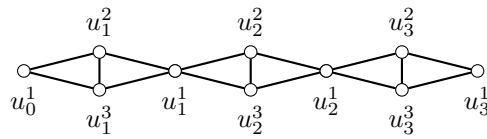
3 3-Colorability

We start with the 3-COLORABILITY problem. A *3-coloring* of a graph $G = (V, E)$ is a function $c: V \rightarrow \{1, 2, 3\}$ such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$. In the 3-COLORABILITY problem (3-COL), one is given a graph as input and asked to decide whether it admits a 3-coloring. Every RCG is 3-colorable [13] because RCGs do not contain any 3-cycles and every planar graph without 3-cycles is 3-colorable by Grötzsch's theorem [21]. As a result, 3-COL is trivial when restricted to RCGs. Regarding RNGs and GGs, we prove the following.

► **Theorem 3.1 (★).** *3-COLORABILITY on RNGs and on GGs is NP-hard, even if the maximum degree is 7. Moreover, unless the ETH fails, it admits no $2^{\mathcal{O}(n^{1/4})}$ -time algorithm where n is the number of vertices.*



■ **Figure 3** The construction proving Theorem 3.1 applied to the example graph from Figure 2. (a) The graph after steps 1–2. (b) The graph after steps 1–3. (c) The final graph.



■ **Figure 4** A coloring path of length 3.

This confirms a conjecture by Cimikowski [11]. Our proof is based on a polynomial-time many-one reduction from the NP-hard [19] 3-COLORABILITY problem on planar graphs with maximum degree 4. The complete proof is deferred to the full version, but we give the construction and some intuition concerning its correctness. We give an illustration of the construction in Figure 3 and the following high-level description: After computing a 2-page book embedding, we replace each vertex by a “color-preserving” path (Figure 3(a)). We further replace each original edge by a path enforcing different colors for the respective endpoints (Figure 3(b)). Finally, we fill the remaining space with a subdivided grid (Figure 3(c)), which preserves 3-colorability and turns the graph simultaneously into an RNG and a GG.

We will use so-called *coloring paths* (see Figure 4 for an illustration), which essentially allows us to copy the color of a vertex. The *coloring path* of length k from u_0^1 to u_k^1 is the graph $\tilde{P}_k := (V_k, E_k)$ with:

$$V_k := \{u_0^1\} \cup \{u_i^1, u_i^2, u_i^3 \mid i \in \{1, \dots, k\}\} \text{ and}$$

$$E_k := \{\{u_{i-1}^1, u_i^2\}, \{u_{i-1}^1, u_i^3\}, \{u_i^2, u_i^3\}, \{u_i^2, u_i^1\}, \{u_i^3, u_i^1\} \mid i \in \{1, \dots, k\}\}.$$

We will call u_i^1 the i -th *center vertex*, u_i^2 the i -th *left vertex*, and u_i^3 the i -th *right vertex*.

► **Construction 3.2.** Let $G = (V, E)$ be an undirected planar graph of maximum degree 4. We will construct a graph $G' = (V', E')$ and subsequently show that G' is both an RNG and a GG and that G is 3-colorable if and only if G' is. The vertex set of G' will mostly consist of groups of vertices called (x, y) -*corners* where $2 \leq x \leq 2n$ and $-2h_2(G) \leq y \leq 2h_1(G)$. Each corner consists of either a single vertex or of a pair of adjacent vertices. Corners can have dedicated top, left, right, and bottom *connecting vertices*, some of which may coincide. For example, if a corner consists of a single vertex, that vertex simultaneously forms all four connecting vertices. Finally, there will be some *intermediate vertices* that are not part of any corner.

We start with $G' := G$.

Step 1: Compute a 2-page book embedding of G . Let v_1, \dots, v_n be the vertices of G enumerated in the order in which they appear on the spine, and let $\{E_1, E_2\}$ denote the partition of E .

Step 2: Replace every vertex v_i with a coloring path of length $h_1(v_i) + h_2(v_i)$. Every edge e of G incident to v_i is now instead attached to the $(h_2(v_i) + h(e))$ -th center vertex, if $e \in E_1$, or the $(h_2(v_i) - h(e))$ if $e \in E_2$. For $r = 0, \dots, h_1(v_i) + h_2(v_i)$, the r -th center vertex of that path forms the $(2i, 2r - 2h_2(v_i))$ -corner. For $r = 1, \dots, h_1(v_i)$, the r -th left and right vertices jointly form the $(2i, 2r - 1 - 2h_2(v_i))$ -corner. The left vertex is the left connecting vertex of this corner and the right vertex is the right connecting vertex.

Step 3: For every edge $e = \{v_i, v_j\} \in E_1$, $i < j$, replace the corresponding edge of G' with a coloring path of length $\ell(e)$. Identify the first vertex of that path with the $(2i, 2h(e))$ -corner (which consists of a single vertex). Denote the last vertex of that path by w . Add an edge from w to the $(2j, 2h(e))$ -corner. For $r = 1, \dots, \ell(e) - 1$, the r -th center vertex of that path is the $(2i + 2r, 2h(e))$ -corner. The vertex w , which is the $\ell(e)$ -th center vertex, is an intermediate vertex. For $r = 1, \dots, \ell(e)$, the r -th left and right vertices jointly form the $(2i + 2r - 1, 2h(e))$ -corner. The left vertex is the top connecting vertex of this corner and the right vertex is the bottom connecting vertex.

Step 4: For every (x, y) with $2 \leq x \leq 2n$ and $0 \leq y \leq 2h_1(G)$, if an (x, y) -corner was not added in one of the previous two steps, then add a single vertex, which becomes the (x, y) -corner, to G' . In that case add an edge from the (x, y) -corner to the top connecting vertex of the $(x, y - 1)$ -corner, to the left connecting vertex of the $(x + 1, y)$ -corner, and so on. Subdivide each of these edges once, introducing four new intermediate vertices.

Note that Steps 3 and 4 take only E_1 into account. These steps must be repeated analogously for E_2 , using negative y -coordinates. \lrcorner

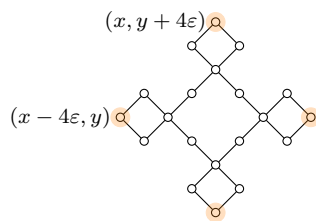
The correctness builds on the following two facts. By replacing each original edge with a path of coloring paths in Steps 2 and 3, we enforce different colors for the respective endpoints (see Figure 3(b)). Filling the remaining space with a subdivided grid in Step 4 (see Figure 3(c)) preserves 3-colorability and turns the graph both into an RNG and a GG.

Further remarks. We remark that RNGs and GGs with maximum degree 3 are always 3-colorable. This follows from Brooks' theorem [9, 29], which states that any graph with maximum degree $\Delta \geq 3$ is Δ -colorable, if it contains no $(\Delta + 1)$ -clique. RNGs and GGs contain no 4-cliques (see Section 2). It remains open whether 3-COL can be solved in polynomial time on RNGs or GGs with maximum degree between 4 and 6.

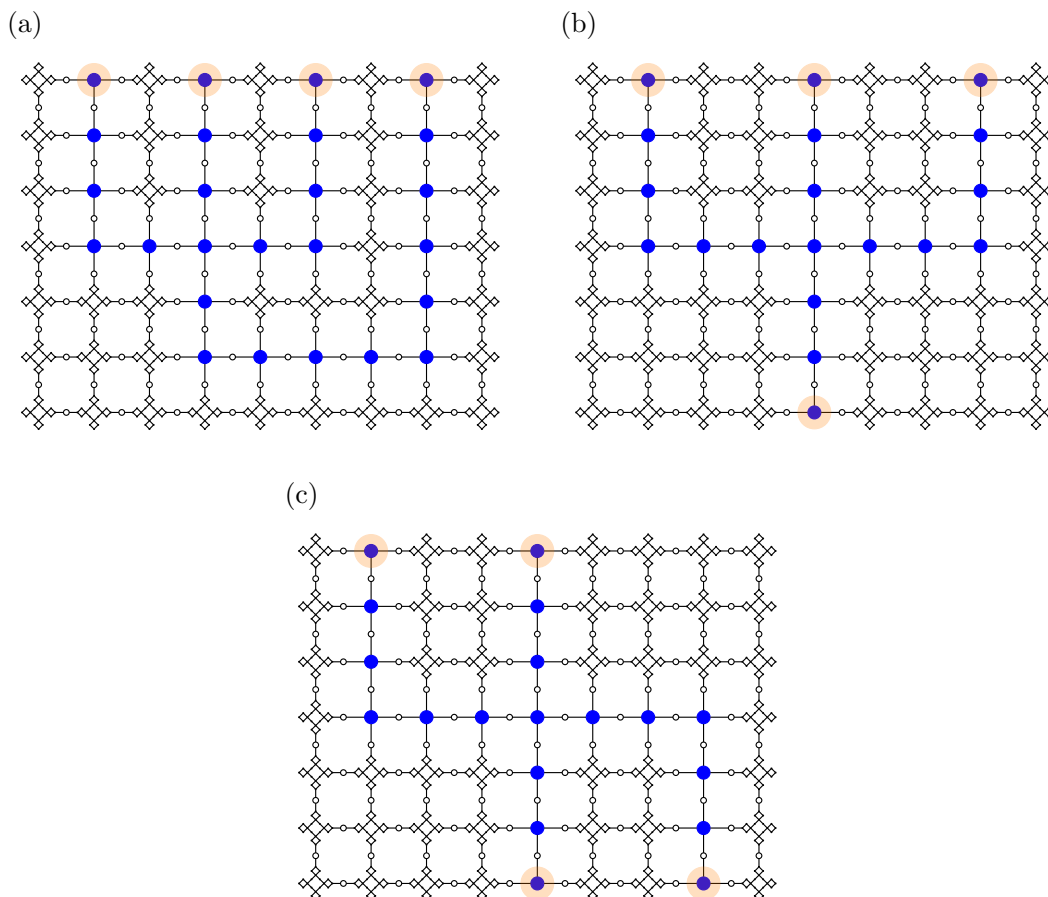
By the well-known four color theorem, all planar graphs are 4-colorable. The fastest known algorithm to compute a 4-coloring of a planar graph has quadratic running time [32]. For RNGs, Cimikowski [12] proposed an algorithm for computing 4-colorings in linear time. However, this algorithm is based on the claim [37, Lemma 4.2] that the wheel graph W_6 cannot occur as subgraph of an RNG. This claim was disproved by Bose et al. [6]. Cimikowski's algorithm additionally implicitly assumes that RNGs are closed under minors, since the algorithm sometimes merges two adjacent vertices. This can lead to graphs that are not RNGs. Thus, it remains open whether or not a linear-time algorithm for this task exists.

4 Feedback Vertex Set

We now turn our attention to the FEEDBACK VERTEX SET problem. In a graph $G = (V, E)$, a *feedback vertex set* is a set $X \subseteq V$ such that $G - X$ is a forest. In the FEEDBACK VERTEX SET problem (FVS), one is given a graph and $k \in \mathbb{N}_0$ and asked whether the graph contains a feedback vertex set of size at most k . We prove the following:



■ **Figure 5** Buffer at position $(x, y) \in \mathbb{R}^2$ with highlighted outer vertices.



■ **Figure 6** The (a) (4,0)-, (b) (3,1)-, and (c) (2,2)-vertex gadget with highlighted outlets.

► **Theorem 4.1 (★).** FEEDBACK VERTEX SET on RCGs, on RNGs, and on GGs is NP-hard, even if the maximum degree is 4. Moreover, unless the ETH fails, it admits no $2^{o(n^{1/4})}$ -time algorithm where n is the number of vertices.

Our proof is based on a polynomial-time many-one reduction from the NP-complete [33] FEEDBACK VERTEX SET on planar graphs of maximum degree 4. We only give an intuitive high-level description here, deferring the complete proof to the full version, and illustrate the resulting graph in Figure 7. We utilize the graph pictured in Figure 5 and call it a “buffer”. Adding a copy of this graph to an existing graph increases the size of a minimum feedback vertex set by exactly 4 even if the vertices marked as outer vertices are each connected to exactly one vertex in the original graph.

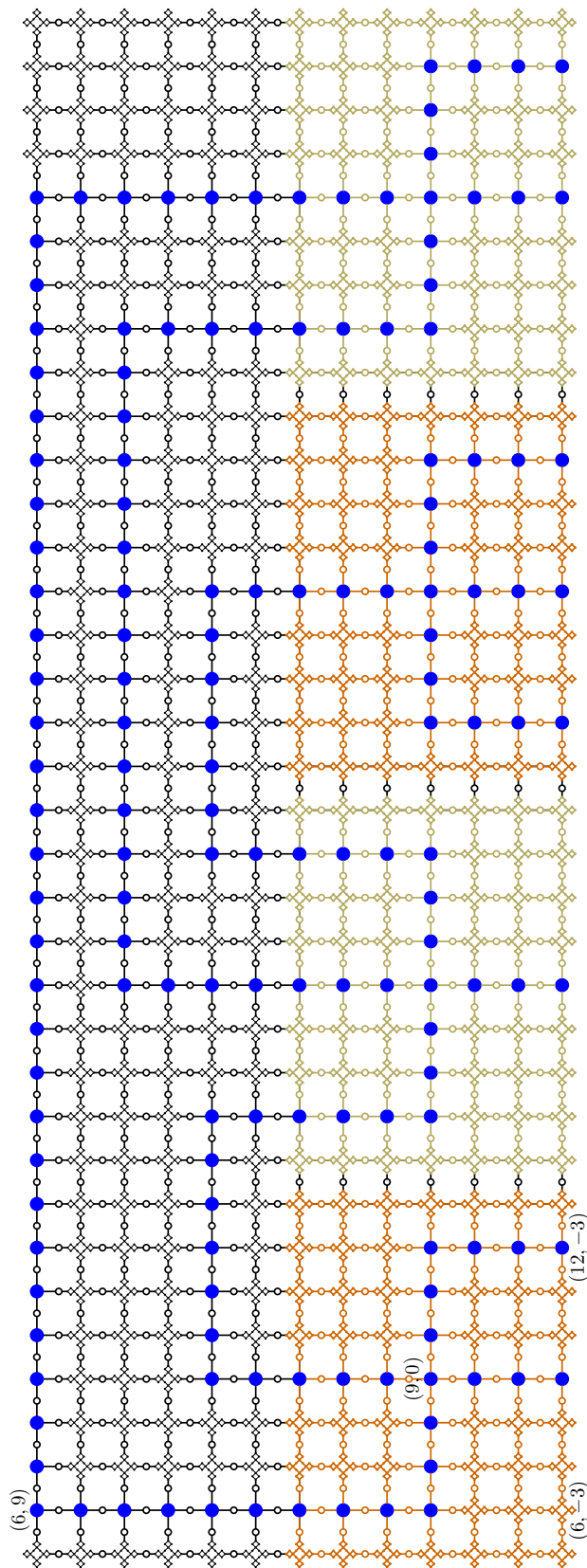
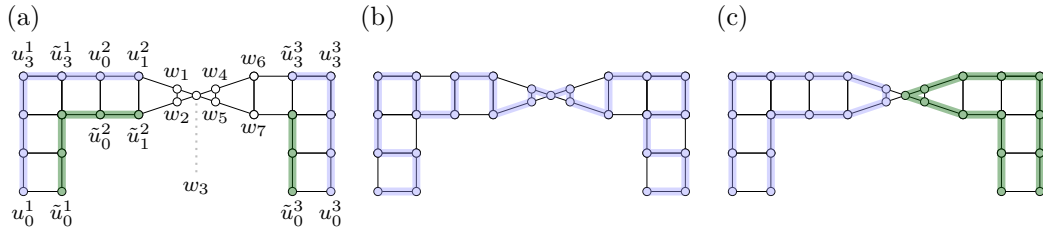


Figure 7 The construction proving Theorem 4.1 applied to the graph in Figure 2. The large blue vertices along with the degree-2 vertices between them form a subdivision of the original graph (with a forest added). The bottom half consists of four vertex gadgets, colored alternatingly orange and yellow.



■ **Figure 8** The ladder path $L_{4,2}$ with (a) selected vertex labels and inside/outside edges highlighted in dark green / light blue; (b) a traversal; (c) a partial/full cover (light blue / dark green).

We compute a 2-page book embedding of the input graph and represent every vertex v in that graph by a gadget which depends on $\deg_1(v)$ and $\deg_2(v)$. The three possible gadgets for a vertex with degree 4 are pictured in Figure 6. They contain many copies of the buffer graph. We then represent edges in the input path by long paths between the outlets of the vertex gadgets that represent the edges' endpoints. We fill the space between these paths with copies of the buffer.

The resulting graph has the following structure once all buffers have been deleted (this is the blue subgraph in Figure 7). Iteratively removing degree-1 vertices from this graph results in a subdivision of the input graph. This leads to our correctness proof for this reduction, as FVS is invariant under deletion of degree-1 vertices and under subdivisions.

Note that FVS is polynomial-time solvable on graphs with maximum degree 3 [36].

5 Hamiltonian Cycle

In a graph $G = (V, E)$, a *Hamiltonian cycle* is a cycle that visits every vertex in V exactly once. In the HAMILTONIAN CYCLE problem (HC), one is given a graph and asked to decide whether the graph contains a Hamiltonian cycle. As the proof of the following is more complex than for previous problems, we will give more details.

► **Theorem 5.1.** HAMILTONIAN CYCLE on RNGs and on GGs is NP-hard, even if the maximum degree is 4. Moreover, unless the ETH fails, it admits no $2^{o(n^{1/4})}$ -time algorithm where n is the number of vertices.

To prove Theorem 5.1, we give a polynomial-time many-one reduction from the restriction of HAMILTONIAN CYCLE to 3-regular planar graphs, for which we have the following.

► **Proposition 5.2** ([20, 28]). HAMILTONIAN CYCLE on 3-regular planar graphs is NP-hard and, unless the ETH fails, admits no $2^{o(n^{1/2})}$ -time algorithm where n is the number of vertices.

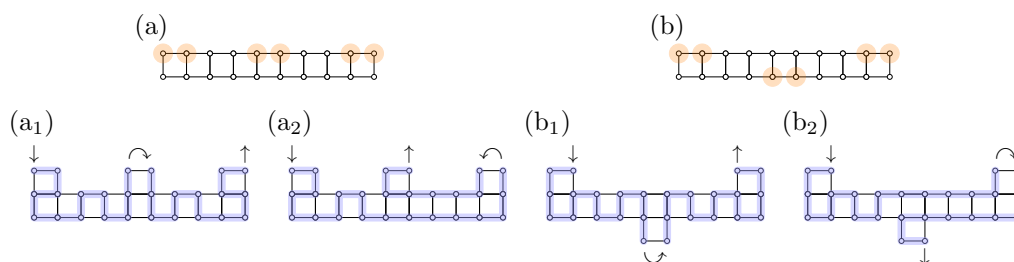
The reduction in the proof of Theorem 5.1 consists of two Hamiltonicity-preserving modifications: gadget expansion (Section 5.1) and face filling (Section 5.2).

5.1 Gadget Expansion

The gadgets that will replace the edges are called *ladder paths*. For $k_1, k_2 \in \mathbb{N}$, the *ladder path* with length (k_1, k_2) is the graph $L_{k_1, k_2} = (V, E)$, where

$$V = \{u_i^1, \tilde{u}_i^1, u_i^3, \tilde{u}_i^3 \mid i \in \{0, \dots, k_1 - 1\}\} \cup \{u_i^2, \tilde{u}_i^2 \mid i \in \{0, \dots, k_2 - 1\}\} \cup \{w_1, \dots, w_7\}.$$

The edges are given using the example pictured in Figure 8(a) and listed explicitly in the full version.



■ **Figure 9** (a) (3,0)-vertex gadget and (b) (2,1)-vertex gadget, with highlighted outlets. (a₁), (a₂) and (b₁), (b₂) show two generic ways a Hamiltonian cycle can pass through each vertex gadget.

The vertices u_i^j, \tilde{u}_i^j with $j \in \{1, 2\}$ along with w_1 and w_2 form the *first half* of the ladder path and those with $j = 3$ along with w_4, \dots, w_7 form the *second half*. The vertex w_3 is the transitional vertex. The vertices w_1, \dots, w_7 form the *switch*. The vertices u_0^1 and \tilde{u}_0^1 form the *end* of the first half, while u_0^3 and \tilde{u}_0^3 form the *end* of the second half. The edges highlighted in light blue in Figure 8(a) will be called *outside edges*, while the edges highlighted in dark green are *inside edges*. An edge $\{u_i^j, u_{i+1}^j\}$ or $\{\tilde{u}_i^j, \tilde{u}_{i+1}^j\}$ is called *even* if i is even.

A *traversal* of a ladder path is a path that begins in either vertex at one end of the ladder path, terminates in either vertex at the other end, and visits every vertex on the ladder path and no other vertex. A *partial cover* of a half of a ladder path is a path that begins in either vertex in the end of the half, terminates in the other vertex in that end, and visits every vertex of that half, but no other vertex. A *full cover* of a half additionally visits the transitional vertex. Examples of a traversal, a partial cover, and a full cover are pictured in Figure 8(b) and (c). The main property of ladder paths is that any Hamiltonian cycle must either contain a traversal, or a full and a half cover of each ladder path.

► **Lemma 5.3** (★). *Suppose that the Hamiltonian graph $G = (V, E)$ contains a ladder, that the only vertices on the ladder path with neighbors outside of the ladder path are on its ends, and that the vertices on the ends each have no more than one neighbor outside of the ladder path. Then, any Hamiltonian cycle in G contains either:*

- a traversal of the ladder path or
- a partial cover of one of its halves and a full cover of the other half.

For a ladder path and a Hamiltonian cycle in a graph, we say that the ladder path is *traversed* if the Hamiltonian cycle contains a traversal of the ladder path. Otherwise, it is *covered*.

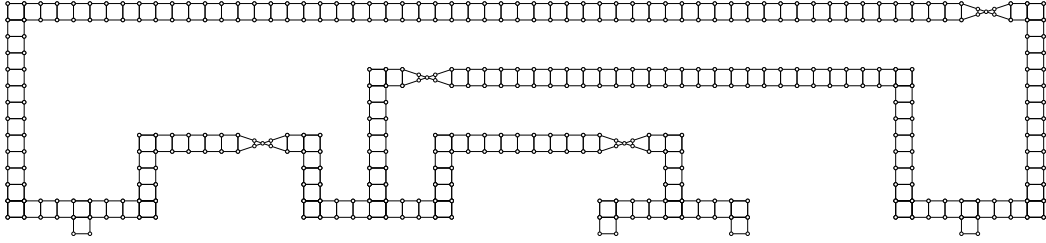
Next, we discuss the vertex gadgets. Recall that the graph G is assumed to be 3-regular. We will use four types of vertex gadgets. Each vertex gadget consist of a grid of size 2×10 , with the only difference being the position of their three *outlets*, which are designated vertex pairs to which the ladder paths representing the edges will be connected. The (3,0)-*vertex gadget* and the (2,1)-*vertex gadget* are pictured in Figure 9 with the three outlets highlighted. The (0,3)-*vertex gadget* and the (1,2)-*vertex gadget* are obtained from the former two by mirroring along the horizontal axis. The value (i, j) will be called the *type* of the gadget.

We will refer to the outlets as *top* or *bottom* outlets, as well as the *left*, *middle* or *right* outlet, with the obvious meaning. The left and right outlets are also called the *outer outlets*.

We will now define the *gadget expansion* of a 3-regular graph G , consisting of a graph G' and a straight-line embedding emb resulting from applying the following steps to G .

► **Construction 5.4** (Gadget expansion). Start with G' being the empty graph.

Step 1: Compute a 2-page book embedding of G and let v_1, \dots, v_n be the vertices of G in the order in which they appear on the spine, and let $\{E_1, E_2\}$ denote the partition of E .



■ **Figure 10** Gadget expansion of the graph pictured in Figure 2. This graph is not 3-regular, but we may assume that there are further edges in E_2 .

Step 2: For every vertex $v_i \in V$ add to G' a $(\deg_1(v_i), \deg_2(v_i))$ -vertex gadget. Position the vertices of this gadget at $(18i + x, y)$ with $x \in \{0, \dots, 9\}$ and $y \in \{0, 1\}$, as in Figure 9.

Step 3: For every edge $e = \{v_i, v_j\}$ in E_1 , $i < j$, add to G' a ladder path L_{k_1, k_2} connected to an outlet in v_i 's vertex gadget and an outlet in v_j 's vertex gadget as follows. Recall the ordering of the edges incident to a vertex defined in the preliminaries. If e is the r -th edge at v_i and the s -th edge at v_j , then attach said ladder path to the r -th top outlet from the left of v_i 's vertex gadget and to the s -th top outlet from the left of v_j 's vertex gadget. If only one of these two outlets is an outer outlet, then attach the end of the first half to that outlet and the end of the second half to the other (middle) outlet. (If the outlets are both outer or both middle outlets, then it does not matter which end of the ladder path is connected to which outlet.) This is done by adding two disjoint edges which connect the two vertices forming an end of the ladder path to the two vertices forming the corresponding outlet as in Figure 10.

The value of k_1 is chosen as $k_1 := 6h(e)$ and the value of k_2 as follows. Define α to be 0, 4, or 8, if the ladder path is attached to the left, middle, or right outlet of v_i 's vertex gadget, respectively. Define β in the same manner for v_j . Set $k_2 := 18(j - i) - \alpha + \beta - 5$. Note that $k_2 \geq 5$ always holds. Finally, we will give the embedding of the ladder path's vertices, using the designations introduced in the definition of a ladder path. We only state the case where the first half is attached to v_i , for the other case the coordinates are to be mirrored at a suitable vertical axis. The positions of the vertices in the first half are (with $S_i^\alpha := 18i + \alpha$):

$$\begin{aligned}
 \text{emb}(u_r^1) &:= (S_i^\alpha, r + 2), & \text{emb}(\bar{u}_r^1) &:= (S_i^\alpha + 1, r + 2), \quad r = 0, \dots, k_1 - 1, \\
 \text{emb}(u_r^2) &:= (S_i^\alpha + 2 + r, k_1 + 1), & \text{emb}(\bar{u}_r^2) &:= (S_i^\alpha + 2 + r, k_1), \quad r = 0, \dots, k_2 - 1, \\
 \text{emb}(w_1) &:= (S_i^\alpha + k_2 + 2, k_1 + 2/3), & \text{emb}(w_2) &:= (S_i^\alpha + k_2 + 2, k_1 + 1/3), \\
 \text{emb}(w_3) &:= (S_i^\alpha + k_2 + 2.5, k_1 + 1/2), & \text{emb}(w_4) &:= (S_i^\alpha + k_2 + 3, k_1 + 2/3), \\
 \text{emb}(w_5) &:= (S_i^\alpha + k_2 + 3, k_1 + 1/3), & \text{emb}(w_6) &:= (S_i^\alpha + k_2 + 4, k_1 + 1), \\
 \text{emb}(w_7) &:= (S_i^\alpha + k_2 + 4, k_1).
 \end{aligned}$$

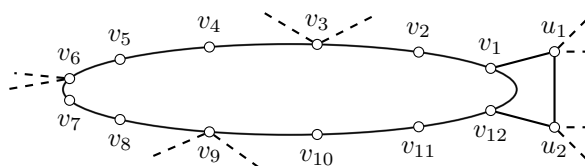
This embedding is illustrated in Figure 8. The positions of the vertices in the second half are analogous to the first. This step is illustrated in Figure 10.

Step 3 must be repeated for E_2 using negative y -coordinates. ┘

This construction is useful due to the following:

► **Lemma 5.5 (★).** *Gadget expansion preserves Hamiltonicity.*

The proof, which is deferred to the full version, is based on the fact that a Hamiltonian cycle may pass through vertex gadgets as pictured in Figure 9.



■ **Figure 11** An example of a permissible cycle addition with a cycle of length 12.

5.2 Face Filling

In order to turn the gadget expansion of a graph into an RNG and GG, we need to add *buffers*. The challenge is doing this in a way that preserves Hamiltonicity. We call an edge e of G *permissible* if G is not Hamiltonian or if G contains a Hamiltonian cycle that passes through e .

► **Lemma 5.6 (★)**. *Subdividing a permissible edge preserves Hamiltonicity. Moreover, both edges resulting from the subdivision are permissible in the resulting graph.*

Our main tool for adding buffers to a graph is called *permissible cycle addition*. Let $\{u_1, u_2\}$ be a permissible edge of $G = (V, E)$. We say that $G' = (V', E')$ is obtained from G by *attaching a permissible cycle to $\{u_1, u_2\}$* if (see Figure 11 for an illustration)

- $V' = V \uplus \{v_1, \dots, v_k\}$, $k \geq 4$, and v_1, \dots, v_k induce a cycle in that order, that is, $\{v_i, v_j\} \in E'$ if and only if $|i - j| = 1$ or $\{i, j\} = \{1, k\}$;
- $E' \cap \binom{V}{2} = E$;
- for all $i \in \{2, \dots, k - 1\}$, if $\deg_{G'}(v_i) \geq 3$, then $\deg_{G'}(v_{i-1}) = \deg_{G'}(v_{i+1}) = 2$; and
- $\deg_{G'}(v_1) = \deg_{G'}(v_k) = 3$, and $\{\{v_1, u_1\}, \{v_k, u_2\}\} \subseteq E'$.

Figure 11 pictures an example of such a cycle addition. This modification is useful due to:

► **Lemma 5.7 (★)**. *Permissible cycle addition preserves Hamiltonicity. Moreover, if v_1, \dots, v_k is the added cycle, then the edges $\{v_i, v_{i+1}\}$, $1 \leq i < k$, are all permissible in the resulting graph.*

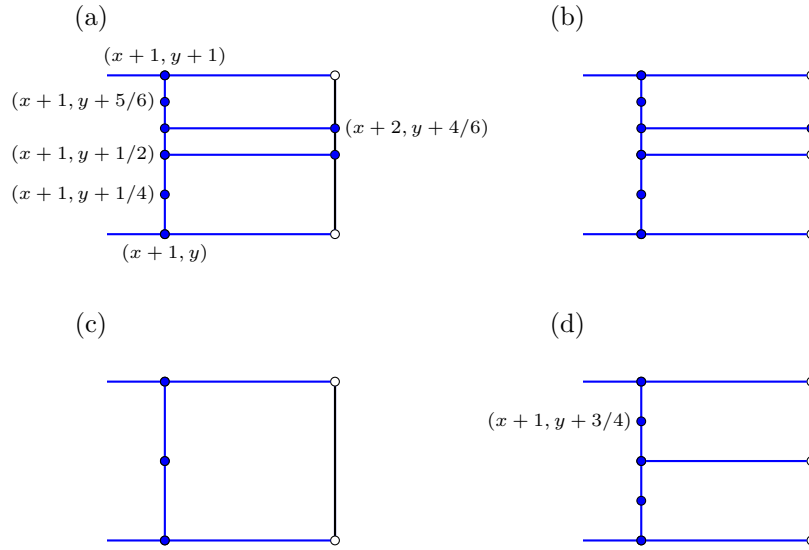
In order to be able to apply *permissible cycle addition* to the gadget expansion G' of a graph G , we need to know permissible edges of G' . For this, we have the following lemma.

► **Lemma 5.8 (★)**. *Let G be a 3-regular graph, G' the gadget expansion of G , and L any ladder path of G' whose first half is attached to an outer outlet (of a vertex gadget). Then, L contains two even inside and two even outside edges, all of which are permissible. Furthermore, these edges can be determined in linear time.*

We are set to give the construction in our polynomial-time many-one reduction from HC on 3-regular planar graphs to HC on RNGs or GGs.

► **Construction 5.9**. Let $G = (V, E)$ be a 3-regular planar graph. We will construct an RNG and GG $G' = (V', E')$ that is Hamiltonian if and only if G is. We will give the embedding of the vertices directly in the reduction.

We start with (G', emb) , the gadget expansion of G . We add one buffer for every $(x, y) \in \mathbb{Z}^2$ where $14 \leq x \leq 16n + 10$ and $-6h_2(G) - 2 \leq y \leq 6h_1(G) + 2$ are both even, except when G' already contains a vertex v with $\text{emb}(v) \in \{(x, y), (x + 1, y)\}$. The buffer then consists of a 4-cycle whose vertices are embedded at $(x, y), (x + 1, y), (x + 1, y + 1)$, and $(x, y + 1)$ and whose edges are then further subdivided. We call it the (x, y) -buffer and refer to the four (subdivided) edges as its *sides*. For each side, if G' previously already contained



■ **Figure 12** Construction of the sides of a cycle (left) with (a) a docking side adjoined to an edge, (b) a docking side adjoined to a previously existing non-docking side, (c) a non-docking side adjoined to an edge, and (d) a non-docking side adjoined to an existing side. In each picture, an edge or previously existing side to adjoin is on the right. The vertices and edges that result from the addition of the cycle are marked in blue while previously existing vertices and edges are in black.

a (possibly subdivided) edge running parallel to that side at distance 1, then we say that this side *adjoins* that (possibly subdivided) edge. For example, the side from $(x, y + 1)$ to $(x + 1, y + 1)$ would adjoin an existing edge from $(x, y + 2)$ to $(x + 1, y + 2)$. A side may also adjoin an edge in a switch to which it is not parallel. For example, the side from $(x, y + 1)$ to $(x + 1, y + 1)$ could adjoin an existing edge from $(x, y + 2)$ to $(x + 1, y + 2 + 1/3)$. Finally, exactly one of the four sides will be designated as the *docking side*. The docking side must adjoin either a side of a previously added buffer or a permissible edge of the gadget expansion.

When adding a buffer, if its sides adjoin existing sides or edges, then we also add edges connecting the buffer to other vertices and possibly also subdivide the adjoined edges or sides. There are four cases, depending on whether the newly added side is docking or non-docking and whether it adjoins a side of another buffer or an edge of the gadget expansion. These four cases are illustrated in Figure 12. In particular,

- the docking side of the added buffer is always subdivided four times and has four edges connecting it to the side or edge it adjoins (see Figure 12(a) and (b));
- a non-docking side adjoining an edge of the gadget expansion is subdivided once and has two connecting edges (see Figure 12(c));
- a non-docking side adjoining another buffer’s side is subdivided thrice and has three connecting edges (see Figure 12(d));
- a (non-docking) side which does not adjoin anything is subdivided once.

Figure 12 explains the positions of the sides’ subdivisions by way of an example for the right-side case. For the case of the other three sides, the coordinates are obtained by rotating around $(x + 1/2, y + 1/2)$.

The docking side must adjoin another buffer’s side or a permissible edge. We will now discuss a strategy to achieve this. First, observe that every position (x, y) at which we intend to add a buffer lies in a face of (G', emb) that borders more than four vertices (possibly the unbounded face). In order to distinguish between such faces and faces within ladder paths,

we will refer to the former as *regions*. We will add the buffers region-by-region. Next, note that once a buffer has been added to a region, then any subsequent buffer in that region can have its docking side adjoined to a (non-docking) side of another buffer added before it, as all edges on non-docking sides of a buffer are permissible by Lemma 5.7. (The gadget expansion is “surrounded” with buffers, so this works for the unbounded face.) Thus, it suffices to show how to add the first buffer for each region.

To this end, we must examine the structure of the gadget expansion. Let R be any region. Clearly, R borders some vertex gadget, and thus also a ladder path L attached to an outside outlet of that vertex gadget. More precisely, R borders either every inside or every outside edge of L . By construction of the gadget expansion, the first half of L is attached to an outside outlet of some vertex gadget. Thus, we can find two permissible edges of L that border R by Lemma 5.8. Because we have two permissible edges to choose from, we can ensure that we never adjoin the docking sides of two buffers to two “parallel” edges of L (i.e., to $\{u_i^j, u_{i+1}^j\}$ and $\{\tilde{u}_i^j, \tilde{u}_{i+1}^j\}$), as every ladder path only borders two regions. \square

We will use the following minor technical lemma.

► **Lemma 5.10 (★).** *Consider a square with corners $x_0, \dots, x_3 \in \mathbb{R}^2$ and $P \subseteq \mathbb{R}^2$. Let S_0, \dots, S_3 be the sides of the square, where x_i is incident to S_i and S_{i+1} (all indices are modulo 4). Let $y_i := \frac{x_{i-1} + x_i}{2}$ for each $i \in \{1, 2, 3\}$ and $y_0 := \frac{x_3 + x_0}{2}$. If P contains x_0, \dots, x_3 and possibly further points on the boundary of the square, but no points inside or outside of the square, then $\text{RNG}(P) = \text{GG}(P) = C_{|P|}$ if either of the following two conditions are met:*

- (i) $y_0, \dots, y_3 \in P$ or
- (ii) there is an $I \subseteq \{1, 2, 3, 4\}$ such that $|I| \geq 3$ and $S_i \cap P \subseteq \{x_0, x_1, x_2, x_3\}$ for each $i \in I$.

We are now prepared to prove the main result of this section.

Proof of Theorem 5.1. The proof builds on Construction 5.9. By Lemma 5.5, gadget expansion preserves Hamiltonicity. Each addition of a cycle involves subdividing a permissible edge (preserving Hamiltonicity by Lemma 5.6), and then adding a permissible cycle (preserving Hamiltonicity by Lemma 5.7). It follows that the construction preserves Hamiltonicity.

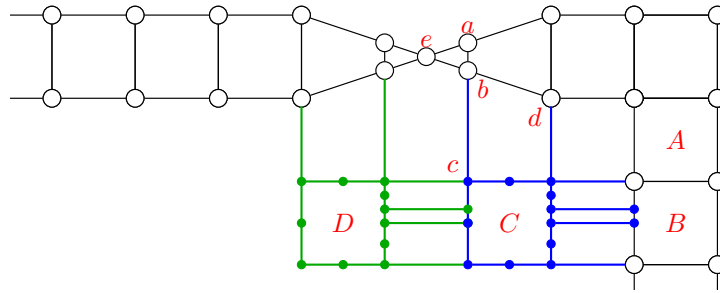
The construction already describes an embedding of the resulting graph G' . So it only remains to show that this embedding induces G' as its RNG and GG. Let

$$\mathcal{A} := \{(x, y) \in \mathbb{Z}^2 \mid 14 \leq x \leq 16n + 11, -6h_2(G) - 2 \leq y \leq 6h_1(G) + 3\}.$$

Note that for most $(x, y) \in \mathcal{A}$ there is a vertex embedded at (x, y) . The only exceptions are positions surrounding switches. For any $(x, y) \in \mathcal{A}$ we will call the vertices embedded at (x, y) , $(x + 1, y)$, $(x, y + 1)$, and $(x + 1, y + 1)$ along with any vertices embedded on the line segments between those four points a *grid face*. There are three classes of grid faces: grid faces within ladder paths or vertex gadgets, buffers, and grid faces between the aforementioned ones.

Within ladder paths, only two grid faces, which are shown in Figure 13 (A and B), can occur. For these, the claim is implied by Lemma 5.10(ii). Within buffers, more variations are possible (e.g. C and D in Figure 13). Here, the claim is implied by Lemma 5.10(i). All grid faces between cycles or between cycles and vertex gadgets are pictured in Figure 12. It is easy to see that all pairs of non-adjacent vertices have GG blockers and no pair of adjacent vertices has an RNG blocker.

We now consider the area surrounding a switch. This area is pictured in Figure 13. The vertex e is not a GG blocker for $\{a, b\}$, because $d(a, b)^2 = 1/9$ and $d(a, e)^2 = d(b, e)^2 = 5/18$. The vertex marked d is also not a GG blocker for $\{b, c\}$, since $d(b, c)^2 = 16/9$, $d(b, d)^2 = 10/9$, and $d(c, d)^2 = 2$. Other cases are analogous or easy to see. E.g., c and e are not adjacent, because b is a blocker. Vertices that do not share a grid face are not adjacent by Lemma 2.1.



■ **Figure 13** An excerpt of the graph G' produced by the reduction: Grid faces in G' : (A) Grid face within a ladder path or a vertex gadget with no docking side adjoined to its edges. (B) Grid face within a ladder path or a vertex gadget with a docking side adjoined to one of its edges. (C) + (D) Grid faces within buffers with the docking side on the right.

If n is the number of vertices in the input graph G , then the graph G' output by the construction contains n vertex gadgets each containing $\mathcal{O}(1)$ vertices, $\mathcal{O}(n)$ ladder paths with $\mathcal{O}(n)$ vertices, and $\mathcal{O}(n^2)$ cycles with $\mathcal{O}(1)$ vertices. It is easy to see that each step in both constructions can be computed in polynomial time. Along with Proposition 5.2, this implies that HC is NP-hard on RNGs and on GGs. Moreover, it also implies that HC cannot be decided by a $2^{o(n^{1/4})}$ -time algorithm on RNGs or GGs, unless the ETH fails. ◀

The computational complexity of HAMILTONIAN CYCLE on RCGs and that of HC on RNGs and GGs with maximum degree 3 is left open.

6 Dominating Set and Independent Set

In a graph $G = (V, E)$, an *independent set* is a vertex set $X \subseteq V$ such that $G[X]$ is edgeless. In the INDEPENDENT SET problem (IS), one is given a graph and $k \in \mathbb{N}_0$ and asked to decide whether the graph contains an independent set of size at least k . A *dominating set* in $G = (V, E)$ is a vertex set $X \subseteq V$ such that $N_G[X] = V$. In the DOMINATING SET problem (DS), one is given a graph and $k \in \mathbb{N}_0$ and asked to decide whether the graph contains a dominating set of size at most k . We also studied these two problems and proved:

► **Theorem 6.1 (★).** DOMINATING SET and INDEPENDENT SET on RCGs, on RNGs, and on GGs are NP-hard, even if the maximum degree is 4. Moreover, unless the ETH fails, neither problem admits a $2^{o(n^{1/4})}$ -time algorithm where n is the number of vertices.

The proof, which is based on reductions from the corresponding problems on planar graphs with maximum degree 3, is deferred to the full version. It remains open whether DS or IS can be solved in polynomial time when restricted to RCGs, RNGs, or GGs with maximum degree 3.

7 Conclusion

We have shown that problems that are NP-hard on planar graphs typically remain NP-hard on the three proximity graph classes we study. This suggests that the main tools of algorithm theory to attack these problems shall be parameterized and approximation algorithms. IS, DS, and FVS all admit polynomial-time approximation schemes on arbitrary planar

graphs, [2, 25] including the three types of proximity graphs we considered. FVS is fixed-parameter tractable on arbitrary graphs [5], while DS and IS are on planar graphs [1, 16]. We are not aware of any improvements (in terms of running time or approximation guarantees) to these results that are specific to RCGs, RNGs, or GGs.

It remains an important open question whether or not RCGs, RNGs, or GGs can be recognized in polynomial time and whether an embedding for a given graph can be computed in polynomial time [7, 8, 17]. If not, then one might suspect that the graph problems we have investigated might be easier if one is given an embedding rather than just the graph. Our reductions, however, prove that this is not the case, since we also give embeddings for the output graphs, which can easily be computed along with those output graphs.

We showed that FVS is NP-hard on proximity graphs with maximum degree 4, and it was already known to be polynomial-time solvable on any graph with maximum degree 3. For the other problems, we did not prove tight bounds on the maximum degree and this remains open. We proved ETH-based lower bounds of $2^{o(n^{1/4})}$ for each of these problems on RCGs, RNGs, and GGs (with the exceptions of 3-COL and HC on RCGs). On planar graphs in general, these problems can be solved in time $2^{\mathcal{O}(n^{1/2})}$ and this running time is optimal unless the ETH fails [28]. However, it might be possible to solve these problems on RCGs, RNGs, and GGs with a time bound strictly between $2^{o(n^{1/4})}$ and $2^{\mathcal{O}(n^{1/2})}$.

More generally, we are not aware of any problem that is known to be easier on the three graph classes we studied than on arbitrary planar graphs (excluding trivial cases like 3-COL on RCGs). Any such example would be of interest.

We conclude by remarking that most of the studied problems also remain hard on another canonical class of plane graphs, *Delaunay triangulations*. As the proofs work differently, we defer the specifics to subsequent publications.

References

- 1 Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002. doi:10.1007/s00453-001-0116-5.
- 2 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, January 1994. doi:10.1145/174644.174650.
- 3 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. Two-page book embeddings of 4-planar graphs. *Algorithmica*, 75(1):158–185, 2016. doi:10.1007/s00453-015-0016-8.
- 4 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry*. Springer, 2008. doi:10.1007/978-3-540-77974-2.
- 5 Hans L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5(1):59–68, 1994. doi:10.1142/S0129054194000049.
- 6 Prosenjit Bose, Vida Dujmović, Ferran Hurtado, John Iacono, Stefan Langerman, Henk Meijer, Vera Sacristán, Maria Saumell, and David R Wood. Proximity graphs: E , δ , Δ , χ and ω . *International Journal of Computational Geometry & Applications*, 22(05):439–469, 2012. doi:10.1142/S0218195912500112.
- 7 Prosenjit Bose, William Lenhart, and Giuseppe Liotta. Characterizing proximity trees. *Algorithmica*, 16(1):83–110, 1996. doi:10.1007/BF02086609.
- 8 Franz Brandenburg, David Eppstein, Michael T. Goodrich, Stephen Kobourov, Giuseppe Liotta, and Petra Mutzel. Selected open problems in graph drawing. In *Proceedings of the 11th International Symposium on Graph Drawing (GD)*, pages 515–539, 2004. doi:10.1007/978-3-540-24595-7_55.
- 9 Rowland Leonard Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941. doi:10.1017/S030500410002168X.

29:18 Most Classic Problems Remain NP-Hard on RNGs and their Relatives

- 10 Jean Cardinal, Sébastien Collette, and Stefan Langerman. Empty region graphs. *Computational Geometry*, 42(3):183–195, 2009. doi:10.1016/J.COMGEO.2008.09.003.
- 11 Robert Cimikowski. Coloring proximity graphs. In *Memoranda in Computer and Cognitive Science: Proceedings of the First Workshop on Proximity Graphs*, pages 141–156, 1989. doi:10.21236/ada237250.
- 12 Robert Cimikowski. Coloring certain proximity graphs. *Computers & Mathematics with Applications*, 20(3):69–82, 1990. doi:10.1016/0898-1221(90)90032-F.
- 13 Robert J Cimikowski. Properties of some Euclidean proximity graphs. *Pattern Recognition Letters*, 13(6):417–423, 1992. doi:10.1016/0167-8655(92)90048-5.
- 14 Reinhard Diestel. *Graph Theory*. Springer, 5th edition, 2016. doi:10.1007/978-3-662-53622-3.
- 15 Michael B. Dillencourt. Finding Hamiltonian cycles in Delaunay triangulations is NP-complete. *Discrete Applied Mathematics*, 64(3):207–217, 1996. doi:10.1016/0166-218X(94)00125-W.
- 16 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 17 Peter Eades and Sue Whitesides. Nearest neighbour graph realizability is NP-hard. In *Proceedings of the 2nd Latin American Symposium on Theoretical Informatics (LATIN)*, pages 245–256, 1995. doi:10.1007/3-540-59175-3_93.
- 18 K. Ruben Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Biology*, 18(3):259–278, 1969. doi:10.2307/2412323.
- 19 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.
- 20 M. R. Garey, D. S. Johnson, and R. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. doi:10.1137/0205049.
- 21 Branko Grünbaum. Grötzsch’s theorem on 3-colorings. *Michigan Mathematical Journal*, 10(3):303–310, 1963. doi:10.1307/mmj/1028998916.
- 22 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 23 J. W. Jaromczyk and M. Kowaluk. A note on relative neighborhood graphs. In *Proceedings of the 3rd Annual Symposium on Computational Geometry (SoCG)*, pages 233–241, 1987. doi:10.1145/41958.41983.
- 24 J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992. doi:10.1109/5.163414.
- 25 Jon Kleinberg and Amit Kumar. Wavelength conversion in optical networks. *Journal of Algorithms*, 38(1):25–50, 2001. doi:10.1006/jagm.2000.1137.
- 26 Pascal Kunz, Till Fluschnik, Rolf Niedermeier, and Malte Renken. Most classic problems remain NP-hard on relative neighborhood graphs and their relatives, 2021. doi:10.48550/ARXIV.2107.04321.
- 27 Philip M Lankford. Regionalization: Theory and alternative algorithms. *Geographical Analysis*, 1(2):196–212, 1969. doi:10.1111/j.1538-4632.1969.tb00615.x.
- 28 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 105:41–71, 2011. URL: <http://bulletin.eatcs.org/index.php/beatcs/article/view/92>.
- 29 László Lovász. Three short proofs in graph theory. *Journal of Combinatorial Theory Series B*, 19(3):269–271, 1975. doi:10.1016/0095-8956(75)90089-1.
- 30 David W. Matula and Robert R. Sokal. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12(3):205–222, 1980. doi:10.1111/j.1538-4632.1980.tb00031.x.
- 31 Joseph SB Mitchell and Wolfgang Mulzer. Proximity algorithms. In *Handbook of Discrete and Computational Geometry*, chapter 32, pages 849–874. Chapman and Hall/CRC, 3rd edition, 2017.

- 32 Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 571–575, 1996. doi:10.1145/237814.238005.
- 33 Ewald Speckenmeyer. *Untersuchungen zum Feedback Vertex Set Problem in ungerichteten Graphen* [Investigations into the Feedback Vertex Set problem in Undirected Graphs]. PhD thesis, Universität Paderborn, 1983.
- 34 Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980. doi:10.1016/0031-3203(80)90066-7.
- 35 Godfried T. Toussaint. Applications of the relative neighbourhood graph. *International Journal of Advances in Computer Science and Its Applications*, 4(3):77–85, 2014. URL: <http://journals.theired.org/journals/paper/details/4323.html>.
- 36 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988. doi:10.1016/0012-365X(88)90226-9.
- 37 Roderick B. Urquhart. Some properties of the planar Euclidean relative neighbourhood graph. *Pattern Recognition Letters*, 1(5):317–322, 1983. doi:10.1016/0167-8655(83)90070-3.

A Scalable Work Function Algorithm for the k -Server Problem

Sharath Raghvendra ✉

Department of Computer Science, Virginia Tech, Blacksburg, VA, USA

Rachita Sowle ✉

Department of Computer Science, Virginia Tech, Blacksburg, VA, USA

Abstract

We provide a novel implementation of the classical Work Function Algorithm (WFA) for the k -server problem. In our implementation, processing a request takes $O(n^2 + k^2)$ time per request; where n is the total number of requests and k is the total number of servers. All prior implementations take $\Omega(kn^2 + k^3)$ time per request. Previous approaches process a request by solving a min-cost flow problem. Instead, we show that processing a request can be reduced to an execution of the Dijkstra's shortest-path algorithm on a carefully computed weighted graph leading to the speed-up.

2012 ACM Subject Classification Theory of computation → K-server algorithms

Keywords and phrases k -server, Work Function Algorithm, Minimum-cost Flow

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.30

Supplementary Material *Software (Source Code)*: https://github.com/RachitaS/ScalableWorkFunction_Public; archived at `swh:1:dir:892a63bde62da09de7b9fb9b2086263cd9f8980b`

Funding Work on this paper was supported by NSF under grant CCF 1909171.

1 Introduction

In several applications such as emergency response, grocery delivery or virtual memory management, a new request has to be irrevocably assigned to a service provider in real-time. The k -server problem is a simplified abstraction of this problem. In this paper, we present a new implementation of the classical Work Function Algorithm for the k -server problem. We begin by introducing the k -server problem.

Problem Statement. Consider a vertex set V and a weighted complete graph where each edge $(u, v) \in V \times V$ has a cost $d(u, v)$. We assume that $d(\cdot, \cdot)$ is a metric. Given any two multi-sets A and B of points with $A, B \subseteq V$ and $|A| = |B|$, we use $d(A, B)$ to denote the minimum-cost bipartite matching of the points in A to points in B under the distance $d(\cdot, \cdot)$. For an integer $k > 0$, we are given k identical servers and their initial locations, also called the *initial configuration* $\mathcal{C}^0 = \{s_1^0, \dots, s_k^0\}$ in the metric space. A *configuration* is simply any multi-set $\mathcal{C} \subset V$, with $|\mathcal{C}| = k$. We use configurations to denote the locations of the k servers. For any request r_i , a configuration \mathcal{C} serves r_i if the location of r_i is contained in the multi-set \mathcal{C} . In other words, a server $s \in \mathcal{C}$ that is co-located with r_i serves r_i at zero cost. We are also given a sequence of n requests $R = \langle r_1, \dots, r_n \rangle$ that arrive over time with r_i arriving at time $t = i$. After r_i arrives, we move the servers to a configuration $\mathcal{C}^i = \{s_1^i, \dots, s_k^i\}$ that serves r_i . The input to the k -server problem is simply the initial configuration \mathcal{C}^0 and the request sequence R .

A *valid* solution to the problem is any sequence of configurations $\sigma = \langle \mathcal{C}^0, \mathcal{C}^1, \dots, \mathcal{C}^n, \mathcal{C}^{n+1} \rangle$ where $\forall 1 \leq i \leq n, \mathcal{C}^i$ serves request r_i . Note that, we set the *final configuration* \mathcal{C}^{n+1} to be the same as \mathcal{C}^n unless otherwise specified. Furthermore, we define the points within the final configuration as *anchor nodes*. The cost of σ is denoted by $w(\sigma)$ and $w(\sigma) = \sum_{i=0}^n d(\mathcal{C}^i, \mathcal{C}^{i+1})$.



© Sharath Raghvendra and Rachita Sowle;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 30; pp. 30:1–30:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *optimal solution*, denoted by $\sigma_{\mathcal{C}^0, R}^*$ is a valid solution with the smallest possible cost when the input is the initial configuration \mathcal{C}^0 and the request sequence is R . We denote $\sigma_{\mathcal{C}^0, R}^*$ as σ^* when the request sequence R and the initial configuration \mathcal{C}^0 are obvious from the context. Let R_i be the sequence of first i requests, i.e., $R_i = \langle r_1, \dots, r_i \rangle$. We use σ_i^* to denote $\sigma_{\mathcal{C}^0, R_i}^*$. Suppose, in addition to the initial configuration \mathcal{C}^0 and the request sequence R_i , the problem also specifies a final configuration \mathcal{C} , then $\sigma_i^*(\mathcal{C})$ denotes the minimum-cost solution that places the k servers in the initial configuration \mathcal{C}^0 after which it serves the i requests, and ends with \mathcal{C} as the final configuration.

In the k -server problem, when a request r_i arrives, one has to immediately and irrevocably commit to a configuration \mathcal{C}^i that serves request r_i . For any algorithm \mathcal{A} , let $\sigma_{\mathcal{A}} = \sigma_{\mathcal{A}, \mathcal{C}^0, R}$ be the sequence of configurations that \mathcal{A} chooses for an input request sequence R . Then, for a constant $\alpha > 0$, we say that \mathcal{A} has a competitive ratio of α if there exist another constant $\beta > 0$ such that, over all possible request sequences R , $w(\sigma_{\mathcal{A}, \mathcal{C}^0, R}) \leq \alpha w(\sigma_{\mathcal{C}^0, R}^*) + \beta$.

Next, we describe the work function algorithm for the k -server problem.

Work Function Algorithm. Given a request r_i , the work function algorithm chooses a configuration \mathcal{C}^i that serves request r_i as follows:

$$\mathcal{C}^i = \operatorname{argmin}_{\mathcal{C}} (w(\sigma_i^*(\mathcal{C})) + d(\mathcal{C}^{i-1}, \mathcal{C})). \quad (1)$$

Note that the minimum is over all possible configurations, i.e., every multi-set of size k . However, work function algorithm can be shown to be a lazy algorithm; see [8, 15], i.e., the configuration \mathcal{C}^i that minimizes (1) is obtained by choosing one server s^* in \mathcal{C}^{i-1} to serve request r_i where s^* is given by

$$s^* = \operatorname{argmin}_{s \in \mathcal{C}^{i-1}, \mathcal{C} = \mathcal{C}^{i-1} \setminus \{s\} \cup \{r_i\}} (w(\sigma_i^*(\mathcal{C})) + d(s, r_i)). \quad (2)$$

Prior Work. The k -server problem is central to the theory of online algorithms. For a survey of the problem, see [7]. The problem was first posted by Manasse et al. [11] who established a lower bound that as long as a metric space has $k + 1$ points, no deterministic algorithm can achieve a competitive ratio better than k . They also showed the competitive ratio of 2 for the 2-server problem. With this as evidence, they conjectured that in fact there is a k -competitive algorithm for this problem for any metric space. This conjecture is the celebrated *k -server conjecture*. Since then, the k -server conjecture has also been shown to be true for the line metric (1-dimensional Euclidean space) [2] and the tree metric [3]. It was shown that the Work Function Algorithm achieves a competitive ratio of $2k - 1$ on any metric space [8]. There has not been any significant progress on this conjecture since then. On the other hand, there has been substantial work on the randomized version of the k -server conjecture; see for instance Bubeck et al. [1] and Lee [10].

The analysis of the WFA in [8] was based on an exponential time dynamic programming implementation which processes the i th request r_i by solving equation (2).

The problem of finding the offline optimal solution for the k -server problem can be carefully modelled as a minimum-cost flow problem [2] where each edge has a unit capacity. Every unit of flow corresponds to a path taken by a server. In this flow network, every request is represented by two nodes connected by an edge of weight $-\infty$. This forces any minimum-cost flow to visit every request. The optimal solution to this flow network of $2n + k + 2$ nodes can be found in $O(n^2k)$ time.

For the online case, processing the i th request r_i requires solving equation (2). Similar to the offline case, evaluating (2) explicitly can be modelled as computing k distinct minimum-cost flow values, each of which can take $\Theta((i+k)^2k)$ time [2]. This observation leads to an $O((i+k)^2k^2)$ time algorithm for the WFA [16]. Using clever observations, evaluation of (2) can be reduced to computation of a single minimum-cost flow which takes $O(k(i+k)^2)$ time [15].

Rudec and Manger [17] presented an alternative approach for computing an offline solution to the k -server problem. Instead of creating a flow network with edges of $-\infty$ cost, they define a graph in the original metric space and define the notion of *regular flow* to be any flow in which each request is served by at least one server. One can move from any regular flow to another one using the so-called *up-down cycles*. Upon adding a new request, they show that finding the minimum-cost regular flow can be done by finding the most negative up-down cycle which they accomplish by conducting an exhaustive search. They argue that there is empirical benefit to this approach despite the worst-case execution time of this algorithm being slower than that of [15]. They also extend this approach to the work function algorithm.

Our Results and Approach. In this paper, we present a new implementation of the classical work function algorithm. Similar to Rudec and Manger [17], after processing each request, we maintain a valid solution that serves all the requests seen so far. Moreover, our algorithm processes the i th request by executing a single Dijkstra's shortest path search on a weighted graph in $O((i+k)^2)$ time which is faster than previous methods that take $\Omega(k(i+k)^2)$ time [15, 17].

For every server $s \in \mathcal{C}^{i-1}$, and $\mathcal{C} = \mathcal{C}^{i-1} \setminus \{s\} \cup \{r_i\}$, our algorithm computes (2) explicitly and then computes the minimum across all k choices of s . We show that the symmetric difference between $\sigma = \sigma_{i-1}^*(\mathcal{C}^{i-1})$ and $\sigma' = \sigma_i^*(\mathcal{C})$ is a trail¹ T whose edges alternate between those in σ and σ' . We refer to this as an *augmenting trail* and define its net-cost to be $w(\sigma') - w(\sigma)$. In order to find $w(\sigma')$, we simply have to identify the minimum net-cost augmenting trail that starts at r_i and ends at s . Our augmenting trails can be seen as a variant of the up-down cycles maintained by Rudec and Manger [17]. However, instead of conducting an exhaustive search, we describe an efficient algorithm (similar to the Kuhn-Munkres algorithm [9]) to find this minimum net-cost augmenting trail.

Using a graph search algorithm to find a minimum net-cost augmenting trail in the residual graph can be difficult since these algorithms find simple paths and not trails. In order to assist in the search of an augmenting trail, we define a weighted graph which we refer to as the *alternating graph*. Any augmenting trail in the residual graph maps to a directed path in the alternating graph and every directed path in the alternating graph corresponds to an alternating trail in the residual graph (See Lemma 2 and Figure 1).

Critically, we also store a set of weights on the vertices of the alternating graph. These weights satisfy a set of feasibility constraints, one for each edge in the alternating graph. Vertex weights have been used to speed-up computation for a shortest path in a graph with negative edge weights, for instance, in Johnson's algorithm [4]. These weights allow us to reduce the problem of finding minimum net-cost augmenting trail from r_i to every server $s \in \mathcal{C}^{i-1}$ to a single execution of Dijkstra's search procedure. Consequently, one can find the optimal choice in (2) in $O((i+k)^2)$ time. After the optimal choice is identified, we augment the solution to serve request r_i . This may create many new edges, delete existing edges and

¹ Recall that a trail is a (possibly non-simple) path that does not repeat edges.

also change the cost of some edges in the alternating graph. Somewhat surprisingly, despite the many updates to the alternating graph, we show that the vertex weights maintained by our algorithm continue to satisfy the feasibility constraints for all edges.

Significance of our Result. Minimum net-cost paths have been central to the design of algorithms for the closely related online minimum metric bipartite matching (OMBM) problem [6, 5, 13]. Unlike in the k -server problem where a server can serve any number of requests, in the OMBM problem, a server can serve no more than one request. Recent analysis of algorithms for the OMBM rely on the behavior of the vertex weights (also called *dual weights*) maintained while computing the minimum net-cost paths [12, 14]. Similarly, we hope that our formulation of the WFA as computing a minimum net-cost augmenting trail as well as the use of vertex weights can shed light into the dynamics of the WFA leading to an improved analysis.

2 Preliminaries

Recollect that a valid solution is provided by a sequence of configurations $\sigma = \langle \mathcal{C}^0, \mathcal{C}^1, \dots, \mathcal{C}^n, \mathcal{C}^{n+1} \rangle$. However, lazy valid solutions can also be represented as a set of k paths $\{\Gamma_1, \dots, \Gamma_k\}$ taken by each of the k servers. More precisely, for any $1 \leq i \leq n$ and $1 \leq j \leq k$, these paths satisfy the following properties:

(P1) For each server s_j , its path Γ_j starts at the location of s_j in the initial configuration.

After the first vertex, Γ_j consists of a sequence of requests served by s_j in increasing order of their arrival time. Finally, the last vertex of Γ_j is the location of s_j in the final configuration.

(P2) Every request in R_i participates in exactly one of the k paths.

Furthermore, it can be shown that any set of paths $\{\Gamma_1, \dots, \Gamma_k\}$ that satisfies (P1) and (P2) will be a valid solution .

The work function algorithm is a lazy algorithm. Therefore, we can represent the solution it produces as k paths satisfying (P1) and (P2). Next, we introduce the notations that are needed to describe an efficient implementation of the work function algorithm.

Notations. Throughout the rest of the paper, we consider directed graphs. We assume that any edge (u, v) is directed from u to v unless otherwise stated. Let σ_i be a valid solution to the first i requests of the k -server problem. Let $\{\Gamma_1, \Gamma_2, \dots, \Gamma_k\}$ be the set of k paths satisfying (P1) and (P2). For $1 \leq j \leq k$, and for any vertex v on a path Γ_j where v is not an anchor node, let $f(v)$ denote the location of the next vertex on the path. Similarly, for any vertex v on a path Γ_j where v is not a location in the initial configuration, let $p(v)$ denote the vertex that precedes v in the path Γ_j . In our algorithm, for any given valid solution σ_i , we create a directed graph G_i called the *residual graph* as follows. The vertex set V_i of G_i contains all vertices participating in any of the k paths, i.e., $V_i = \bigcup_{l=1}^k \left(\bigcup_{v \in \Gamma_l} v \right)$. There are two types of edges in the edge set E_i of G_i

- *Forward edges:* For each of the k paths and any vertex $v \in V$ where v is not an anchor node, we add a directed edge from v to $f(v)$ denoting that the server at v moves to $f(v)$.
- *Backward edges:* For every request r_j , we add a backward edge to $r_{j'}$, provided $j' < j$ and $j \neq f(j')$. This edge is directed from r_j to $r_{j'}$. We also add a backward edge directed from r_j to the k vertices of the initial configuration.

We refer to the residual graph with respect to σ_i as G_i . The k paths $\{\Gamma_1, \dots, \Gamma_k\}$ are represented as k directed paths consisting of all the forward edges in G_i . The backward edges, on the other hand, are not in the solution.

The set of all forward edges of a residual graph G_i corresponds to a valid solution if and only if

- (Q1) The forward edges are directed from an earlier request to a later request, and,
- (Q2) Every request r has exactly one incoming forward edge and one outgoing forward edge, every vertex from the initial configuration has one outgoing forward edge and every anchor node has one incoming forward edge.

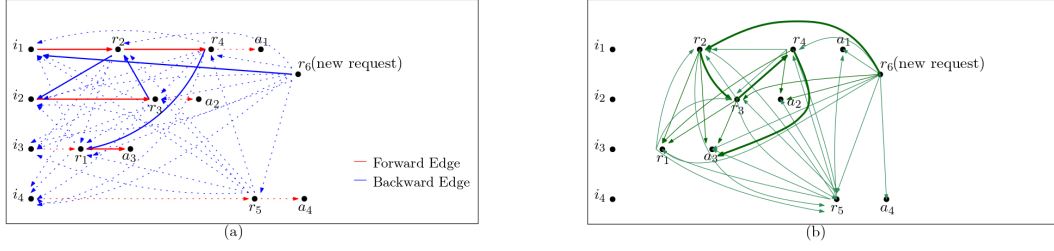
One can prove this by showing their equivalence to (P1) and (P2). Next, we define alternating and augmenting trails that play a critical role in processing a request.

Alternating Trails. Recollect that, in graph theory, a *trail* T is a path that is not necessarily a simple path but it does not repeat edges. We define an *alternating trail* T in G_i as a directed trail that alternates between forward and backward edges and ends at an anchor node.

When a new request r_{i+1} arrives, we include the request r_{i+1} and extend the residual graph to create an *extended* graph G_{i+1}^0 from G_i as follows. The new vertex set V_{i+1} is $V_i \cup \{r_{i+1}\}$. The edges incident on r_{i+1} are as follows: for each vertex $v \in V_i$, if v is not an anchor node, we add a backward edge directed from r_{i+1} to v . Figure 1(a) shows an example of an extended graph where $r_{i+1} = r_6$ with i_1, i_2, i_3, i_4 being the nodes in initial configuration and a_1, a_2, a_3, a_4 are the anchor nodes. Any alternating trail T in the extended graph G_{i+1}^0 that starts at r_{i+1} is an *augmenting trail*. Every edge going out of r_{i+1} in the extended graph G_{i+1}^0 is a backward edge. Therefore, an augmenting trail T starts with a backward edge and ends at an anchor node. For example, in Figure 1(a), $\langle r_6, i_1, r_2, i_2, r_3, r_2, r_4, r_1, a_3 \rangle$ is an augmenting trail.

Alternating Graph. Finding augmenting trails can be tricky. Typical graph search algorithms only find paths and not trails. In order to assist us in finding an augmenting trail efficiently, we define a different directed graph called the *alternating graph* for G_{i+1}^0 and denote it by $\mathcal{G}_{i+1}^0(\mathcal{V}_{i+1}^0, \mathcal{E}_{i+1}^0)$. Every directed simple path in this alternating graph \mathcal{G}_{i+1}^0 maps to a unique alternating trail in G_{i+1}^0 and every augmenting trail T in G_{i+1}^0 maps to a unique simple path in the alternating graph \mathcal{G}_{i+1}^0 which we refer to as the *augmenting path* (Lemma 2).

Thus, finding augmenting trails in G_{i+1}^0 reduces to finding augmenting paths in \mathcal{G}_{i+1}^0 which can be done via graph search algorithms. We describe the alternating graph next. The vertex set \mathcal{V}_{i+1}^0 of the alternating graph is the same as that of G_{i+1}^0 , i.e., $\mathcal{V}_{i+1}^0 = V_{i+1}$. The edge set of the alternating graph, \mathcal{E}_{i+1}^0 , is defined as follows. For every vertex v , if v has a backward edge to a node v' , then we add a directed edge from v to $f(v')$ in \mathcal{E}_{i+1}^0 . Figure 1(a) is an extended graph and Figure 1(b) is its alternating graph. For any directed edge $(v, f(v'))$ in \mathcal{G}_{i+1}^0 , denoted by $\text{PROJ}(v, f(v'))$ is the backward edge (v, v') concatenated with the forward edge $(v', f(v'))$, i.e., $\text{PROJ}(v, f(v')) = \langle (v, v'), (v', f(v')) \rangle$. For example, the projection of an edge (r_3, r_4) (Figure 1(b)) in the alternating graph consists of the edges $\langle (r_3, r_2), (r_2, r_4) \rangle$ (Figure 1(a)) of the residual graph. For any path P in the alternating graph, its projection is simply the concatenation of the projection of the individual edges. The highlighted augmenting path $\langle r_6, r_2, r_3, r_4, a_3 \rangle$ (Figure 1(b)) when projected gives the highlighted augmenting trail $\langle r_6, i_1, r_2, i_2, r_3, r_2, r_4, r_1, a_3 \rangle$ (Figure 1(a)). The construction of alternating graph and the definition of projection will also extend to the residual graph G_{i+1} in a straight-forward way. The alternating graph for G_{i+1} will be referred to as $\mathcal{G}_{i+1}(\mathcal{V}_{i+1}, \mathcal{E}_{i+1})$.



■ **Figure 1** Example of an (a) Extended graph G_6^0 and (b) its alternating graph \mathcal{G}_6^0 .

► **Lemma 1.** For any two edges (u, v) and (u', v') in \mathcal{G}_i^0 , their projections are edge-disjoint if and only if the head of both of the edges are distinct, i.e., $v \neq v'$.

Proof. The projection of (u, v) is $\langle (u, p(v)), (p(v), v) \rangle$ and the projection of (u', v') is $\langle (u', p(v')), (p(v'), v') \rangle$. Note that for every vertex s , there is a unique previous vertex $p(s)$. Therefore, if these projections are edge-disjoint and $(p(v), v)$ and $(p(v'), v')$ are distinct edges, then v and v' must be distinct points, i.e., $v \neq v'$. If $v \neq v'$, then $p(v) \neq p(v')$. Therefore, the forward edges $(p(v), v)$ and $(p(v'), v')$ are two vertex-disjoint edges implying that the projections of (u, v) and (u', v') are edge-disjoint. ◀

► **Lemma 2.** For every directed simple path P in the alternating graph \mathcal{G}_i^0 that ends at an anchor node, its projection $T = \text{PROJ}(P)$ is an alternating trail. Furthermore, for every alternating trail T in \mathcal{G}_i^0 where the first edge of T is a backward edge, there is a directed simple path P in \mathcal{G}_i^0 such that $\text{PROJ}(P) = T$.

Proof. The in-degree of any vertex on a simple directed path is at most one. Therefore, for any two edges (u, v) and (u', v') on a simple directed path P , $v \neq v'$. From Lemma 1, the projections of (u, v) and (u', v') are edge-disjoint. Therefore, the projection T of P which is simply the concatenation of projections of all the edges of P will be a path that does not repeat any edges, i.e., T is a trail. By construction, T starts with a backward edge, alternates between backward and forward edges, and ends at an anchor node, i.e., T is an alternating trail.

For any alternating trail T in \mathcal{G}_i^0 , let the backward edges be $(u_1, v_1), (u_2, v_2), \dots, (u_j, v_j)$ in the order in which they appear on the trail. Similarly let the forward edges be $(v_1, f(v_1)), (v_2, f(v_2)), \dots, (v_j, f(v_j))$, i.e., $T = \langle (u_1, v_1), (v_1, f(v_1)), (u_2, v_2), (v_2, f(v_2)), \dots, (u_j, v_j), (v_j, f(v_j)) \rangle$. By our assumption, the first edge of the alternating trail (u_1, v_1) must be a backward edge, and, $f(v_j)$ must be an anchor node and for $1 \leq t < j$, $f(v_t) = u_{t+1}$. For each $1 \leq t \leq j$, the pair of edges $(u_t, v_t)(v_t, f(v_t))$ is represented by a unique directed edge $(u_t, f(v_t))$ in \mathcal{G}_i^0 . We can therefore lift T to a path P in \mathcal{G}_i^0 by simply replacing successive pairs $(u_t, v_t)(v_t, f(v_t))$ with $(u_t, f(v_t))$. The resulting sequence of edges is $P = \langle (u_1, f(v_1)), (u_2, f(v_2)), \dots, (u_j, f(v_j)) \rangle$. Since for $1 \leq t < j$, $f(v_t) = u_{t+1}$, P is precisely the directed path $\langle u_1, u_2, \dots, u_j, f(v_j) \rangle$. Furthermore, since T is a trail and does not repeat any edges, for any two edges (u, v) and (u', v') in P its projections will be edge-disjoint. Therefore, from Lemma 1, $v \neq v'$ and so, P is a simple path. ◀

Augmentation. Consider any augmenting trail T in the extended graph G_{i+1}^0 that starts at r_{i+1} and ends at an anchor node a . We augment a valid solution σ_i (represented by the extended graph G_{i+1}^0) along an augmenting trail T to produce a solution σ_{i+1} and the residual graph G_{i+1} as follows:

To obtain the residual graph G_{i+1} from the extended graph G_{i+1}^0 , we can simply reverse the direction of all the edges on the augmenting trail T and relabel the forward edges as backward and all backward edges as forward. Finally, we remove the incoming forward edge to the anchor node a and add a new forward edge from r_{i+1} to the anchor node a and update the location of a to that of r_{i+1} .

Equivalently, one can consider modifying the k paths $\{\Gamma_1, \Gamma_2, \dots, \Gamma_k\}$ of σ_i by removing all forward edges of T and adding all backward edges of T to obtain the k paths $\{\Gamma'_1, \Gamma'_2, \dots, \Gamma'_k\}$ of σ_{i+1} . It can be shown that the solution σ_{i+1} is a valid solution that also serves request r_{i+1} . One can also generalize the augment operation for alternating trails and cycles. We refer to this generalized operation as the *flip* operation and use it in Section 4.2.

We define the net-cost of an augmenting trail T to be

$$\Phi(T) = \sum_{(u,v) \text{ is backward}} d(u,v) - \sum_{(u,v) \text{ is forward}} d(u,v).$$

Note that the net-cost of an augmenting trail T with respect to σ_i is the change in the cost due to augmenting σ_i along T . Therefore, one can express net-cost as $\Phi(T) = w(\sigma_{i+1}) - w(\sigma_i)$.

For any edge (u, v) in the alternating graph, we set its cost $c(u, v) = d(u, p(v)) - d(p(v), v)$ and for any simple path P , let $c(P) = \sum_{(u,v) \in P} c(u, v)$ denote its *net-cost*. Note that the cost of any edge in the alternating graph can be negative. For any simple augmenting path P in the alternating graph G_{i+1}^0 , its net-cost $c(P)$ is simply the net-cost of its projection $\Phi(\text{PROJ}(P))$.

► **Lemma 3.** *The net-cost of an augmenting path P in the alternating graph is equal to the net-cost of its projection.*

Proof. Given an augmenting path P in the alternating graph let the augmenting trail P' be its projection. Note that the set $B = \{(a, p(b)) \mid (a, b) \in P\}$ is the set of all backward edges of P' . Similarly, the set $F = \{(p(b), b) \mid (a, b) \in P\}$ is the set of all forward edges of P' . Therefore, the net-cost of P is $\sum_{(a,b) \in P} c(a, b) = \sum_{(a,b) \in P} (d(a, p(b)) - d(p(b), b)) = \sum_{(u,v) \in B} d(u, v) - \sum_{(u,v) \in F} d(u, v) = \Phi(P')$. ◀

Let $y(\cdot)$ be a weight associated with every vertex of the alternating graph. We say that any valid solution σ_i and the weight function $y(\cdot)$ is *feasible* if for any edge (a, b) directed from a to b

$$y(a) - y(b) \leq c(a, b). \quad (3)$$

We say that any edge satisfying this inequality is *feasible*. We define *slack* of any edge (a, b) directed from a to b to be $c(a, b) + y(b) - y(a)$ and denote it by $s(a, b)$. Given these notations, we are ready to describe our algorithm.

3 The Algorithm

After processing i requests, our algorithm will maintain a feasible valid solution $\sigma = \sigma_i$. We refer to this as the *offline* solution. Initially, the weight $y(v)$ for every vertex $v \in \mathcal{C}^0$ is set to 0 and the offline solution σ is empty. For $i \geq 0$, using the alternating graph G_{i+1}^0 , our algorithm will identify an appropriate augmenting trail T in the extended graph G_{i+1}^0 . Recollect that T ends at an anchor node a . The algorithm then moves the server located at

a and that served request $p(a)$ to serve request r_{i+1} . The offline solution σ is updated by augmenting σ_i along T leading to a valid solution $\sigma = \sigma_{i+1}$. The algorithm consists of four steps²:

- (1) *Augmenting path search*: Let G' be identical to this alternating graph \mathcal{G}_{i+1}^0 except the cost of any edge (a, b) is replaced by its slack $s(a, b)$. Note that G' is a graph with only non negative edge-costs. The algorithm executes Dijkstra's algorithm on G' with r_{i+1} as the source. Dijkstra's algorithm returns the shortest path from r_{i+1} to every other vertex in \mathcal{V}_{i+1}^0 . Let, for any vertex $v \in \mathcal{V}_{i+1}^0$, ℓ_v be its shortest path cost as returned by Dijkstra's algorithm from the source r_{i+1} .
- (2) *Determine net-cost*: Next, we compute the minimum net-cost augmenting path from r_{i+1} to each of the k anchor nodes $\{a_1, \dots, a_k\}$. For any anchor node $a_j \in \{a_1, \dots, a_k\}$, we set the minimum net-cost to be $\Phi_j = \ell_{a_j} - y(a_j)$ and the path P_j corresponding to this net-cost is the shortest path from r_{i+1} to a_j in G' as returned by Dijkstra's algorithm (Step 1).
- (3) *Choose server*: Let $a_m = \arg \min_{a_j \in \{a_1, \dots, a_k\}} (d(a_j, r_{i+1}) + \Phi_j)$ and let s be the server located at a_m with $p(a_m)$ as its last served request. We assign s to serve request r_{i+1} .
- (4) *Update offline solution*: We update the offline solution as follows: **(a)** For any vertex $v \in V_{i+1}$, if $\ell_v < \ell_{a_m}$, we set its weight $y(v) \leftarrow y(v) + \ell_{a_m} - \ell_v$. **(b)** After updating the weights, we augment σ_i along P_{a_m} to obtain $\sigma = \sigma_{i+1}$ and update the edges of the alternating graph to reflect the new solution σ . We also set $y(a_m) \leftarrow 0$.

Our algorithm maintains the following two invariants at all times:

- (I1): The offline solution σ along with the weights $y(\cdot)$ is a valid and feasible solution, and,
- (I2): Let \mathcal{C}^i be the final configuration of σ_i . Then, $\sigma_i = \sigma_i^*(\mathcal{C}^i)$. Furthermore, for every anchor node $a_j \in \{a_1, \dots, a_k\}$, let $\mathcal{C}_j^{i+1} = \mathcal{C}^i \setminus \{a_j\} \cup \{r_{i+1}\}$. Then $\Phi_j = w(\sigma_{i+1}^*(\mathcal{C}_j^{i+1})) - w(\sigma_i^*(\mathcal{C}^i))$.

The proofs of these invariants are given in Section 4. Note that, after each request is processed the set of edges in the alternating graph can change substantially. Despite this, our weight updates guarantee that every newly added edge in the alternating graph continues to be feasible (Section 4.1).

Efficiency. Note that the $|V_{i+1}| = i + 1 + 2k$ and $|E_{i+1}| = O((i + k)^2)$. The extended graph and alternating graphs also have identical bounds. Step 1 of the algorithm requires computation of G' and an execution of Dijkstra's algorithm on G' which takes $O((i + k)^2)$ time. Step 2 of the algorithm requires constant time computation for each of the anchor nodes and therefore takes $O(k)$ time. The paths P_j computed in Step 2 is can be compactly represented using the shortest path tree that is returned by Dijkstra's algorithm. Therefore, computing P_j does not require any additional time. Choosing the server in Step 3 can be performed by simply accessing the cost between the r_{i+1} and each of the k servers and computing the one that minimizes $\Phi_j + d(a_j, r_{i+1})$. Step 3, therefore, takes only $O(k)$ time. Step 4(a) requires us to update the weight at each vertex which can be done in $O(i + k)$ time. Step 4(b) requires augmenting and updating the residual and alternating graphs each of which can be performed in $O((i + k)^2)$. Therefore, the time taken to process each request is dominated by $O(i^2 + k^2) = O(n^2)$.

Next, assuming the invariants hold, we will show that the algorithm picks the same server as the Work Function Algorithm.

² An implementation of this algorithm is available here: https://github.com/RachitaS/ScalableWorkFunction_Public

Correctness: The following lemma establishes a link between the net-cost of an augmenting path and the sum of the slacks along its edges.

► **Lemma 4.** *Suppose σ_i and the weights $y(\cdot)$ form a feasible solution. For any augmenting path P in the alternating graph that starts at r_{i+1} and ends at an anchor node a , its net-cost is*

$$\Phi(P) = y(r_{i+1}) - y(a) + \sum_{(u,v) \in P} s(u,v). \quad (4)$$

Proof. Every vertex $v' \in P$ with the exception of the first vertex r_{i+1} and the last vertex a will have an incoming edge (u', v') and an outgoing edge (v', w') in P . The weight of v' , $y(v')$ is added with respect to (v', w') and subtracted with respect to the edge (u', v') and therefore, the net-contribution of v' to Equation (4) is zero. The first vertex r_{i+1} participates in the first edge of P and contributes $+y(r_{i+1})$ to Equation (4). The last vertex a participates only in the last edge and contributes $-y(a)$ to Equation (4). ◀

From Invariant (I1), Equation 4 and since $y(r_{i+1}) = 0$, the minimum net-cost path P_j from r_{i+1} to some anchor node a_j in the alternating graph \mathcal{G}_{i+1}^0 is also the augmenting path that minimizes the sum of slacks along its edges. From invariant (I1) all slacks are non-negative and so, P_j will be the augmenting path returned by the execution of Dijkstra's algorithm in Step 1 of the algorithm. Furthermore, $\ell_{a_j} = \sum_{(u,v) \in P_j} s(u,v)$. Therefore, from Equation 4, we conclude that $\Phi_j = \Phi(P_j) = \ell_{a_j} - y(a_j)$. Therefore, Step 2 of the algorithm will correctly compute the minimum net-cost augmenting path to every anchor node $a_j \in \{a_1, \dots, a_k\}$.

Step 3 of the algorithm selects the server located at the anchor node $a_m = \operatorname{argmin}_{a_j \in \{a_1, \dots, a_k\}} (d(a_j, r_{i+1}) + \Phi_j)$. Let $X_j = \mathcal{C}_j^{i+1} = \mathcal{C}^i \setminus \{a_j\} \cup \{r_{i+1}\}$. By Invariant (I2), $(d(a_j, r_{i+1}) + \Phi_j) = d(a_j, r_{i+1}) + w(\sigma_{i+1}^*(X_j)) - w(\sigma_i^*(\mathcal{C}^i))$ and, $\operatorname{argmin}_{a_j \in \{a_1, \dots, a_k\}} (d(a_j, r_{i+1}) + \Phi_j)$ is

$$\begin{aligned} &= \operatorname{argmin}_{j \in \{1, \dots, k\}} (d(a_j, r_{i+1}) + w(\sigma_{i+1}^*(X_j)) - w(\sigma_i^*(\mathcal{C}^i))) \\ &= \operatorname{argmin}_{j \in \{1, \dots, k\}} (d(a_j, r_{i+1}) + w(\sigma_{i+1}^*(X_j))). \end{aligned}$$

The last equality follows from the fact that $w(\sigma_i^*(\mathcal{C}^i))$ is the same for every choice of j . Thus, we choose the same server as required by the work function algorithm.

4 Proof of Invariants

4.1 Proof of Invariant (I1)

Recall the definition of feasibility. Any valid solution σ_i with the weight function $y(\cdot)$ associated with each vertex of its alternating graph is *feasible* if for every edge (a, b) directed from a to b in its alternation graph satisfies the following equation.

$$y(a) - y(b) \leq c(a, b). \quad (5)$$

Furthermore, any edge satisfying the above inequality is *feasible*.

We prove a slightly stronger version of Invariant (I1)

30:10 A Scalable Work Function Algorithm for the k -Server Problem

(I1): The offline solution σ along with the weights $y(\cdot)$ maintained by the algorithm is a valid and feasible solution. Furthermore, for any forward edge (u, v) in the residual graph of σ ,

$$y(v) \geq d(u, v). \quad (6)$$

We can prove this invariant by induction. At time $t = 0$, we have the servers in the initial configuration. The vertex set of the initial residual graph G_0 only contains the vertices for the initial configuration and the anchor nodes. The edge set of the initial residual graph G_0 contains only forward edges directed from each server in the initial configuration to its anchor node. Since there are no backward edges in G_0 , the alternating graph \mathcal{G}_0 does not have edges. Therefore, σ_0 and the vertex weights $y(\cdot)$ are trivially feasible.

Let the solution σ_i after serving i requests be a valid and feasible solution and let $y(\cdot)$ at the end of processing request r_i satisfy (6). Given this, we will now prove that the solution σ_{i+1} is valid and feasible and the updated vertex weight satisfies (6). Note that each new request arrives with a default weight 0. Given a valid feasible solution σ_i along with the vertex weight function $y(\cdot)$, the algorithm first constructs the extended graph G_{i+1}^0 . Lemma 6 shows that the corresponding alternating graph \mathcal{G}_{i+1}^0 with weight function $y(\cdot)$ continues to be feasible. Steps 1, 2 and 3 do not modify σ_i or the vertex weights. Therefore, σ_i continues to remain valid and feasible till the end of Step 3.

In Step 4(a) of the algorithm, the weights on the vertices of \mathcal{G}_{i+1}^0 are updated. Let $y(\cdot)$ be the weights prior to executing step 4(a) and let $y'(\cdot)$ be the weights after executing step 4(a). Lemma 8 proves that \mathcal{G}_{i+1}^0 along with the updated vertex weights $y'(\cdot)$ remain feasible. In Step 4(b), the algorithm augments the solution σ_i along the augmenting trail T (as determined in Step 3) leading to σ_{i+1} . Lemma 10 shows that the solution σ_{i+1} remains valid and feasible. Finally, in Lemma 11, we argue that the vertex weights at the end of Step 4(b) satisfies (6).

► **Lemma 5.** *Suppose P is the augmenting path from r_{i+1} to the anchor node a_m chosen in step 3 of the algorithm, then the slack $s(u, v)$ on every edge (u, v) of P after the vertex weight update in step 4(a) is zero.*

Proof. Let $y(\cdot)$ be the vertex weight prior to Step 4(a) and $y'(\cdot)$ be the vertex weight after Step 4(a). By our choice in Step 3, P is the shortest path computed by Dijkstra's algorithm from r_{i+1} to a_m in G' . Since ℓ_{a_m} is the cost of P in G' , every vertex $v \in P$ has a shortest path cost at most ℓ_{a_m} , i.e., $\ell_v \leq \ell_{a_m}$. Furthermore, by the optimal sub-structure property of shortest paths, for any edge $(u, v) \in P$, $\ell_v - \ell_u = s(u, v) = c(u, v) + y(v) - y(u)$ or $\ell_v - \ell_u = c(u, v) + y'(v) - y'(u)$.

$$\begin{aligned} c(u, v) + (y(v) - \ell_v) - (y(u) - \ell_u) &= 0, \\ c(u, v) + (y(v) - \ell_v + \ell_{a_m}) - (y(u) - \ell_u + \ell_{a_m}) &= 0, \\ c(u, v) + y'(v) - y'(u) &= 0. \end{aligned}$$

The second to last equality is obtained by simply adding and subtracting ℓ_{a_m} to the LHS. The last equality follows from the fact that $\ell_v \leq \ell_{a_m}$ and $\ell_u \leq \ell_{a_m}$ and the update of the vertex weights defined in Step 4(a). ◀

► **Lemma 6.** *Given a valid feasible solution σ_i , all the edges of the alternating graph \mathcal{G}_{i+1}^0 are feasible.*

Proof. From the inductive hypothesis, σ_i is a feasible solution i.e. the edges of \mathcal{G}_i satisfy (5), and, the weights $y(\cdot)$ satisfies (6). The extended graph G_{i+1}^0 is created by the addition of r_{i+1} to the vertex set of G_i in the vertex set. Furthermore, for every $j \leq i$ a backward edge is added from r_{i+1} to r_j and an edge $(r_{i+1}, f(r_j))$ is added to \mathcal{G}_{i+1}^0 . We show that for every such edge, $(r_{i+1}, f(r_j)) \in \mathcal{G}_{i+1}^0$, the feasibility condition (5) holds. The cost of the edge $(r_{i+1}, f(r_j))$ is

$$c(r_{i+1}, f(r_j)) = d(r_{i+1}, r_j) - d(r_j, f(r_j)) \quad (7)$$

$$\geq -d(r_j, f(r_j)). \quad (8)$$

Note that the vertex weight of r_{i+1} , $y(r_{i+1})$ is set to 0. By the inductive hypothesis, $-y(f(r_j)) \leq -d(r_j, f(r_j))$. Adding $y(r_{i+1})$ to the LHS and 0 to the RHS, we get $y(r_{i+1}) - y(f(r_j)) \leq -d(r_j, f(r_j))$ or $y(r_{i+1}) - y(f(r_j)) \leq c(r_{i+1}, f(r_j))$. The last inequality follows from 8. This implies that the edge (r_{i+1}, v) satisfies (5). ◀

Feasibility after Step 4(b). Step 4(b) in the algorithm augments the solution along an augmenting trail T . In doing so, the alternating graph \mathcal{G}_{i+1}^0 is updated to \mathcal{G}_{i+1} . The edges \mathcal{E}_{i+1} of \mathcal{G}_{i+1} may include several new edges that were not in \mathcal{G}_{i+1}^0 . Furthermore, there may be edges whose costs $c(\cdot, \cdot)$ change due to augmentation. We classify all such edges in the alternating graph \mathcal{G}_{i+1} as *affected* edges. The following two lemmas establishes important properties of the affected edges.

► **Lemma 7.** *Let P be the augmenting path from r_{i+1} to an anchor node a_m in the alternating graph \mathcal{G}_{i+1}^0 that is computed in Step 3 of our algorithm. Given any affected edge (u, v) in \mathcal{G}_{i+1} , let $\langle (u, x), (x, v) \rangle$ be its projection. Then, $v \neq a_m$ and (v, x) is a backward edge on the augmenting trail $\text{PROJ}(P)$.*

Proof. Let P be the augmenting path from r_{i+1} to an anchor node a_m in the alternating graph \mathcal{G}_{i+1}^0 that is computed in Step 3 of our algorithm. And let T be the augmenting trail in G_{i+1}^0 such that $T = \text{PROJ}(P)$.

First we will prove that given any affected edge (u, v) in \mathcal{G}_{i+1} , $v \neq a_m$. After augmentation along T , we add the forward edge (r_{i+1}, a_m) . Since all backward edges are directed from a later request to an earlier request and since r_{i+1} is the latest request in the residual graph G_{i+1} , there are no in-coming backward edges to r_{i+1} . Therefore, by its description, there will not be any incoming edges to a_m in the alternating graph \mathcal{G}_{i+1} and so $v \neq a_m$.

Since (u, v) is an affected edge, at least one of the edges in its projection $\langle (u, x), (x, v) \rangle$ is newly introduced by the augment operation. We claim that the forward edge (x, v) was added by the augment operation in Step 4(b) of the algorithm. Suppose, for the sake of contradiction, (x, v) was not added in Step 4(b), i.e., (x, v) is a forward edge in G_{i+1}^0 . Therefore, (u, x) must be the backward edge that was newly added by the augment operation. We can conclude that the augmenting trail T contains the forward edge (x, u) , implying (x, u) is a forward edge in G_{i+1}^0 . Note that (x, u) and (x, v) are both forward edges in G_{i+1}^0 which contradicts the fact that σ_i is a valid solution. Therefore, we conclude that the forward edge (x, v) was introduced by the augment operation in Step 4(b), i.e., (v, x) is a backward edge in the augmenting trail T . ◀

► **Lemma 8.** *The edges of the alternating graph \mathcal{G}_{i+1}^0 remains feasible after the vertex weight update in step 4(a).*

Proof. The alternating graph \mathcal{G}_{i+1}^0 before the execution of Step 4(a) is feasible. Let $y(\cdot)$ (resp. $y'(\cdot)$) denote the vertex weights before (resp. after) the weight update of step 4(a). For each edge $(u, v) \in \mathcal{G}_{i+1}^0$, let $s(u, v)$ (resp. $s'(u, v)$) represent the slack on (u, v) before (resp. after) weight update in step 4(a). Let (u, v) be any directed edge in \mathcal{G}_{i+1}^0 . Note that every edge in \mathcal{G}_{i+1}^0 along with the weights $y(\cdot)$ satisfies (3) and so the slack $s(u, v) \geq 0$. Let a_m be the anchor node chosen by algorithm in Step 3 and for any vertex $w \in \mathcal{G}_{i+1}^0$, recollect that ℓ_w is the shortest path cost returned by Dijkstra's algorithm in Step 1 of the algorithm. For the edge (u, v) , there are four possibilities after step 4(a):

Case (i) $\ell_v \geq \ell_{a_m}$ and $\ell_u \geq \ell_{a_m}$. In this case, Step 4(a) will not update the vertex weights for u and v . So, $y'(u) = y(u)$, $y'(v) = y(v)$, and $s'(u, v) = s(u, v)$. Therefore, (u, v) remains feasible with respect to $y'(\cdot)$.

Case (ii) $\ell_v < \ell_{a_m}$ and $\ell_u \geq \ell_{a_m}$. In this case, Step 4(a) will update the vertex weights to $y'(v) = y(v) + (\ell_{a_m} - \ell_v) > y(v)$ and $y'(u) = y(u)$. Furthermore, from feasibility of (u, v) with respect to $y(\cdot)$, we have $s(u, v) \geq 0$. Therefore, $s'(u, v) = c(u, v) - y'(u) + y'(v) \geq c(u, v) - y(u) - y(v) \geq 0$ implying that (u, v) remains feasible with respect to the updated vertex weight $y'(\cdot)$.

Case (iii) $\ell_u < \ell_{a_m}$ and $\ell_v \geq \ell_{a_m}$. In this case, Step 4(a) updates the vertex weight to $y'(u) = y(u) + (\ell_{a_m} - \ell_u)$ and $y'(v) = y(v)$. From the property of shortest path distances, the shortest path distance from r_{i+1} to v is bounded by the shortest path distance from r_{i+1} to u and the slack of the edge from (u, v) , i.e., $\ell_v - \ell_u \leq s(u, v)$. Using the definition of slack we get,

$$\begin{aligned} \ell_v - \ell_u &\leq c(u, v) - y(u) + y(v), \\ (\ell_{a_m} - \ell_u) - (\ell_{a_m} - \ell_v) &\leq c(u, v) - y(u) + y(v), \\ [(\ell_{a_m} - \ell_u) + y(u) - \\ &((\ell_{a_m} - \ell_v) + y(v))] \leq c(u, v), \\ y'(u) - ((\ell_{a_m} - \ell_v) + y'(v)) &\leq c(u, v) \end{aligned}$$

The last inequality follows from the fact that $y'(u) = (\ell_{a_m} - \ell_u) + y(u)$. Furthermore, since $\ell_{a_m} < \ell_v$, we get $y'(u) - y'(v) \leq c(u, v)$, i.e., the edge (u, v) remains feasible.

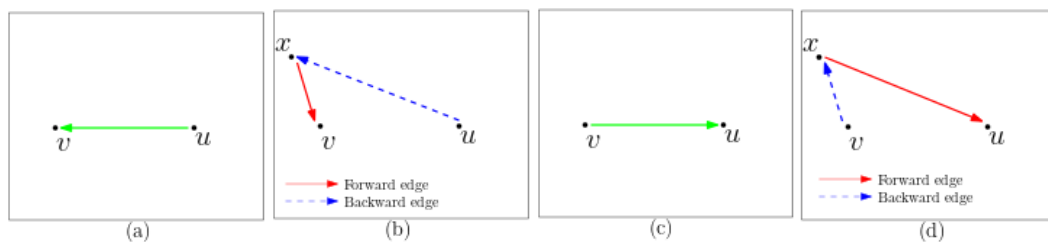
Case (iv) $\ell_u < \ell_{a_m}$ and $\ell_v < \ell_{a_m}$. In this case, Step 4(a) sets $y'(u) = y(u) + (\ell_{a_m} - \ell_u)$ and $y'(v) = y(v) + (\ell_{a_m} - \ell_v)$. Again, the shortest path from r_{i+1} to v is of cost bounded by the shortest path from r_{i+1} to u and the slack on the edge (u, v) . Therefore, $\ell_v \leq \ell_u + s(u, v)$ i.e. $\ell_v - \ell_u \leq s(u, v)$. Using the definition of slack we get

$$\begin{aligned} \ell_v - \ell_u &\leq c(u, v) - y(u) + y(v), \\ (\ell_{a_m} - \ell_u) - (\ell_{a_m} - \ell_v) &\leq c(u, v) - y(u) + y(v), \\ [(\ell_{a_m} - \ell_u) + y(u) - \\ &((\ell_{a_m} - \ell_v) + y(v))] \leq c(u, v), \\ y'(u) - y'(v) &\leq c(u, v). \end{aligned}$$

implying (u, v) is feasible with respect to $y'(\cdot)$. ◀

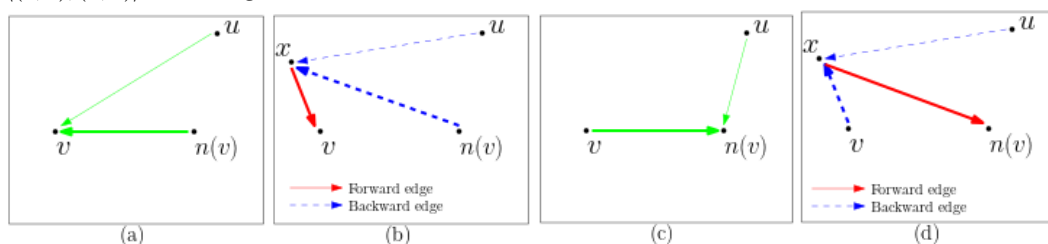
► **Lemma 9.** Let P be the augmenting path computed in Step 3 that goes from request r_{i+1} to an anchor node a_m in the alternating graph \mathcal{G}_{i+1}^0 . For any vertex v on the path P where $v \neq a_m$, let $n(v)$ denote the vertex that succeeds v on P . Given any affected edge (u, v) in \mathcal{G}_{i+1} , let $\langle (u, x), (x, v) \rangle$ be its projection. Then,

- (i) Either the edge (v, u) is on the augmenting path P with $\text{PROJ}(v, u) = \langle (v, x), (x, u) \rangle$, or
- (ii) There is an edge $(u, n(v))$ in \mathcal{G}_{i+1}^0 with its projection $\text{PROJ}(u, n(v)) = \langle (u, x), (x, n(v)) \rangle$.



(I) Demonstrating Case(a): (u, v) is an affected edge with projection $\langle(u, x), (x, v)\rangle$ such that (x, u) was a forward edge prior to augmentation.

(a) Affected edge $(u, v) \in \mathcal{G}_{i+1}$. (b) Projection of the affected edge (u, v) , $\text{PROJ}(u, v) = \langle(u, x), (x, v)\rangle$. (c) Alternating graph edge $(v, u) \in \mathcal{G}_{i+1}^0$ before augmentation. (d) Projection of the edge $(v, u) = \langle(v, x), (x, u)\rangle$ before augmentation.



(II) Demonstrating Case(b): (u, v) is an affected edge with projection $\langle(u, x), (x, v)\rangle$ such that (u, x) was a backward edge prior to augmentation as well.

(a) Affected edge $(u, v) \in \mathcal{G}_{i+1}$. (b) Projection of the affected edge $(u, v) \in \mathcal{G}_{i+1}$, $\langle(u, x), (x, v)\rangle$. (c) Alternating graph edges of \mathcal{G}_{i+1}^0 before augmentation where $(v, n(v)) \in P$ and $(u, n(v)) \in \mathcal{G}_{i+1}^0$. (d) Projection of the edge $(u, n(v)) \in \mathcal{G}_{i+1}^0$, $\langle(u, x), (x, n(v))\rangle$ before augmentation.

■ **Figure 2**

Proof. From Lemma 7, we know that (v, x) is a backward edge in the augmenting trail T . There are two possibilities for the edge (u, x) : (a) (u, x) was also added as a backward edge by the augment process, or (b) (u, x) was also an edge in the extended graph G_{i+1}^0 .

For case (a), the edge (x, u) was a forward edge prior to augmentation. Since both the backward edge (v, x) and the forward edge (x, u) are in the augmenting trail T , we will have an edge (v, u) in the augmenting path P , implying (i). An instance of this case is shown in Figure 2I where directed edge (u, v) is the affected edge.

In case (b), (u, x) is also a backward edge in G_{i+1}^0 (Figure 2II(b)). Since (v, x) is a backward edge in T , the projection of the edge $(v, n(v))$ will contain the backward edge (v, x) followed by the forward edge $(x, n(v))$. Since (u, x) is a backward edge and $(x, n(v))$ is a forward edge in G_{i+1}^0 , we will have an edge $(u, n(v))$ in the alternating graph \mathcal{G}_{i+1}^0 . An example of this case is demonstrated in Figure 2II where the directed edge (u, v) in Figure 2II(a) is the affected edge. Figure 2II(b) shows the projection of (u, v) . Figure 2II(c) shows the scenario before augmentation where $(v, u) \in \mathcal{G}_{i+1}^0$ and Figure 2II(d) shows (v, u) 's projection in G_{i+1}^0 . ◀

► **Lemma 10.** σ_{i+1} is a valid and feasible solution after the Augment operation in Step 4(b).

Proof. Let P be the augmenting path from r_{i+1} to the anchor node chosen in Step 3. To prove that σ_{i+1} is a feasible solution after Step 4(b), we need to show that the edges of alternating graph \mathcal{G}_{i+1} that are affected by the augment operation along the path P continue to be feasible and satisfy (5).

Let $c(\cdot, \cdot)$ be the cost function of edges in the alternating graph \mathcal{G}_{i+1}^0 , i.e., prior to augmentation and let $c'(\cdot, \cdot)$ be the cost function of edges in the alternating graph \mathcal{G}_{i+1} , i.e., after augmentation. From Lemma 9, one of the following cases is true.

- (i) Edge (v, u) is on the augmenting path P with $\text{PROJ}(v, u) = \langle (v, x), (x, u) \rangle$, or
- (ii) There is an edge $(u, n(v))$ in \mathcal{G}_{i+1}^0 with its projection $\text{PROJ}(u, n(v)) = \langle (u, x), (x, n(v)) \rangle$.

We will consider both these cases separately.

Case (i): For the edge (v, u) , $c(v, u) = d(v, x) - d(x, u)$. After augmentation, $\text{PROJ}(u, v) = \langle (u, x), (x, v) \rangle$ and the cost of the affected edge $(u, v) \in \mathcal{G}_{i+1}$ is $c'(u, v) = d(u, x) - d(x, v) = -c(v, u)$. From Lemma 5, after Step 4(a), the slack on every edge of the augmenting path P , including $(v, u) \in P$ is 0 i.e. $s(v, u) = (c(v, u) + y(u) - y(v)) = 0$. The slack on the affected edge $(u, v) \in \mathcal{G}_{i+1}$ can be calculated as,

$$\begin{aligned} s(u, v) &= c'(u, v) - y(u) + y(v) \\ &= -(c(v, u) + y(u) - y(v)) \\ &= 0. \end{aligned}$$

Hence, the affected edge $(u, v) \in \mathcal{G}_{i+1}$ is feasible.

Case (ii): There is an edge $(u, n(v))$ in \mathcal{G}_{i+1}^0 and $\text{PROJ}(u, n(v)) = \langle (u, x), (x, n(v)) \rangle$. From Lemma 5, after Step 4(a), every edge on P including $(v, n(v)) \in P$ has a slack of 0. Hence, $y(v) - y(n(v)) = c(v, n(v)) = d(v, x) - d(x, n(v))$, or

$$y(n(v)) = y(v) + d(x, n(v)) - d(v, x). \quad (9)$$

Since $(u, n(v))$ is an edge in \mathcal{G}_{i+1}^0 , it is a feasible edge after the step 4(a) (Lemma 8) and we get $y(u) - y(n(v)) \leq c(u, n(v))$. Substituting $c(u, n(v))$ as $d(u, x) - d(x, n(v))$, we get $y(u) - y(n(v)) \leq d(u, x) - d(x, n(v))$. From equation (9), we can rewrite this inequality as

$$\begin{aligned} y(u) - y(v) - d(x, n(v)) + d(v, x) &\leq d(u, x) - d(x, n(v)), \\ y(u) - y(v) &\leq d(u, x) - d(x, v), \\ y(u) - y(v) &\leq c'(u, v), \end{aligned}$$

implying that the affected edge (u, v) is a feasible edge.

Next, we show that σ_{i+1} is a valid solution after Step 4(b). To show that σ_{i+1} remains a valid solution, we need to show that the forward edges of G_{i+1} satisfies (Q1) and (Q2). By construction, every backward edge is directed from a later request to an earlier one. During augmentation, for every backward edge on the augmenting trail, we reverse its direction and label it as a forward edge. Therefore, every newly introduced forward edge is from an earlier request to a later one implying (Q1).

Next, we will show (Q2)

The first vertex of P is r_{i+1} . Let a be the anchor node at the end of P . Consider any edge (u, v) of the simple directed path P . Note that its projection is $\langle (u, p(v)), (p(v), v) \rangle$ where $(u, p(v))$ is a backward edge and $(p(v), v)$ is a forward edge. Due to augmentation, this projection is modified as follows: $(v, p(v))$ becomes a backward edge and $(p(v), u)$ is now a forward edge. We abuse notation and refer to the modifications made by augmentation along the projection of (u, v) as augmentation along the edge (u, v) .

Augmentation along (u, v) modifies the outgoing forward edge from $p(v)$. It also removes the incoming forward edge to v and adds an incoming forward edge to u . Therefore, augmentation along (u, v) does not change the number of forward edges coming in and going out of $p(v)$, i.e., $p(v)$ continues to satisfy (Q2). However, it increases the number of forward edges coming into u by 1 and reduces the number of incoming forward edges incident on v by 1.

Every vertex v' along P , except for the first and the last vertex, i.e., $v' \notin \{r_{i+1}, a\}$, will be the tail for some edge $(u', v') \in P$ and the head for some edge $(v', w') \in P$. Augmentation along (u', v') will reduce the incoming forward edge on v' by one. On the other hand,

augmenting along (v', w') will increase the incoming forward edge on v' by one. As a result the incoming and outgoing forward edges incident on v' remain unchanged. Therefore, for every vertex except r_{i+1} and a , we can conclude that (Q2) holds.

The first vertex r_{i+1} is the tail of the first edge in P , augmentation will result in a new incoming forward edge in G_{i+1} . Finally, the last vertex $a \in P$ is the head of the last edge. Therefore, augmentation causes removal of the only incoming forward edge to a . Instead, we add a forward edge from r_{i+1} to a . This guarantees that r_{i+1} contains exactly one incoming forward edge and one outgoing forward edge satisfying (Q2). Furthermore, the anchor node a will have exactly one incoming forward edge satisfying (Q2). ◀

Finally, we will show that the updated vertex weights satisfy (6).

► **Lemma 11.** *Consider any solution σ_i maintained by the algorithm with residual graph G_i , for any forward edge (u, v) in the residual graph,*

$$y(v) \geq d(u, v). \quad (10)$$

4.2 Proof of Invariant (I2)

Before we present the proof for (I2), we would like to remind the reader of the following discussion.

From Invariant (I1), Equation (4) and since $y(r_{i+1}) = 0$, the minimum net-cost path P_j from r_{i+1} to some anchor node a_j in the alternating graph G_{i+1}^0 is also the augmenting path that minimizes the sum of slacks along its edges. From invariant (I1) all slacks are non-negative and so, P_j will be the augmenting path returned by the execution of Dijkstra's algorithm in Step 1 of the algorithm. Furthermore, $\ell_{a_j} = \sum_{(u,v) \in P_j} s(u, v)$. Therefore, from Equation (4), we conclude that $\Phi_j = \Phi(P_j) = \ell_{a_j} - y(a_j)$. Therefore, Step 2 of the algorithm will correctly compute the minimum net-cost augmenting path to every anchor node $a_j \in \{a_1, \dots, a_k\}$.

Invariant (I2). Let \mathcal{C}^i be the final configuration of σ_i . Then, $\sigma_i = \sigma_i^*(\mathcal{C}^i)$. Furthermore, for every anchor node $a_j \in \{a_1, \dots, a_k\}$, let $\mathcal{C}_j^{i+1} = \mathcal{C}^i \setminus \{a_j\} \cup \{r_{i+1}\}$. Then $\Phi_j = w(\sigma_{i+1}^*(\mathcal{C}_j^{i+1})) - w(\sigma_i^*(\mathcal{C}^i))$.

Proof. Prior to processing any request, all the servers are in their initial configuration and σ_0 with zero cost is indeed the optimal solution. Assume that, after processing i requests, $\sigma_i = \sigma_i^*(\mathcal{C}^i)$. We will use this to show that $\Phi_j = w(\sigma_{i+1}^*(\mathcal{C}_j^{i+1})) - w(\sigma_i^*(\mathcal{C}^i))$ and $\sigma_{i+1} = \sigma_{i+1}^*(\mathcal{C}^{i+1})$.

Consider $\sigma_{i+1}^j = \sigma_{i+1}^*(\mathcal{C}_j^{i+1})$ to be the smallest cost solution that serves $i+1$ requests and ends in \mathcal{C}_j^{i+1} . If there are many minimum-cost solutions, we set σ_{i+1}^j to be the one that has the fewest edges in the symmetric difference with $\sigma_i = \sigma_i^*(\mathcal{C}^i)$. Consider the symmetric difference of the edges of σ_i and σ_{i+1}^j . Since their final configurations \mathcal{C}^i and \mathcal{C}_j^{i+1} differ in only the locations of a_j and r_{i+1} , the symmetric difference will include exactly one augmenting trail from r_{i+1} to a_j and possibly a set \mathbb{C} of alternating cycles.

First, we show that \mathbb{C} is an empty set. For the sake of contradiction, assume \mathbb{C} is not empty. Let C be an alternating cycle with respect to G_{i+1}^0 (extended graph for solution σ_i) in the symmetric difference. The net-cost of C cannot be zero, since otherwise applying the flip operation on the cycle C in G_{σ_i} will lead to another valid solution σ'' whose cost is identical to that of σ_{i+1}^j and the final configuration is \mathcal{C}_j^{i+1} . However, the flip operation will reduce

the size of the symmetric difference and so, σ_{i+1}^j has more edges than σ'' in the symmetric difference with σ_i . This contradicts our assumption that σ_{i+1}^j is the minimum-cost valid solution that ends in configuration \mathcal{C}_j^{i+1} and has the smallest symmetric difference with σ_i .

Similarly, the net-cost $\Phi(C)$ cannot be negative, since otherwise applying the flip operation along the cycle C on G_{i+1}^0 will lead to another valid solution that ends in \mathcal{C}^i and has a smaller cost than σ_i . This contradicts the fact that $\sigma_i = \sigma_i^*(\mathcal{C}^i)$ is a minimum-cost solution.

If the net-cost $\Phi(C)$ with respect to G_{i+1}^0 is positive, i.e., $\Phi(C) > 0$, then let C' be the alternating cycle corresponding to C in $G_{\sigma_{i+1}^j}$ (the residual graph with respect to σ_{i+1}^j). From Corollary 13 presented in Section 4.3, it follows that the net-cost $\Phi(C') = -\Phi(C) < 0$. Again, applying the flip operation along the cycle C' in $G_{\sigma_{i+1}^j}$ will lead to a valid solution whose cost is smaller than σ_{i+1}^j contradicting the fact that σ_{i+1}^j is the smallest cost solution.

From the above discussion, it follows that the symmetric difference of σ_i and σ_{i+1}^j is an augmenting trail T' . We claim that T' is in fact the minimum net-cost augmenting trail that starts at r_{i+1} and ends at a_j . For the sake of contradiction, suppose T' is not the minimum net-cost augmenting trail and $\Phi(T') > \Phi_j$. Let T be some minimum net-cost augmenting trail in G_{i+1}^0 that starts at r_{i+1} and ends at an anchor node a_j . Note that $\Phi_j = \Phi(T)$. Let $\bar{\sigma}_{i+1}^j$ be the valid solution obtained by augmenting σ_i along T in the extended graph G_{i+1}^0 . The final configuration of $\bar{\sigma}_{i+1}^j$ is \mathcal{C}_{i+1}^j . Then, by its definition,

$$\begin{aligned} w(\sigma_{i+1}^j) - w(\sigma_i) &> w(\bar{\sigma}_{i+1}^j) - w(\sigma_i), \\ w(\sigma_{i+1}^j) &> w(\bar{\sigma}_{i+1}^j), \end{aligned}$$

contradicting the fact that σ_{i+1}^j is a minimum cost solution to serve $i+1$ requests and end in configuration \mathcal{C}_{i+1}^j . Thus the net-cost of the augmenting trail T' in the symmetric difference of σ_i and σ_{i+1}^j is Φ_j . From the definition of net-cost, $\Phi_j = w(\sigma_{i+1}^j) - w(\sigma_i) = w(\sigma_{i+1}^*(\mathcal{C}_j^{i+1})) - w(\sigma_i^*(\mathcal{C}^i))$.

Our algorithm chooses the minimum net-cost path from r_{i+1} to a_m and augments the valid solution along this path. As a result, the cost of the solution σ_{i+1} increases precisely by $w(\sigma_{i+1}^*(\mathcal{C}_m^{i+1})) - w(\sigma_i^*(\mathcal{C}^i))$ and the new valid solution will end in configuration $\mathcal{C}_m^{i+1} = \mathcal{C}^{i+1}$ and have a cost equal to $w(\sigma_{i+1}^*(\mathcal{C}^{i+1}))$ proving invariant (I2). \blacktriangleleft

4.3 Symmetric Difference of Valid Solutions

Next, we introduce properties of the symmetric difference of two valid solutions. These properties are used in the proof of invariant (I2).

Let σ and σ' be two valid solutions where σ serves the first i requests and σ' serves the first $i+1$ requests. Let G_σ^0 be the extended graph with respect to σ and $G_{\sigma'}$ be the residual graph with respect to σ' . We show that the edges in the symmetric difference of σ and σ' can be decomposed into a edge-disjoint set of alternating trails, augmenting trail and alternating cycles.

Let \mathcal{C} and \mathcal{C}' be the final configurations of σ and σ' . Let X denote the edges in the symmetric difference of σ and σ' . For any edge in X , there is a corresponding directed edge in G_σ^0 . We assign the same direction for edge in X . Therefore, by construction, the edges of $X \cap \sigma$ will be forward edges and the edges of $X \cap \sigma'$ will be backward edges in G_σ^0 . Figures 3(a), (b) and (c) highlight the edges of σ , σ' and $(X \cap G_\sigma^0)$ respectively. For each vertex $v \in V_i$, suppose v is not an anchor node (resp. if v is not a vertex in the initial configuration), let $f(v)$ (resp. $p(v)$) denote the vertex that appears after (resp. before) v in σ . Similarly, for all $v \in V_{i+1}$, where v is not a vertex from the initial configuration (resp. not

an anchor node), let $p'(v)$ (resp. $f'(v)$) denote the vertex that appears before (resp. after) v in $G_{\sigma'}$. Let $\{a_1, \dots, a_k\}$ denote the anchor nodes of G_{σ}^0 and $\{a'_1, a'_2, \dots, a'_k\}$ denote the anchor nodes of $G_{\sigma'}$. Let $v'_j = p'(a'_j)$ in $G_{\sigma'}$ and $v_j = p(a_j)$ in G_{σ}^0 . Let $Y = \{v_1, \dots, v_k\}$ and let $Y' = \{v'_1, \dots, v'_k\}$. We use these notations throughout this section.

Next, consider any edge $(u, v) \in X$. Suppose (u, v) is a forward edge and $v \notin \{a_1, \dots, a_k\}$. Since (u, v) is in the symmetric difference and since v is not an anchor node, there is a different in-coming forward edge to v , namely (u', v) in $G_{\sigma'}$ with $u \neq u'$. The backward edge (v, u') will be in X and we denote the edge (v, u') as $\text{next}(u, v)$ in G_{σ}^0 . For example, in Figure 3(c), the backward edge (r_9, r_8) is $\text{next}(r_6, r_9)$. For the forward edge $(u, v) \in X$, there is a different out-going forward edge (u, v') in $G_{\sigma'}$. This edge appears as the backward edge (v', u) in X provided $v' \notin \{a'_1, a'_2, \dots, a'_k\}$. Therefore, we define the backward edge (v', u) to be the $\text{prev}(u, v)$ in X . For example, in Figure 3(c), $(r_3, r_1) = \text{prev}(r_1, r_7)$.

Finally, we define $\text{next}(u, v)$ and $\text{prev}(u, v)$ for the case where (u, v) is a backward edge in X . Since (u, v) is in the symmetric difference, the edge (v, u) is an out-going forward edge from v in $G_{\sigma'}$. Since (u, v) is in the symmetric difference, there is a different out-going forward edge from v , namely $(v, v') \in X$ with $v' \neq u$. We denote the forward edge (v, v') as $\text{next}(u, v)$ in G_{σ}^0 . For instance, in Figure 3(c), $(r_1, r_7) = \text{next}(r_3, r_1)$. Similarly, for the backward edge $(u, v) \in X$, if $u \neq r_{i+1}$, there is a different in-coming forward edge to u (v', u) in G_{σ} . This edge appears as a forward edge (v', u) in X . Therefore, we define the forward edge (v', u) to be the $\text{prev}(u, v)$ in X . For instance, in Figure 3(c), $(r_3, r_5) = \text{prev}(r_5, i_4)$.

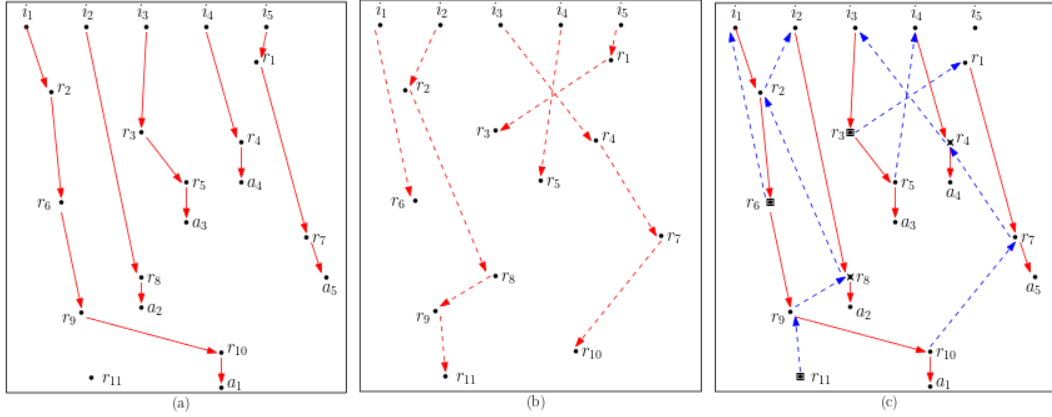
Thus, every edge in X have a unique $\text{next}(\cdot, \cdot)$ edge except those that are directed towards an anchor nodes $\{a_1, \dots, a_k\}$. Edges directed towards the anchor nodes $\{a_1, \dots, a_k\}$ do not have any $\text{next}(\cdot, \cdot)$ edge.

Similarly, every edge in X has a unique $\text{prev}(\cdot, \cdot)$, except the edges going out of $\{v'_1, v'_2, \dots, v'_k\}$. Edges going out of $\{v'_1, v'_2, \dots, v'_k\}$ do not have any previous edge.

Decomposing X into augmenting trail, alternating trails and cycles. Consider any vertex in $v \in Y \cap Y'$. The forward edge (v, a) is in X . However, since $v \in Y'$, the $\text{prev}(v, a)$ does not exist. Similarly, since $a \in \{a_1, \dots, a_k\}$, $\text{next}(v, a)$ does not exist. So, we create a trivial alternating trail with one edge (v, a) .

For every vertex $v \in Y' \setminus Y$, let (v, v') be the outgoing edge in X . We initialize T to be (v, v') . We construct an alternating trail incrementally by concatenating the last edge (u, v) of T with $\text{next}(u, v)$. This construction stops when we reach some edge (u', a) for which $\text{next}(u', a)$ is not defined. Suppose the vertex $v \neq r_{i+1}$, then this trail starts with a forward edge and ends at an anchor node. On the other hand, suppose $v = r_{i+1}$, this trail is an augmenting trail that starts with a backward edge and ends at an anchor node. Any edge (u, v) of X that did not participate in the alternating and augmenting trails have a well defined $\text{next}(u, v)$. Therefore, we can construct an alternating cycle that contains (u, v) by repeatedly concatenating the last added edge (u', v') with its $\text{next}(u', v')$. The construction stops when $\text{next}(u', v') = (u, v)$ and we get an alternating cycle. Figure 3(c) illustrates an example such decomposition. Vertex r_{10} and r_5 are in $Y \cap Y'$, so $\langle r_{10}, a_1 \rangle$ and $\langle r_5, a_3 \rangle$ are trivial trails. Vertex r_6 and r_3 are in $Y' \setminus Y$ and therefore we have two alternating trails $\langle r_6, r_9, r_8, a_2 \rangle$ and $\langle r_3, r_5, i_4, r_4, i_3, r_3, r_1, r_7, r_4, a_4 \rangle$. The vertex r_{11} is r_{i+1} , therefore we have an augmenting trail $\langle r_{11}, r_9, r_{10}, r_7, a_5 \rangle$. All the remaining edges form an alternating cycle $\langle r_2, r_6, i_1, r_2, i_2, r_8, r_2 \rangle$.

Note that the construction described above also extends to the residual graph $G_{\sigma'}$ and we obtain the following lemma.



■ **Figure 3** (a) Forward edges of G_σ^0 , representing σ , (b) Forward edges of $G_{\sigma'}$, representing σ' , (c) Edges in the symmetric difference of σ and σ' . Edges of σ are shown as forward edges in G_σ (red edges), and edges of σ' are shown as the backward edges in G_σ (dashed blue edges). This graph has an augmenting trail $\langle r_{11}, r_9, r_{10}, r_7, a_5 \rangle$, two alternating trails $\langle r_6, r_9, r_8, a_2 \rangle$, $\langle r_3, r_5, i_4, r_4, i_3, r_3, r_1, r_7, r_4, a_4 \rangle$ and one directed alternating cycle $\langle r_2, r_6, i_1, r_2, i_2, r_8, r_2 \rangle$.

► **Lemma 12.** *The edges of symmetric difference X of two valid solutions σ and σ' in G_σ^0 can be decomposed into (a) one augmenting trail A , (b) a set \mathbb{T} of $|Y' \setminus Y| - 1$ non-trivial alternating trails that start with a forward edge, (c) a set \mathbb{T}' of $|Y' \cap Y|$ trivial alternating trails and (d) a set \mathbb{C} of alternating cycles. Similarly, the edges of X in $G_{\sigma'}$ can be decomposed into alternating trails and cycles each of which are obtained by simply applying the flip operation to the trails and cycles in A , \mathbb{T} , \mathbb{T}' and \mathbb{C}*

► **Corollary 13.** *Given the set X of edges in the symmetric difference, consider the decomposition of X into $\{A\} \cup \mathbb{T} \cup \mathbb{C}$ in G_σ^0 as described in Lemma 12. Then the edges of X in $G_{\sigma'}$ can be decomposed into alternating trails such that for each alternating trail (resp. cycle) $T \in \{A\} \cup \mathbb{T} \cup \mathbb{C}$, there is an alternating trail T' (resp. cycle) induced by the edges of X in $G_{\sigma'}$ such that T' is obtained by applying the flip operation on T and $\Phi(T) = -\Phi(T')$.*

5 Missing Proofs

Proof of Lemma 11. The initial solution σ_0 is a trivially valid solution and the only edges in the residual graph are forward edges that go from a vertex in the initial configuration to an anchor node. For any such forward edge (u, v) , v is an anchor node with $y(v) = 0$. The cost $d(u, v)$ is also 0 since the anchor node v and the vertex u share the same location. Therefore, inequality (10) holds.

Let us assume that the inequality (10) holds after request r_i is processed by our algorithm. To complete the proof, we will show that the inequality will continue to be satisfied after request r_{i+1} is processed. To do so, we will show that the any of the changes made by the algorithm will not violate inequality (10).

At the start of the algorithm, we add r_{i+1} to G_i to create G_{i+1}^0 . Since all the edges incident on r_{i+1} in G_{i+1}^0 are backward edges, all forward edges will continue to satisfy inequality (10). Steps 1, 2 and 3 do not alter the weights $y(\cdot)$ or the alternating graph. Therefore, the inequality (10) continues to hold for every forward edge during these three steps. In Step 4(a), we modify the vertex weights for every vertex v with $\ell_v < \ell$. Consider any such vertex and let (u, v) be the in-coming forward edge to v . Recollect that $y(v)$ is

the weight prior to the execution of Step 4(a) and $y'(v)$ is the weight after Step 4(a). Since inequality (10) is true prior to execution of Step 4(a), we have $y(v) \geq d(u, v)$. In Step 4(a), the weight is updated to $y'(v) \leftarrow y(v) - \ell_v + \ell$ provided $\ell_v < \ell$. Since $\ell_v < \ell$, it follows that $y'(v) \geq y(v) \geq d(u, v)$ and inequality (10) continues to hold.

Next, we show that the inequality (10) continues to hold after Step 4(b). In Step 4(b), we apply the augment operation along an augmenting trail T (computed in Step 3). Recollect that the first vertex of T is r_{i+1} and the last vertex of T is an anchor node a . Note that the weights of every vertex, except the anchor node a remains unchanged. However, for any v along the alternating trail T , its incoming forward edge may change. As a result of augmentation along T , every backward edge (v_2, v_1) in T changes to a forward edge (v_1, v_2) . Let P be the augmenting path in the alternating graph \mathcal{G}_{i+1}^0 such that trail $T = \text{PROJ}(P)$. Let (v_2, v_1) be a backward edge in T . Let (v_2, w) be the edge in P such that $\text{PROJ}(v_2, w)$ contains the backward edge (v_2, v_1) , i.e., $\text{PROJ}(v_2, w) = \langle (v_2, v_1), (v_1, w) \rangle$. By definition of slack,

$$\begin{aligned} s(v_2, w) &= c(v_2, w) - y(v_2) + y(w) \\ &= d(v_2, v_1) - d(v_1, w) - y(v_2) + y(w). \end{aligned}$$

Since $(v_2, w) \in P$, at the end of Step 4(a), the slack $s(v_2, w)$ becomes 0 (Lemma 5). Therefore, we have $(d(v_2, v_1) - d(v_1, w)) - y'(v_2) + y'(w) = 0$ which can be rearranged as

$$y'(v_2) = d(v_2, v_1) - d(v_1, w) + y'(w). \quad (11)$$

Since, (v_1, w) is a forward edge in T , after Step 4(a), we have $y'(w) \geq d(v_1, w)$. Substituting this in 11, we get

$$y'(v_2) \geq d(v_1, v_2). \quad (12)$$

After augmentation, the backward edge $(v_2, v_1) \in \text{PROJ}(v_2, w)$ changes to a forward edge $(v_1, v_2) \in G_{i+1}$. Hence, inequality (11) implies inequality (10) continues to hold. Therefore, inequality (10) continues to hold after the augment operation in Step 4(b).

Finally, step 4(b) also adds a forward edge from r_{i+1} to a_m and modifies the vertex weight of the anchor node a_m to 0. Since the cost of this forward edge $d(r_{i+1}, a_m)$ is 0, inequality (10) holds for the vertex a_m . This concludes the argument that inequality (10) holds after Step 4(b). ◀

References

- 1 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k -server via multiscale entropic regularization. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16. ACM, 2018. doi:10.1145/3188745.3188798.
- 2 Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discret. Math.*, 4(2):172–181, 1991. doi:10.1137/0404017.
- 3 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k -servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991. doi:10.1137/0220008.
- 4 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.
- 5 B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993. doi:10.1006/jagm.1993.1026.

- 6 Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994. doi:10.1016/0304-3975(94)90042-6.
- 7 Elias Koutsoupias. The k -server problem. *Comput. Sci. Rev.*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.
- 8 Elias Koutsoupias and Christos H. Papadimitriou. On the k -server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.
- 9 Harold W. Kuhn. The hungarian method for the assignment problem. In Michael Jünger, Thomas M. Lieblich, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. doi:10.1007/978-3-540-68279-0_2.
- 10 James R. Lee. Fusible hsts and the randomized k -server conjecture. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 438–449. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00049.
- 11 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.
- 12 Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 505–515. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.53.
- 13 Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPICs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.APPROX-RANDOM.2016.18.
- 14 Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, volume 99 of *LIPICs*, pages 67:1–67:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.SoCG.2018.67.
- 15 Tomislav Rudec, Alfonso Baumgartner, and Robert Manger. A fast work function algorithm for solving the k -server problem. *Central Eur. J. Oper. Res.*, 21(1):187–205, 2013. doi:10.1007/s10100-011-0222-7.
- 16 Tomislav Rudec, Alfonso Baumgartner, and Robert Manger. A fast work function algorithm for solving the k -server problem. *Central European Journal of Operations Research*, 21:187–205, 2013.
- 17 Tomislav Rudec and Robert Manger. A new approach to solve the k -server problem based on network flows and flow cost reduction. *Comput. Oper. Res.*, 40(4):1004–1013, 2013. doi:10.1016/j.cor.2012.11.006.

Erdős–Selfridge Theorem for Nonmonotone CNFs

Md Lutfar Rahman ✉

Amazon, Seattle, WA, USA

Thomas Watson ✉

University of Memphis, TN, USA

Abstract

In an influential paper, Erdős and Selfridge introduced the Maker-Breaker game played on a hypergraph, or equivalently, on a monotone CNF. The players take turns assigning values to variables of their choosing, and Breaker’s goal is to satisfy the CNF, while Maker’s goal is to falsify it. The Erdős–Selfridge Theorem says that the least number of clauses in any monotone CNF with k literals per clause where Maker has a winning strategy is $\Theta(2^k)$.

We study the analogous question when the CNF is not necessarily monotone. We prove bounds of $\Theta(\sqrt{2}^k)$ when Maker plays last, and $\Omega(1.5^k)$ and $O(r^k)$ when Breaker plays last, where $r = (1 + \sqrt{5})/2 \approx 1.618$ is the golden ratio.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Game, nonmonotone, CNFs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.31

Funding This work was supported by NSF grants CCF-1657377 and CCF-1942742.

Acknowledgements We thank anonymous reviewers for their comments.

1 Introduction

In 1973, Erdős and Selfridge published a paper [3] with several fundamental contributions, including:

- Being widely regarded as the genesis of the method of conditional expectations. The subsequent impact of this method on theoretical computer science needs no explanation.
- Introducing the so-called Maker-Breaker game, variants of which have since been studied in numerous papers in the combinatorics literature.

We revisit that seminal work and steer it in a new direction. The main theorem from [3] can be phrased in terms of CNFs (conjunctive normal form boolean formulas) that are monotone (they contain only positive literals). We investigate what happens for general CNFs, which may contain negative literals. We feel that the influence of Erdős–Selfridge and the pervasiveness of CNFs in theoretical computer science justify this question as inherently worthy of attention. Our pursuit of the answer uncovers new techniques and invites the development of further techniques to achieve a full resolution in the future.

In the Maker-Breaker game played on a monotone CNF, the eponymous players take turns assigning boolean values to variables of their choosing. Breaker wins if the CNF gets satisfied, and Maker wins otherwise; there are no draws. Since the CNF is monotone, Breaker might as well assign 1 to every variable she picks, and Maker might as well assign 0 to every variable he picks. In the generalization to nonmonotone CNFs, each player can pick which remaining variable and which bit to assign it during their turn. To distinguish this general game, we rename Breaker as T (for “true”) and Maker as F (for “false”). The computational complexity of deciding which player has a winning strategy has been studied in [10, 11, 2, 5, 6, 1, 7, 8, 9].



© Md Lutfar Rahman and Thomas Watson;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 31; pp. 31:1–31:11

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

31:2 Erdős–Selfridge Theorem for Nonmonotone CNFs

A CNF is *k-uniform* when every clause has exactly k literals (corresponding to k distinct variables). The Erdős–Selfridge Theorem answers an extremal question: How few clauses can there be in a k -uniform monotone CNF that Maker can win? It depends a little on which player gets the opening move: 2^k if Breaker plays first, and 2^{k-1} if Maker plays first. The identity of the player with the final move doesn't affect the answer for monotone CNFs. In contrast, “who gets the last laugh” matters a lot for general CNFs:

► **Theorem 1 (informal).** *If F plays last, then the least number of clauses in any k -uniform CNF where F has a winning strategy is $\Theta(\sqrt{2}^k)$.*

► **Theorem 2 (informal).** *If T plays last, then the least number of clauses in any k -uniform CNF where F has a winning strategy is $\Omega(1.5^k)$ and $O(r^k)$ where $r = (1 + \sqrt{5})/2 \approx 1.618$.*

The most involved proof is the $\Omega(1.5^k)$ lower bound in Theorem 2. We conjecture the correct bound is $\Theta(r^k)$.

2 Results

In the *unordered CNF game*, there is a CNF φ and a set of variables X containing all variables that appear in φ and possibly more. The players T and F alternate turns; each turn consists of picking an unassigned variable from X and picking a value 0 or 1 to assign it.¹ The game ends when all variables are assigned; T wins if φ is satisfied (every clause has a true literal), and F wins if φ is unsatisfied (some clause has all false literals). There are four possible patterns according to “who goes first” and “who goes last.” If the same player has the first and last moves, then $|X|$ is odd, and if different players have the first and last moves, then $|X|$ is even.

► **Definition 3.** *For $k \geq 0$ and $a, b \in \{T, F\}$, we let $M_{k,a..b}$ be the minimum number of clauses in φ , over all unordered CNF game instances (φ, X) where φ is k -uniform and F has a winning strategy when player a has the first move and player b has the last move.*

► **Theorem 1 (formal).** $M_{k,T..F} = \sqrt{2}^k$ for even k , and $1.5\sqrt{2}^{k-1} \leq M_{k,T..F} \leq \sqrt{2}^{k+1}$ for odd k .

Let Fib_k denote the k^{th} Fibonacci number. It is well-known that $\text{Fib}_k = \Theta(r^k)$ where $r = (1 + \sqrt{5})/2 \approx 1.618$.

► **Theorem 2 (formal).** $1.5^k \leq M_{k,T..T} \leq \text{Fib}_{k+2}$ for all k .

► **Observation 3.** $M_{k,F..b} = M_{k-1,T..b}$ for all $k \geq 1$ and $b \in \{T, F\}$.

Proof. $M_{k,F..b} \leq M_{k-1,T..b}$: Suppose F wins (φ, X) when T moves first, where φ is $(k-1)$ -uniform. Then F wins $(\varphi', X \cup \{x_0\})$ when F moves first, where x_0 is a fresh variable (not already in X) and φ' is the same as φ but with x_0 added to each clause. F's winning strategy is to play $x_0 = 0$ first and then use the winning strategy for (φ, X) . Note that φ' is k -uniform and has the same number of clauses as φ .

$M_{k-1,T..b} \leq M_{k,F..b}$: Suppose F wins (φ, X) when F moves first, where φ is k -uniform. Say the opening move in F's winning strategy is $\ell_i = 1$, where $\ell_i \in \{x_i, \bar{x}_i\}$ is some literal. Obtain φ' from φ by removing each clause containing ℓ_i , removing $\bar{\ell}_i$ from each clause

¹ This game is called “unordered” to contrast it with the related TQBF game, in which the variables must be played in a prescribed order.

containing $\bar{\ell}_i$, and removing an arbitrary literal from each clause containing neither ℓ_i nor $\bar{\ell}_i$. Then F wins $(\varphi', X - \{x_i\})$ when T moves first, and φ' is $(k-1)$ -uniform and has at most as many clauses as φ . ◀

► **Corollary 4.**

- $M_{k,F\dots F} = \sqrt{2}^{k-1}$ for odd k , and $1.5\sqrt{2}^{k-2} \leq M_{k,F\dots F} \leq \sqrt{2}^k$ for even k .
- $1.5^{k-1} \leq M_{k,F\dots T} \leq \text{Fib}_{k+1}$ for all k .

(Observation 3 requires $k \geq 1$, but the bounds in Corollary 4 also hold for $k = 0$ since $M_{0,a\dots b} = 1$: F wins a CNF with an empty clause, and T wins a CNF with no clauses.)

3 Upper bounds

In this section, we prove the upper bounds of Theorem 1 and Theorem 2 by giving examples of game instances with few clauses where F wins. In [3], Erdős and Selfridge proved the upper bound for the Maker-Breaker game by showing a k -uniform monotone CNF with 2^k clauses where Maker (F) wins. The basic idea is that F can win on the following formula, which is not a CNF:

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \cdots \vee (x_{2k-1} \wedge x_{2k})$$

Whenever T plays a variable, F responds by assigning 0 to the paired variable. By the distributive law, this expands to a k -uniform monotone CNF with 2^k clauses. We study nonmonotone CNFs, which may have both positive and negative literals.

3.1 F plays last

► **Lemma 5.** $M_{k,T\dots F} \leq \sqrt{2}^k$ for even k .

Proof. F can win on the following formula, which is not a CNF, with variables $X_k = \{x_1, \dots, x_k\}$.

$$(x_1 \oplus x_2) \vee (x_3 \oplus x_4) \vee \cdots \vee (x_{k-1} \oplus x_k)$$

Whenever T plays a variable, F responds by playing the paired variable to make them equal. To convert this formula to an equivalent CNF, first replace each $(x_i \oplus x_{i+1})$ with $(x_i \vee x_{i+1}) \wedge (\bar{x}_i \vee \bar{x}_{i+1})$. Then by the distributive law, this expands to a k -uniform CNF φ_k where one clause is

$$((x_1 \vee x_2) \vee (x_3 \vee x_4) \vee \cdots \vee (x_{k-1} \vee x_k))$$

and for $i \in \{1, 3, 5, \dots, k-1\}$, each clause contains either $(x_i \vee x_{i+1})$ or $(\bar{x}_i \vee \bar{x}_{i+1})$. Therefore φ_k has $2^{k/2} = \sqrt{2}^k$ clauses: one clause for each $S \subseteq \{1, 3, 5, \dots, k-1\}$. F wins in (φ_k, X_k) . ◀

► **Lemma 6.** $M_{k,T\dots F} \leq \sqrt{2}^{k+1}$ for odd k .

Proof. Suppose φ_{k-1} is the $(k-1)$ -uniform CNF with $\sqrt{2}^{k-1}$ clauses from Lemma 5 (since $k-1$ is even). We take two copies of φ_{k-1} , and put a new variable x_k in each clause of one copy, and a new variable x_{k+1} in each clause of the other copy. Call this φ_k . Formally:

$$\begin{aligned} \varphi_k &= \bigwedge_{C \in \varphi_{k-1}} (C \vee x_k) \wedge (C \vee x_{k+1}) \\ X_k &= \{x_1, x_2, \dots, x_{k+1}\} \end{aligned}$$

31:4 Erdős–Selfridge Theorem for Nonmonotone CNFs

We argue F wins in (φ_k, X_k) . If T plays x_k or x_{k+1} , F responds by assigning 0 to the other one. For other variables, F follows his winning strategy for (φ_{k-1}, X_{k-1}) from Lemma 5. Since φ_{k-1} is a $(k-1)$ -uniform CNF with $\sqrt{2}^{k-1}$ clauses, φ_k is a k -uniform CNF with $2\sqrt{2}^{k-1} = \sqrt{2}^{k+1}$ clauses. ◀

3.2 T plays last

Before proving Lemma 7 we draw an intuition. We already know that F wins on

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \cdots \vee (x_{2k-1} \wedge x_{2k}).$$

Now replace each $(x_i \wedge x_{i+1})$ with $(x_i \wedge (\bar{x}_i \vee x_{i+1}))$, which is equivalent. This does not change the function expressed by the formula, so F still wins this T \cdots F game. To turn it into a T \cdots T game, we can introduce a dummy variable x_0 . Since the game is equivalent to a monotone game, neither player has any incentive to play x_0 , so F still wins this T \cdots T game [4, Proposition 2.1.6].

If we convert it to a CNF, then by the distributive law it will again have 2^k clauses. But this CNF is not uniform – each clause has at least k literals and at most $2k$ literals. We can do a similar construction that balances the CNF to make it uniform. This intuitively suggests that $\sqrt{2}^k < M_{k,T\dots T} < 2^k$.

► **Lemma 7.** $M_{k,T\dots T} \leq \text{Fib}_{k+2}$.

Proof. For every $k \in \{0, 1, 2, \dots\}$ we recursively define a k -uniform CNF φ_k on variables X_k , where $X_k = \{x_0, x_1, \dots, x_{2k-2}\}$ if $k > 0$, and $X_0 = \{x_0\}$ (these φ_k, X_k are different than in Subsection 3.1):

- $k = 0$: $\varphi_0 = ()$
- $k = 1$: $\varphi_1 = (x_0) \wedge (\bar{x}_0)$
- $k > 1$: $\varphi_k = \bigwedge_{C \in \varphi_{k-1}} (C \vee x_{2k-3}) \wedge \bigwedge_{C \in \varphi_{k-2}} (C \vee \bar{x}_{2k-3} \vee x_{2k-2})$

Now we argue F wins in (φ_k, X_k) . F’s strategy is to assign 0 to at least one variable from each pair $\{x_1, x_2\}, \{x_3, x_4\}, \{x_5, x_6\}, \dots, \{x_{2k-3}, x_{2k-2}\}$. Whenever T plays from a pair, F responds by assigning 0 to the other variable. After T plays x_0 , F picks a fresh pair $\{x_i, x_{i+1}\}$ where i is odd and assigns one of them 0, then “chases” T until T plays the other from $\{x_i, x_{i+1}\}$. Here the “chase” means whenever T plays from a fresh pair, F responds by assigning 0 to the other variable in that pair. After T returns to $\{x_i, x_{i+1}\}$, then F picks another fresh pair to start another chase, and so on in phases. We prove by induction on k that this strategy ensures φ_k is unsatisfied:

- $k = 0$: φ_0 is obviously unsatisfied.
- $k = 1$: φ_1 is obviously unsatisfied.
- $k > 1$: By induction, both φ_{k-1} and φ_{k-2} are unsatisfied. Now φ_k is unsatisfied since: By F’s strategy, at least one of $\{x_{2k-3}, x_{2k-2}\}$ is assigned 0. If $x_{2k-3} = 0$ then one of the clauses of φ_k that came from φ_{k-1} is unsatisfied. If $x_{2k-3} = 1$ and $x_{2k-2} = 0$ then one of the clauses of φ_k that came from φ_{k-2} is unsatisfied.

Letting $|\varphi_k|$ represent the number of clauses in φ_k , we argue $|\varphi_k| = \text{Fib}_{k+2}$ by induction on k :

- $k = 0$: $|\varphi_0| = 1 = \text{Fib}_2$.
- $k = 1$: $|\varphi_1| = 2 = \text{Fib}_3$.
- $k > 1$: By induction, $|\varphi_{k-1}| = \text{Fib}_{k+1}$ and $|\varphi_{k-2}| = \text{Fib}_k$. So

$$|\varphi_k| = |\varphi_{k-1}| + |\varphi_{k-2}| = \text{Fib}_{k+1} + \text{Fib}_k = \text{Fib}_{k+2}.$$

Therefore $M_{k,T\dots T} \leq \text{Fib}_{k+2}$. ◀

4 Lower bounds

4.1 Notation

In the proofs, we will define a potential value $p(C)$ for each clause C . The value of $p(C)$ depends on the context. If φ is a CNF (any set of clauses), then the potential of φ is $p(\varphi) = \sum_{C \in \varphi} p(C)$. The potential of a literal ℓ_i with respect to φ is defined as $p(\varphi, \ell_i) = p(\{C \in \varphi : \ell_i \in C\})$. When we have a particular φ in mind, we can abbreviate $p(\varphi, \ell_i)$ as $p(\ell_i)$.

Suppose φ is a CNF and ℓ_i, ℓ_j are two literals. We define the potentials of different sets of clauses based on which of ℓ_i, ℓ_j , and their complements exist in the clause. For example, $a(\varphi, \ell_i, \ell_j)$ is the sum of the potentials of clauses in φ that contain both ℓ_i, ℓ_j .

| | ℓ_j | $\bar{\ell}_j$ | neither ℓ_j nor $\bar{\ell}_j$ |
|-------------------------------------|----------|----------------|-------------------------------------|
| ℓ_i | a | b | c |
| $\bar{\ell}_i$ | d | e | f |
| neither ℓ_i nor $\bar{\ell}_i$ | g | h | |

$$\begin{aligned}
 a(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \ell_i \in C \text{ and } \ell_j \in C\}) \\
 b(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \ell_i \in C \text{ and } \bar{\ell}_j \in C\}) \\
 c(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \ell_i \in C \text{ and } \ell_j \notin C \text{ and } \bar{\ell}_j \notin C\}) \\
 d(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \bar{\ell}_i \in C \text{ and } \ell_j \in C\}) \\
 e(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \bar{\ell}_i \in C \text{ and } \bar{\ell}_j \in C\}) \\
 f(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \bar{\ell}_i \in C \text{ and } \ell_j \notin C \text{ and } \bar{\ell}_j \notin C\}) \\
 g(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \ell_i \notin C \text{ and } \bar{\ell}_i \notin C \text{ and } \ell_j \in C\}) \\
 h(\varphi, \ell_i, \ell_j) &= p(\{C \in \varphi : \ell_i \notin C \text{ and } \bar{\ell}_i \notin C \text{ and } \bar{\ell}_j \in C\})
 \end{aligned}$$

We can abbreviate these quantities as a, b, c, d, e, f, g, h in contexts where we have particular φ, ℓ_i, ℓ_j in mind. Also the following relations hold:

$$\begin{aligned}
 p(\ell_i) &= a + b + c \\
 p(\bar{\ell}_i) &= d + e + f \\
 p(\ell_j) &= a + d + g \\
 p(\bar{\ell}_j) &= b + e + h
 \end{aligned}$$

When we assign $\ell_i = 1$ (i.e., assign $x_i = 1$ if ℓ_i is x_i , or assign $x_i = 0$ if ℓ_i is \bar{x}_i), φ becomes the *residual* CNF denoted $\varphi[\ell_i = 1]$ where all clauses containing ℓ_i get removed, and the literal $\bar{\ell}_i$ gets removed from remaining clauses.

4.2 F plays last

► **Lemma 8.** $M_{k, T \dots F} \geq \sqrt{2}^k$ for even k .

Proof. Consider any $T \dots F$ game instance (φ, X) where φ is a k -uniform CNF with $< \sqrt{2}^k$ clauses and $|X|$ is even. We show T has a winning strategy. In this proof, we use $p(C) = 1/\sqrt{2}^{|C|}$. A *round* consists of a T move followed by an F move.

31:6 Erdős–Selfridge Theorem for Nonmonotone CNFs

▷ **Claim 9.** In every round, there exists a move for T such that for every response by F, we have $p(\psi) \geq p(\psi')$ where ψ is the residual CNF before the round and ψ' is the residual CNF after the round.

At the beginning we have $p(C) = 1/\sqrt{2}^k$ for each clause $C \in \varphi$, so $p(\varphi) < \sqrt{2}^k/\sqrt{2}^k = 1$. By Claim 9, T has a strategy guaranteeing that $p(\psi) \leq p(\varphi) < 1$ where ψ is the residual CNF after all variables have been played. If this final ψ contained a clause, the clause would be empty and have potential $1/\sqrt{2}^0 = 1$, which would imply $p(\psi) \geq 1$. Thus the final ψ must have no clauses, which means φ got satisfied and T won. This concludes the proof of Lemma 8, except for the proof of Claim 9. ◀

Proof of Claim 9. Let ψ be the residual CNF at the beginning of a round. T picks a literal ℓ_i maximizing $p(\psi, \ell_i)$ and plays $\ell_i = 1$.² Suppose F responds by playing $\ell_j = 1$, and let ψ' be the residual CNF after F's move. Letting the a, b, c, d, e, f, g, h notation be with respect to ψ, ℓ_i, ℓ_j , we have

$$p(\psi) - p(\psi') = a + b + c + d + g - (e + (\sqrt{2} - 1)(f + h))$$

because:

- Clauses from the a, b, c, d, g groups are satisfied and removed (since they contain $\ell_i = 1$ or $\ell_j = 1$ or both), so their potential gets multiplied by 0.
- Clauses from the e group each shrink by two literals (since they contain $\bar{\ell}_i = 0$ and $\bar{\ell}_j = 0$), so their potential gets multiplied by $\sqrt{2} \cdot \sqrt{2} = 2$.
- Clauses from the f, h groups each shrink by one literal, so their potential gets multiplied by $\sqrt{2}$.

By the choice of ℓ_i , we have $p(\ell_i) \geq p(\bar{\ell}_i)$ and $p(\ell_i) \geq p(\bar{\ell}_j)$ with respect to ψ , in other words, $a + b + c \geq d + e + f$ and $a + b + c \geq b + e + h$. Thus $p(\psi) \geq p(\psi')$ because

$$\begin{aligned} a + b + c + d + g &\geq a + b + c \geq \frac{1}{2}(d + e + f) + \frac{1}{2}(b + e + h) \geq e + \frac{1}{2}(f + h) \\ &\geq e + (\sqrt{2} - 1)(f + h). \end{aligned}$$

◁

Note: It did not matter whether k is even or odd! Lemma 8 is true for any k . Lemma 10 actually uses oddness of k . The main idea is to exploit the slack $1/2 \geq \sqrt{2} - 1$ that appeared at the end of the proof of Claim 9.

► **Lemma 10.** $M_{k, T \dots F} \geq 1.5 \sqrt{2}^{k-1}$ for odd k .

Proof. Consider any T...F game instance (φ, X) where φ is a k -uniform CNF with $< 1.5 \sqrt{2}^{k-1}$ clauses and $|X|$ is even. We show T has a winning strategy. In this proof, we use

$$p(C) = \begin{cases} 1/\sqrt{2}^{|C|} & \text{if } |C| \text{ is even.} \\ 1/1.5\sqrt{2}^{|C|-1} & \text{if } |C| \text{ is odd.} \end{cases}$$

▷ **Claim 11.** In every round, there exists a move for T such that for every response by F, we have $p(\psi) \geq p(\psi')$ where ψ is the residual CNF before the round and ψ' is the residual CNF after the round.

² It is perhaps counterintuitive that T's strategy ignores the effect of clauses that contain $\bar{\ell}_i$, which increase in potential after playing $\ell_i = 1$. A more intuitive strategy would be to pick a literal ℓ_i maximizing $p(\psi, \ell_i) - (\sqrt{2} - 1)p(\psi, \bar{\ell}_i)$, which is the overall decrease in potential from playing $\ell_i = 1$; this strategy also works but is trickier to analyze.

At the beginning we have $p(C) = 1/1.5\sqrt{2}^{k-1}$ for each clause $C \in \varphi$ (since $|C| = k$, which is odd), so $p(\varphi) < 1.5\sqrt{2}^{k-1}/1.5\sqrt{2}^{k-1} = 1$. By Claim 11, T has a strategy guaranteeing that $p(\psi) \leq p(\varphi) < 1$ where ψ is the residual CNF after all variables have been played. If this final ψ contained a clause, the clause would be empty and have potential $1/\sqrt{2}^0 = 1$ (since 0 is even), which would imply $p(\psi) \geq 1$. Thus the final ψ must have no clauses, which means φ got satisfied and T won. This concludes the proof of Lemma 10, except for the proof of Claim 11. \blacktriangleleft

Proof of Claim 11. Let ψ be the residual CNF at the beginning of a round. T picks a literal ℓ_i maximizing $p(\psi, \ell_i)$ and plays $\ell_i = 1$. Suppose F responds by playing $\ell_j = 1$, and let ψ' be the residual CNF after F's move. Letting the a, b, c, d, e, f, g, h notation be with respect to ψ, ℓ_i, ℓ_j , we have

$$p(\psi) - p(\psi') \geq a + b + c + d + g - \left(e + \frac{1}{2}(f + h)\right)$$

because:

- Clauses from the a, b, c, d, g groups are satisfied and removed (since they contain $\ell_i = 1$ or $\ell_j = 1$ or both), so their potential gets multiplied by 0.
- Clauses from the e group each shrink by two literals (since they contain $\bar{\ell}_i = 0$ and $\bar{\ell}_j = 0$). Here odd-width clauses remain odd and even-width clauses remain even, so their potential gets multiplied by $\sqrt{2} \cdot \sqrt{2} = 2$.
- Clauses from the f, h groups each shrink by one literal. There are two cases for a clause C in these groups:
 - $|C|$ is even, so $p(C) = 1/\sqrt{2}^{|C|}$. After C being shrunk by 1, the new clause C' has potential $p(C') = 1/1.5\sqrt{2}^{|C'|-1} = 1/1.5\sqrt{2}^{|C|-2}$. So the potential of an even-width clause gets multiplied by $p(C')/p(C) = 4/3$.
 - $|C|$ is odd, so $p(C) = 1/1.5\sqrt{2}^{|C|-1}$. After C being shrunk by 1, the new clause C' has potential $p(C') = 1/\sqrt{2}^{|C'|} = 1/\sqrt{2}^{|C|-1}$. So the potential of an odd-width clause gets multiplied by $p(C')/p(C) = 3/2$.

So their potential gets multiplied by $\leq 3/2$ (since $4/3 \leq 3/2$).

By the choice of ℓ_i , we have $p(\ell_i) \geq p(\bar{\ell}_i)$ and $p(\ell_i) \geq p(\bar{\ell}_j)$ with respect to ψ , in other words, $a + b + c \geq d + e + f$ and $a + b + c \geq b + e + h$. Thus $p(\psi) \geq p(\psi')$ because

$$a + b + c + d + g \geq a + b + c \geq \frac{1}{2}(d + e + f) + \frac{1}{2}(b + e + h) \geq e + \frac{1}{2}(f + h). \quad \triangleleft$$

4.3 T plays last

► **Lemma 12.** $M_{k, T \dots T} \geq 1.5^k$.

Proof. Consider any $T \dots T$ game instance (φ, X) where φ is a k -uniform CNF with $< 1.5^k$ clauses and $|X|$ is odd. We show T has a winning strategy. In this proof, we use $p(C) = 1/1.5^{|C|}$.

For intuition, how can T take advantage of having the last move? She will look out for certain pairs of literals to “set aside” and wait for F to assign one of them, and then respond by assigning the other one the opposite value. We call such a pair “zugzwang,” which means a situation where F’s obligation to make a move is a disadvantage for F. Upon finding such a pair, T anticipates that certain clauses will get satisfied later, but other clauses containing those literals might shrink when the zugzwang pair eventually gets played. Thus T can update the CNF to pretend those events have already transpired. The normal gameplay of TF rounds (T plays, then F plays) will sometimes get interrupted by FT rounds

of playing previously-designated zugzwang pairs. We define the zugzwang condition so that T’s modifications won’t increase the potential of the CNF (which is no longer simply a residual version of φ). When there are no remaining zugzwang pairs to set aside, we can exploit this fact – together with T’s choice of “best” literal for her normal move – to analyze the potential change in a TF round. This allows the proof to handle a smaller potential function and hence more initial clauses, compared to when F had the last move.

We describe T’s winning strategy in (φ, X) as Algorithm 1. In the first line, the algorithm declares and initializes ψ, Y, ζ, Z , which are accessed globally. Here ψ is a CNF (initially the same as φ), and ζ is a set (conjunction) of constraints of the form $(\ell_i \oplus \ell_j)$. We consider $(\ell_i \oplus \ell_j), (\ell_j \oplus \ell_i), (\bar{\ell}_i \oplus \bar{\ell}_j), (\bar{\ell}_j \oplus \bar{\ell}_i)$ to be the same constraint as each other. The algorithm maintains the following three invariants:

- (1) Y and Z are disjoint subsets of X , and $Y \cup Z$ is the set of unplayed variables, and Y contains all variables that appear in ψ , and Z is exactly the set of variables that appear in ζ , and $|Z|$ is even.
- (2) For every assignment to $Y \cup Z$, if ψ and ζ are satisfied, then φ is also satisfied by the same assignment together with the assignment played by T and F so far to the other variables of X .
- (3) $p(\psi) < 1$.

Now we argue how these invariants are maintained at the end of the outer loop in Algorithm 1. Invariant (1) is straightforward to see.

▷ **Claim 13.** Invariant (2) is maintained.

Proof. Invariant (2) trivially holds at the beginning.

Each iteration of the first inner loop maintains (2): Say ψ and ζ are at the beginning of the iteration, and ψ' and ζ' denote the formulas after the iteration. Assume (2) holds for ψ and ζ . To see that (2) holds for ψ' and ζ' , consider any assignment to the unplayed variables. We will argue that if ψ' and ζ' are satisfied, then ψ and ζ are satisfied, which implies (by assumption) that φ is satisfied. So suppose ψ' and ζ' are satisfied. Then ψ is satisfied because each clause containing $\ell_i \vee \ell_j$ or containing $\bar{\ell}_i \vee \bar{\ell}_j$ is satisfied due to $(\ell_i \oplus \ell_j)$ being satisfied in ζ' , and each other clause is satisfied since it contains the corresponding clause in ψ' which is satisfied. Also, ζ is satisfied since each of its constraints is also in ζ' which is satisfied.

It is immediate that T’s and F’s “normal” moves in the outer loop maintain (2), because of the way we update ψ and Y .

Each iteration of the second inner loop maintains (2): If an assignment satisfies ψ' and ζ' (after the iteration) then it also satisfies ψ and ζ (at the beginning of the iteration) since T’s move satisfies $(\ell_k \oplus \ell_m)$ – and therefore the assignment satisfies φ . ◁

▷ **Claim 14.** Invariant (3) is maintained.

Proof. Invariant (3) holds at the beginning by the assumption that φ has $< 1.5^k$ clauses (and each clause has potential $1/1.5^k$).

The first inner loop maintains (3) by the following proposition, which we prove later.

► **Proposition 15.** *If `FindZugzwang()` returns (ℓ_i, ℓ_j) , then $p(\psi) \geq p(\psi')$ where ψ and ψ' are the CNFs before and after the execution of `TfoundZugzwang()`.*

The second inner loop does not affect (3). In each outer iteration except the last, T’s and F’s moves from Y maintain (3) by the following proposition, which we prove later.

■ **Algorithm 1** T's winning strategy in (φ, X) .

```

initialize  $\psi \leftarrow \varphi$ ;  $Y \leftarrow X$ ;  $\zeta \leftarrow \{\}$ ;  $Z \leftarrow \{\}$ 
while game is not over do
  while FindZugzwang() returns a pair  $(\ell_i, \ell_j)$  do
     $\perp$  TfoundZugzwang( $\ell_i, \ell_j$ )
    TplayNormal()
    while F picks  $x_k \in Z$  and  $\ell_k \in \{x_k, \bar{x}_k\}$  and assigns  $\ell_k = 1$  do
       $\perp$  TplayZugzwang( $\ell_k$ )
    if  $|Y \cup Z| = 0$  then halt
     $\perp$  FplayNormal()

subroutine FindZugzwang():
   $\perp$  if there exist distinct  $x_i, x_j \in Y$  and  $\ell_i \in \{x_i, \bar{x}_i\}$  and  $\ell_j \in \{x_j, \bar{x}_j\}$  such that (with
    respect to  $\psi, \ell_i, \ell_j$ ):  $a + e \geq \frac{5}{4}(b + d) + \frac{1}{2}(c + f + g + h)$  then return  $(\ell_i, \ell_j)$ 
   $\perp$  return NULL

subroutine TfoundZugzwang( $\ell_i, \ell_j$ ):
   $\perp$  /* T modifies  $\psi$  with the intention to make  $\ell_i \neq \ell_j$  by waiting for F to touch
     $\{x_i, x_j\}$  */
   $\zeta \leftarrow \zeta \cup \{(\ell_i \oplus \ell_j)\}$ ;  $Z \leftarrow Z \cup \{x_i, x_j\}$ ;  $Y \leftarrow Y - \{x_i, x_j\}$ 
  remove from  $\psi$  every clause containing  $\ell_i \vee \ell_j$  or containing  $\bar{\ell}_i \vee \bar{\ell}_j$ 
   $\perp$  remove  $\ell_i, \bar{\ell}_i, \ell_j, \bar{\ell}_j$  from all other clauses of  $\psi$ 

subroutine TplayZugzwang( $\ell_k$ ):
   $\perp$  /* T makes  $\ell_m \neq \ell_k$  */
  T picks  $x_m \in Z$  and  $\ell_m \in \{x_m, \bar{x}_m\}$  such that  $(\ell_k \oplus \ell_m) \in \zeta$  and assigns  $\ell_m = 0$ 
   $\perp$   $\zeta \leftarrow \zeta - \{(\ell_k \oplus \ell_m)\}$ ;  $Z \leftarrow Z - \{x_k, x_m\}$ 

subroutine TplayNormal():
   $\perp$  T picks  $x_i \in Y$  and  $\ell_i \in \{x_i, \bar{x}_i\}$  maximizing  $p(\psi, \ell_i) - p(\psi, \bar{\ell}_i)$  and assigns  $\ell_i = 1$ 
   $\perp$   $\psi \leftarrow \psi[\ell_i = 1]$ ;  $Y \leftarrow Y - \{x_i\}$ 

subroutine FplayNormal():
   $\perp$  F picks  $x_j \in Y$  and  $\ell_j \in \{x_j, \bar{x}_j\}$  and assigns  $\ell_j = 1$ 
   $\perp$   $\psi \leftarrow \psi[\ell_j = 1]$ ;  $Y \leftarrow Y - \{x_j\}$ 

```

31:10 Erdős–Selfridge Theorem for Nonmonotone CNFs

► **Proposition 16.** *If $\text{FindZugzwang}()$ returns NULL, then $p(\psi) \geq p(\psi')$ where ψ is the CNF before $\text{TplayNormal}()$ and ψ' is the CNF after $\text{FplayNormal}()$.*

This concludes the proof of Claim 14. ◁

Now we argue why T wins in the last outer iteration. Right before $\text{TplayNormal}()$, $|Y|$ must be odd by invariant (1), because an even number of variables have been played so far (since T has the first move) and $|X|$ is odd (since T also has the last move) and $|Z|$ is even. Thus, T always has an available move in $\text{TplayNormal}()$ since $|Y| > 0$ at this point. When T is about to play the last variable $x_i \in Y$ (possibly followed by some Z moves in the second inner loop), all remaining clauses in ψ have width ≤ 1 . There cannot be an empty clause in ψ , because then $p(\psi)$ would be $\geq 1/1.5^0 = 1$, contradicting invariant (3). There cannot be more than one clause in ψ , because then $p(\psi)$ would be $\geq 2/1.5^1 \geq 1$. Thus ψ is either empty (already satisfied) or just (x_i) or just (\bar{x}_i) , which T satisfies in one move.

At termination, Y and Z are empty, and ψ and ζ are empty and thus satisfied. By invariant (2), this means φ is satisfied by the gameplay, so T wins.

This concludes the proof of Lemma 12 except Proposition 15 and Proposition 16. ◀

Proof of Proposition 15. Since $\text{FindZugzwang}()$ returns (ℓ_i, ℓ_j) , the following holds with respect to ψ, ℓ_i, ℓ_j :

$$a + e \geq \frac{5}{4}(b + d) + \frac{1}{2}(c + f + g + h) \quad (\spadesuit)$$

We also have

$$p(\psi) - p(\psi') = a + e - \left(\frac{5}{4}(b + d) + \frac{1}{2}(c + f + g + h)\right)$$

because:

- Clauses from the a, e groups are removed (since they contain $\ell_i \vee \ell_j$ or $\bar{\ell}_i \vee \bar{\ell}_j$), so their potential gets multiplied by 0. (Intuitively, T considers these clauses satisfied in advance since she will satisfy $(\ell_i \oplus \ell_j)$ later.)
- Clauses from the b, d groups each shrink by two literals (since they contain two of $\ell_i, \bar{\ell}_i, \ell_j, \bar{\ell}_j$ which are removed), so their potential gets multiplied by $1.5 \cdot 1.5 = 9/4$. (Some of these four literals will eventually get assigned 1, but since T cannot predict which ones, she pessimistically assumes they are all 0.)
- Clauses from the c, f, g, h groups each shrink by one literal (since they contain one of $\ell_i, \bar{\ell}_i, \ell_j, \bar{\ell}_j$ which are removed), so their potential gets multiplied by $1.5 = 3/2$.

Since (\spadesuit) holds, $p(\psi) \geq p(\psi')$. ◀

Proof of Proposition 16. In $\text{TplayNormal}()$, T picks the literal ℓ_i maximizing $p(\psi, \ell_i) - p(\psi, \bar{\ell}_i)$ and plays $\ell_i = 1$.³ In $\text{FplayNormal}()$, F plays $\ell_j = 1$. With respect to ψ, ℓ_i, ℓ_j we have

$$p(\psi) - p(\psi') = a + b + c + d + g - \left(\frac{5}{4}e + \frac{1}{2}(f + h)\right)$$

because:

- Clauses from the a, b, c, d, g groups are satisfied and removed (since they contain $\ell_i = 1$ or $\ell_j = 1$ or both), so their potential gets multiplied by 0.

³ Some other strategies would also work here, but this one is the simplest to analyze.

- Clauses from the e group each shrink by two literals (since they contain $\bar{\ell}_i = 0$ and $\bar{\ell}_j = 0$), so their potential gets multiplied by $1.5 \cdot 1.5 = 9/4$.
- Clauses from the f, h groups each shrink by one literal, so their potential gets multiplied by $1.5 = 3/2$.

By the choice of ℓ_i (i.e., maximizing $p(\ell_i) - p(\bar{\ell}_i)$), we have:

$$\begin{aligned} p(\ell_i) - p(\bar{\ell}_i) &\geq p(\bar{\ell}_j) - p(\ell_j) \\ \implies a + b + c - d - e - f &\geq b + e + h - a - d - g \\ \implies 2a + 0b + 1c + 0d - 2e - 1f + 1g - 1h &\geq 0 \end{aligned} \quad (\clubsuit)$$

Since `FindZugzwang()` returns `NULL`, (\spadesuit) does not hold in ψ . Thus the following holds:

$$\begin{aligned} (a + e) &< \frac{5}{4}(b + d) + \frac{1}{2}(c + f + g + h) \\ \implies -1a + \frac{5}{4}b + \frac{1}{2}c + \frac{5}{4}d - 1e + \frac{1}{2}f + \frac{1}{2}g + \frac{1}{2}h &> 0 \end{aligned} \quad (\diamond)$$

Thus $p(\psi) \geq p(\psi')$ because the linear combination $\frac{9}{16}(\clubsuit) + \frac{1}{8}(\diamond)$ implies:

$$\begin{aligned} &\frac{9}{16}(2a + 0b + 1c + 0d - 2e - 1f + 1g - 1h) + \\ &\frac{1}{8}(-1a + \frac{5}{4}b + \frac{1}{2}c + \frac{5}{4}d - 1e + \frac{1}{2}f + \frac{1}{2}g + \frac{1}{2}h) > 0 \\ \implies 1a + \frac{5}{32}b + \frac{5}{8}c + \frac{5}{32}d - \frac{5}{4}e - \frac{1}{2}f + \frac{5}{8}g - \frac{1}{2}h &> 0 \\ \implies 1a + 1b + 1c + 1d - \frac{5}{4}e - \frac{1}{2}f + 1g - \frac{1}{2}h &> 0 \\ \implies a + b + c + d + g - (\frac{5}{4}e + \frac{1}{2}(f + h)) &> 0 \end{aligned} \quad \blacktriangleleft$$

References

- 1 Lauri Ahlroth and Pekka Orponen. Unordered constraint satisfaction games. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 64–75. Springer, 2012.
- 2 Jesper Byskov. Maker-Maker and Maker-Breaker games are PSPACE-complete. Technical Report RS-04-14, BRICS, Department of Computer Science, Aarhus University, 2004.
- 3 Paul Erdős and John Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14(3), 1973.
- 4 Dan Hefetz, Michael Krivelevich, Miloš Stojaković, and Tibor Szabó. *Positional Games*. Springer, 2014.
- 5 Martin Kutz. *The Angel Problem, Positional Games, and Digraph Roots*. PhD thesis, Freie Universität Berlin, 2004. Chapter 2: Weak Positional Games.
- 6 Martin Kutz. Weak positional games on hypergraphs of rank three. In *Proceedings of the 3rd European Conference on Combinatorics, Graph Theory, and Applications (EuroComb)*, pages 31–36. Discrete Mathematics & Theoretical Computer Science, 2005.
- 7 Md Lutfar Rahman and Thomas Watson. Complexity of unordered CNF games. *ACM Transactions on Computation Theory*, 12(3):18:1–18:18, 2020.
- 8 Md Lutfar Rahman and Thomas Watson. Tractable unordered 3-CNF games. In *Proceedings of the 14th Latin American Theoretical Informatics Symposium (LATIN)*, pages 360–372. Springer, 2020.
- 9 Md Lutfar Rahman and Thomas Watson. 6-uniform Maker-Breaker game is PSPACE-complete. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 57:1–57:15. Schloss Dagstuhl, 2021.
- 10 Thomas Schaefer. Complexity of decision problems based on finite two-person perfect-information games. In *Proceedings of the 8th Symposium on Theory of Computing (STOC)*, pages 41–49. ACM, 1976.
- 11 Thomas Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.

Unit-Disk Range Searching and Applications

Haitao Wang 

Department of Computer Science, Utah State University, Logan, UT, USA

Abstract

Given a set P of n points in the plane, we consider the problem of computing the number of points of P in a query unit disk (i.e., all query disks have the same radius). We show that the main techniques for simplex range searching can be adapted to this problem. For example, by adapting Matoušek's results, we can build a data structure of $O(n)$ space so that each query can be answered in $O(\sqrt{n})$ time; alternatively, we can build a data structure of $O(n^2/\log^2 n)$ space with $O(\log n)$ query time. Our techniques lead to improvements for several other classical problems in computational geometry.

1. Given a set of n unit disks and a set of n points in the plane, the batched unit-disk range counting problem is to compute for each disk the number of points in it. Previous work [Katz and Sharir, 1997] solved the problem in $O(n^{4/3} \log n)$ time. We give a new algorithm of $O(n^{4/3})$ time, which is optimal as it matches an $\Omega(n^{4/3})$ -time lower bound. For small χ , where χ is the number of pairs of unit disks that intersect, we further improve the algorithm to $O(n^{2/3} \chi^{1/3} + n^{1+\delta})$ time, for any $\delta > 0$.
2. The above result immediately leads to an $O(n^{4/3})$ time optimal algorithm for counting the intersecting pairs of circles for a set of n unit circles in the plane. The previous best algorithms solve the problem in $O(n^{4/3} \log n)$ deterministic time [Katz and Sharir, 1997] or in $O(n^{4/3} \log^{2/3} n)$ expected time by a randomized algorithm [Agarwal, Pellegrini, and Sharir, 1993].
3. Given a set P of n points in the plane and an integer k , the distance selection problem is to find the k -th smallest distance among all pairwise distances of P . The problem can be solved in $O(n^{4/3} \log^2 n)$ deterministic time [Katz and Sharir, 1997] or in $O(n \log n + n^{2/3} k^{1/3} \log^{5/3} n)$ expected time by a randomized algorithm [Chan, 2001]. Our new randomized algorithm runs in $O(n \log n + n^{2/3} k^{1/3} \log n)$ expected time.
4. Given a set P of n points in the plane, the discrete 2-center problem is to compute two smallest congruent disks whose centers are in P and whose union covers P . An $O(n^{4/3} \log^5 n)$ -time algorithm was known [Agarwal, Sharir, and Welzl, 1998]. Our techniques yield a deterministic algorithm of $O(n^{4/3} \log^{10/3} n \cdot (\log \log n)^{O(1)})$ time and a randomized algorithm of $O(n^{4/3} \log^3 n \cdot (\log \log n)^{1/3})$ expected time.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Unit disks, disk range searching, batched range searching, distance selection, discrete 2-center, arrangements, cuttings

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.32

Related Version *Full Version:* <https://arxiv.org/abs/2204.08992>

Funding This research was supported in part by NSF under Grant CCF-2005323.

1 Introduction

We consider unit-disk range counting queries. Given a set P of n points in the plane, the problem is to build a data structure so that the number of points of P in D can be computed efficiently for any query unit disk D (i.e., all query disks have the same known radius).

Our problem is a special case of the general disk range searching problem in which each query disk may have an arbitrary radius. Although we are not aware of any previous work particular for our special case, the general problem has been studied before [4, 5, 19, 29, 32]. First of all, it is well-known that the lifting method can reduce the disk range searching



© Haitao Wang;

licensed under Creative Commons License CC-BY 4.0

18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022).

Editors: Artur Czumaj and Qin Xin; Article No. 32; pp. 32:1–32:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the d -dimensional space to half-space range searching in $(d + 1)$ -dimensional space; see, e.g., [20, 32]. For example, using Matoušek’s results in 3D [27], with $O(n)$ space, each disk query in the plane can be answered in $O(n^{2/3})$ time. Using the randomized results for general semialgebraic range searching [5, 29], one can build a data structure of $O(n)$ space in $O(n^{1+\delta})$ expected time that can answer each disk query in $O(\sqrt{n} \log^{O(1)} n)$ time, where (and throughout the paper) δ denotes any small positive constant. For deterministic results, Agarwal and Matoušek’s techniques [4] can build a data structure of $O(n)$ space in $O(n \log n)$ time, and each query can be answered in $O(n^{1/2+\delta})$ time.

A related problem is to report all points of P in a query disk. If all query disks are unit disks, the problem is known as *fixed-radius neighbor problem* in the literature [9, 14, 17, 18]. Chazelle and Edelsbrunner [18] gave an optimal solution (in terms of space and query time): they constructed a data structure of $O(n)$ space that can answer each query in $O(\log n + k)$ time, where k is the output size; their data structure can be constructed in $O(n^2)$ time. By a standard lifting transformation that reduces the problem to the halfspace range reporting queries in 3D, Chan and Tsakalidis [12] constructed a data structure of $O(n)$ space in $O(n \log n)$ time that can answer each query in $O(\log n + k)$ time; the result also applies to the general case where the query disks may have arbitrary radii. Refer to [1, 2, 28] for excellent surveys on range searching.

In this paper, we focus on unit-disk counting queries. By taking advantage of the property that all query disks have the same known radius, we manage to adapt the techniques for simplex range searching to our problem. We show that literally all main results for simplex range searching can be adapted to our problem with asymptotically the same performance. For example, by adapting Matoušek’s result in [26], we build a data structure of $O(n)$ space in $O(n \log n)$ time and each query can be answered in $O(\sqrt{n} \log^{O(1)} n)$ time. By adapting Matoušek’s result in [27], we build a data structure of $O(n)$ space in $O(n^{1+\delta})$ time and each query can be answered in $O(\sqrt{n})$ time. By adapting Chan’s randomized result in [11], we build a data structure of $O(n)$ space in $O(n \log n)$ expected time and each query can be answered in $O(\sqrt{n})$ time with high probability.

In addition, we obtain the following trade-off: After $O(nr)$ space and $O(nr(n/r)^\delta)$ time preprocessing, each query can be answered in $O(\sqrt{n/r})$ time, for any $1 \leq r \leq n/\log^2 n$. In particular, setting $r = n/\log^2 n$, we can achieve $O(\log n)$ query time, using $O(n^2/\log^2 n)$ space and $O(n^2/\log^{2-\delta} n)$ preprocessing time. To the best of our knowledge, the only previous work we are aware of with $O(\log n)$ time queries for the disk range searching problem is a result in [24],¹ which can answer each general disk query in $O(\log n)$ time with $O(n^2 \log n)$ space and preprocessing time.

Probably more interestingly to some extent, our techniques can be used to derive improved algorithms for several classical problems, as follows. Our results are the first progress since the previous best algorithms for these problems were proposed over two decades ago.

Batched unit-disk range counting. Let P be a set of n points and \mathcal{D} be a set of m (possibly overlapping) congruent disks in the plane. The problem is to compute for all disks $D \in \mathcal{D}$ the number of points of P in D . The algorithm of Katz and Sharir [24] solves the problem in $O((m^{2/3}n^{2/3} + m + n) \log n)$ time. By using our techniques for unit-disk range searching and adapting a recent result of Chan and Zheng [13], we obtain a new algorithm of $O(n^{2/3}m^{2/3} + m \log n + n \log m)$ time. We further improve the algorithm so that the complexities are sensitive to χ , the number of pairs of disks of \mathcal{D} that intersect. The runtime of the algorithm is $O(n^{2/3}\chi^{1/3} + m^{1+\delta} + n \log n)$.

¹ See Theorem 3.1 [24]. The authors noted in their paper that the result was due to Pankaj K. Agarwal.

On the negative side, Erickson [21] proved a lower bound of $\Omega(n^{2/3}m^{2/3} + m \log n + n \log m)$ time for the problem in a so-called *partition algorithm model*, even if each disk is a half-plane (note that a half-plane can be considered as a special unit disk of infinite radius). Therefore, our algorithm is optimal under Erickson's model.

Counting intersections of congruent circles. As discussed in [24], the following problem can be immediately solved using batched unit-disk range counting: Given a set of n congruent circles of radius r in the plane, compute the number of intersecting pairs. To do so, define P as the set of the centers of circles and define \mathcal{D} as the set of congruent disks centered at points of P with radius $2r$. Then apply the batched unit-disk range counting algorithm on P and \mathcal{D} . The algorithm runs in $O(n^{4/3})$ time, matching an $\Omega(n^{4/3})$ -time lower bound [21]. To the best of our knowledge, the previous best results for this problem are a deterministic algorithm of $O(n^{4/3} \log n)$ time [24] and a randomized algorithm of $O(n^{4/3} \log^{2/3} n)$ expected time [3]. Agarwal, Pellegrini, and Sharir [6] also studied the problem for circles of different radii and gave an $O(n^{3/2+\delta})$ time deterministic algorithm.

Distance selection. Let P be a set of n points in the plane. Given an integer k in the range $[1, n(n-1)/2]$, the problem is to find the k -th smallest distance among all pairwise distances of P ; let λ^* denote the k -th smallest distance. Given a value λ , the decision problem is to decide whether $\lambda \geq \lambda^*$. We refer to the original problem as the optimization problem.

Chazelle [15] gave the first subquadratic algorithm of $O(n^{9/5} \log^{4/5} n)$ time. Agarwal, Aronov, Sharir, and Suri [3] presented randomized algorithms that solve the decision and optimization problems in $O(n^{4/3} \log^{2/3} n)$ and $O(n^{4/3} \log^{8/3} n)$ expected time, respectively. Goodrich [22] later gave a deterministic algorithm of $O(n^{4/3} \log^{8/3} n)$ time for the optimization problem. Katz and Sharir [24] proposed a deterministic algorithm of $O(n^{4/3} \log n)$ time for the decision problem and used it to solve the optimization problem in $O(n^{4/3} \log^2 n)$ deterministic time. Using the decision algorithm of [3], Chan's randomized technique [10] solved the optimization problem in $O(n \log n + n^{2/3} k^{1/3} \log^{5/3} n)$ expected time.

Our algorithm for the batched unit-disk range counting problem can be used to solve the decision problem in $O(n^{4/3})$ time. Combining it with the randomized technique of Chan [10], the optimization problem can now be solved in $O(n \log n + n^{2/3} k^{1/3} \log n)$ expected time.

Discrete 2-center. Let P be a set of n points in the plane. The discrete 2-center problem is to find two smallest congruent disks whose centers are in P and whose union covers P . Agarwal, Sharir, and Welzl [8] gave an $O(n^{4/3} \log^5 n)$ -time algorithm. Using our techniques for unit-disk range searching, we reduce the time of their algorithm to $O(n^{4/3} \log^{10/3} n (\log \log n)^{O(1)})$ deterministic time or to $O(n^{4/3} \log^3 n (\log \log n)^{1/3})$ expected time by a randomized algorithm.

In the following, we present our algorithms for unit-disk range searching in Section 2. The other problems are discussed in Section 3. Section 4 concludes the paper. Due to the space limit, many proofs are omitted but can be found in the full paper.

2 Unit-disk range searching

In this section, we present our algorithms for the unit-disk range searching problem. Our goal is to show that the main techniques for simplex range searching can be used to solve our problem. In particular, we show that, after overcoming many difficulties, the techniques of Matoušek in [26] and [27] as well as the results of Chan [11] can be adapted to our problem with asymptotically the same performance.

We assume that the radius of unit disks is 1. In the rest of this section, unless otherwise stated, a disk refers to a unit disk. We begin with an overview of our approach.

An overview. We roughly (but not precisely) discuss the main idea. We first implicitly build a grid G of side length $1/\sqrt{2}$ such that any query disk D only intersects $O(1)$ cells of G . This means that it suffices to build a data structure for the subset $P(C')$ of the points of P in each individual cell C' of G with respect to query disks whose centers are in another cell C that is close to C' . A helpful property for processing $P(C')$ with respect to C is that for any two disks with centers in C , their boundary portions in C' cross each other at most once. More importantly, we can define a duality relationship between points in C and disk arcs in C' (and vice versa): a point p in C is dual to the arc of the boundary of D_p in C' , where D_p is the disk centered at p . This duality helps to obtain a *Test Set Lemma* that is crucial to the algorithms in [11, 26, 27]. With these properties and some additional observations, we show that the algorithm for computing cuttings for hyperplanes [16] can be adapted to the disk arcs in C' . With the cutting algorithms and the Test Set Lemma, we show that the techniques in [11, 26, 27] can be adapted to unit-disk range searching for the points of $P(C')$ with respect to the query disks centered in C .

The rest of this section is organized as follows. In Section 2.1, we reduce the problem to problems with respect to pairs of cells (C, C') . Section 2.2 introduces some basic concepts and observations that are fundamental to our approach. We adapt the cutting algorithm of Chazelle [16] to our problem in Section 2.3. Section 2.4 proves the Test Set Lemma. In the subsequent subsections, we adapt the algorithms of [11, 26, 27], whose query times are all $\Omega(\sqrt{n})$ with $O(n)$ space. Section 2.8 presents the trade-offs between the preprocessing and the query time. Section 2.9 finally summarizes all results.

2.1 Reducing the problem to pairs of grid cells

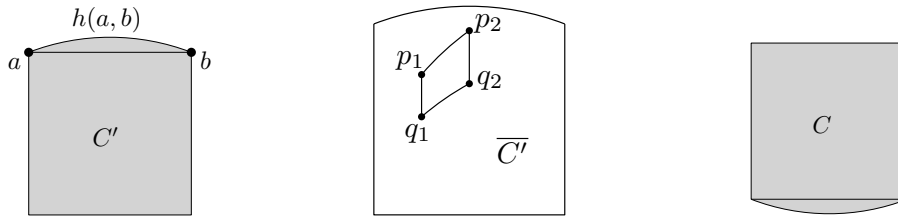
For each point p in the plane, we use $x(p)$ and $y(p)$ to denote its x - and y -coordinates, respectively, and we use D_p to denote the disk centered at p . For any region A in the plane, we use $P(A)$ to denote the subset of points of P in A , i.e., $P(A) = P \cap A$.

We will compute a set \mathcal{C} of $O(n)$ pairwise-disjoint square cells in the plane with the following properties. (1) Each cell has side length $1/\sqrt{2}$. (2) Every two cells are separated by an axis-parallel line. (3) For a disk D_p with center p , if p is not in any cell of \mathcal{C} , then $D_p \cap P = \emptyset$. (4) Each cell C of \mathcal{C} is associated with a subset $N(C)$ of $O(1)$ cells of \mathcal{C} , such that for any disk D with center in C , every point of $P \cap D$ is in one of the cells of $N(C)$. (5) Each cell C' of \mathcal{C} is in $N(C)$ for a constant number of cells $C \in \mathcal{C}$. The following is a key lemma for reducing the problem to pairs of square cells. The proof is in the full paper.

► **Lemma 1.**

1. *The set \mathcal{C} with the above properties, along with the subsets $P(C)$ and $N(C)$ for all cells $C \in \mathcal{C}$, can be computed in $O(n \log n)$ time and $O(n)$ space.*
2. *With $O(n \log n)$ time and $O(n)$ space preprocessing, given any disk with center p , we can determine whether p is in a cell C of \mathcal{C} , and if yes, return the set $N(C)$ in $O(\log n)$ time.*

With Lemma 1 in hand, to solve the unit disk range searching problem, for each cell $C \in \mathcal{C}$ and each cell $C' \in N(C)$, we will preprocess the points of $P(C')$ with respect to the query disks whose centers are in C . Suppose the preprocessing time (resp. space) for each such pair (C, C') is $f(m) = \Omega(m)$, where $m = |P(C')|$. Then, by the property (5) of \mathcal{C} , the total preprocessing time (resp., space) for all such pairs (C, C') is $f(n)$. In the following, we will describe our preprocessing algorithm for (C, C') . Since $N(C) \subset \mathcal{C}$ and the points of P in each cell of \mathcal{C} are already known by Lemma 1, $P(C')$ is available to us. To simplify the



■ **Figure 1** Illustrating $\overline{C'}$, ■ **Figure 2** Illustrating an upper arc pseudo-trapezoid in $\overline{C'}$. ■ **Figure 3** Illustrating \overline{C} , which is the grey region.

notation, we assume that all points of P are in C' , i.e., $P(C') = P$. Note that if $C = C'$, then the problem is trivial because any disk centered in C' covers the entire cell. We thus assume $C \neq C'$. Due to the property (2) of \mathcal{C} , without loss of generality, in the following we assume that C and C' are separated by a horizontal line such that C is below the line.

2.2 Basic concepts and observations

We use \overline{ab} to denote the line segment connecting two points a and b ,. For any compact region A in the plane, let ∂A denote the boundary of A , e.g., if A is a disk, then ∂A is a circle.

Consider a disk D whose center is in C . As the side length of C' is $1/\sqrt{2}$, $\partial D \cap C'$ may contain up to two arcs of the circle ∂D . For this reason, we enlarge C' to a region $\overline{C'}$ so that $\partial D \cap \overline{C'}$ contains at most one arc. The region $\overline{C'}$ is defined as follows (e.g., see Fig. 1).

Let a and b be the two vertices of C' on its top edge. Let D_{ab} be the disk whose center is below \overline{ab} and whose boundary contains both a and b . Let $h(a,b)$ be the arc of ∂D_{ab} above \overline{ab} and connecting a and b . Define $\overline{C'}$ to be the region bounded by $h(a,b)$, and the three edges of C' other than \overline{ab} . As the side length of C' is $1/\sqrt{2}$, for any disk D whose center is in C , $\partial D \cap \overline{C'}$ is either \emptyset or a single arc of ∂D (which is on the upper half-circle of ∂D). Let e_b denote the bottom edge of C' .

Consider a disk D . An arc h on the upper half-circle of ∂D (i.e., the half-circle above the horizontal line through its center) is called an *upper disk arc* (or *upper arc* for short); *lower arcs* are defined symmetrically. Note that an upper arc is *x-monotone*, i.e., each vertical line intersects it at a single point if not empty. If h is an arc of a disk D , then we say that D is the *underlying disk* of h and the center of D is also called the *center* of h . An arc h in $\overline{C'}$ is called a *spanning arc* if both endpoints of h are on $\partial \overline{C'}$. As we are mainly dealing with upper arcs of $\overline{C'}$ whose centers are in C , in the following unless otherwise stated, an upper arc always refers to one whose center is in C . The following is an easy but crucial observation that makes it possible to adapt many techniques for dealing with lines in the plane to spanning upper arcs of $\overline{C'}$.

► **Observation 2.** *Suppose h is an upper arc in $\overline{C'}$, and e is a vertical line segment or an upper arc in $\overline{C'}$. Then, h and e can intersect each other at most once.*

Proof. If e is a vertical segment, since h is *x-monotone*, h and e can intersect each other at most once. If e is an upper arc, since both e and h are upper arcs of disks whose centers are in C and they are both in $\overline{C'}$, they can intersect each other at most once. ◀

Pseudo-trapezoids. Let $h(p_1,p_2)$ be an upper arc with p_1 and p_2 as its left and right endpoints, respectively. Define $h(q_1,q_2)$ similarly, such that $x(p_1) = x(q_1)$ and $x(p_2) = x(q_2)$. Assume that $h(p_1,p_2)$ and $h(q_1,q_2)$ do not cross each other and $h(p_1,p_2)$ is above $h(q_1,q_2)$. The region σ bounded by the two arcs and the two vertical lines $\overline{p_1q_1}$ and $\overline{p_2q_2}$ is called an *upper-arc pseudo-trapezoid* (e.g., see Fig. 2). We call $\overline{p_1q_1}$ and $\overline{p_2q_2}$ the two *vertical sides* of

σ , and call $h(q_1, q_1)$ and $h(p_1, p_2)$ the *top arc* and *bottom arc* of σ , respectively. The region σ is also considered as an upper-arc pseudo-trapezoid if the bottom arc $h(q_1, q_2)$ is replaced by a line segment $\overline{q_1 q_2}$ on e_b (for simplicity, we still refer to $\overline{q_1 q_2}$ as the bottom-arc of σ). In this way, $\overline{C'}$ itself is an upper-arc pseudo-trapezoid. Note that for any pseudo-trapezoid σ in $\overline{C'}$ and a disk D centered in C , $\partial D \cap \sigma$ is either empty or an upper arc.

The counterparts of C (with respect to C'). The above definitions in C' (with respect to C) have counterparts in C (with respect to C') with similar properties. First, we define \overline{C} in a symmetric way as $\overline{C'}$, i.e., a lower arc connecting the two bottom vertices of C' is used to bound $\partial \overline{C}$; e.g., see Fig. 3. Also, we define *lower-arc pseudo-trapezoids* and *spanning lower arcs* similarly, and unless otherwise stated, a lower arc in \overline{C} refer to one whose center is in C' . In the following, unless otherwise stated, properties, algorithms, and observations for the concepts of C' with respect to C also hold for their counterparts of C with respect to C' .

Duality. We define a duality relationship between upper arcs in $\overline{C'}$ and points in C . For an upper arc h in $\overline{C'}$, we consider its center as its *dual point* in C . For a point $q \in C$, we consider the upper arc $\partial D_q \cap \overline{C'}$ as its *dual arc* in $\overline{C'}$ if it is not empty. Similarly, we define duality relationship between lower arcs in \overline{C} and points in C' . Note that if the boundary of a disk centered at a point $p \in P$ does not intersect \overline{C} , then the point p can be ignored from P in our preprocessing because among all disks centered in C one disk contains p if and only if all other disks contain p . Henceforth, without loss of generality, we assume that ∂D_p intersects \overline{C} for all points $p \in P$, implying that every point of P is dual to a lower arc in \overline{C} . Note that our duality is similar in spirit to the duality introduced by Agarwal and Sharir [7] between points and pseudo-lines.

2.3 Computing hierarchical cuttings for disk arcs

Let H be a set of n spanning upper arcs in $\overline{C'}$. For a compact region A of $\overline{C'}$, we use H_A to denote the set of arcs of H that intersect the relative interior of A . By adapting its definition for hyperplanes, e.g., [16, 27], a *cutting* for H is a collection Ξ of closed cells (each of which is an upper-arc pseudo-trapezoid) with disjoint interiors, which together cover the entire $\overline{C'}$. The *size* of Ξ is the number of cells in Ξ . For a parameter $1 \leq r \leq n$, a $(1/r)$ -*cutting* for H is a cutting Ξ satisfying $|H_\sigma| \leq n/r$ for every cell $\sigma \in \Xi$.

We will adapt the algorithm of Chazelle [16] to computing a $(1/r)$ -cutting of size $O(r^2)$ for H . It is actually a sequence of *hierarchical cuttings*. Specifically, we say that a cutting Ξ' *c-refines* a cutting Ξ if every cell of Ξ' is contained in a single cell of Ξ and every cell of Ξ contains at most c cells of Ξ' . Let $\Xi_0, \Xi_1, \dots, \Xi_k$ be a sequence of cuttings such that Ξ_0 consists of the single cell $\overline{C'}$ (recall that $\overline{C'}$ itself is an upper arc pseudo-trapezoid), and every Ξ_i is a $(1/\rho^i)$ -cutting of size $O(\rho^{2i})$ which *c-refines* Ξ_{i-1} , for two constants ρ and c . In order to make Ξ_k a $(1/r)$ -cutting, we set $k = \lceil \log_\rho r \rceil$. The above sequence of cuttings is called a *hierarchical $(1/r)$ -cutting* of H . If a cell $\sigma \in \Xi_{j-1}$ contains a cell $\sigma' \in \Xi_j$, we say that σ is the *parent* of σ' and σ' is a *child* of σ . Hence, one could view Ξ as a tree structure with Ξ_0 as the root.

We have the following theorem.

► **Theorem 3 (The Cutting Theorem).** *Let χ denote the number of intersections of the arcs of H . For any $r \leq n$, a hierarchical $(1/r)$ -cutting of size $O(r^2)$ for H (together with the sets H_σ for every cell σ of Ξ_i for all $0 \leq i \leq k$) can be computed in $O(nr)$ time; more specifically, the size of the cutting is bounded by $O(r^{1+\delta} + \chi \cdot r^2/n^2)$ and the running time of the algorithm is bounded by $O(nr^\delta + \chi \cdot r/n)$.*

► **Remark.** For a set of lower arcs in \overline{C} , we can define cuttings similarly with lower-arc pseudo-trapezoids as cells; the same result as Theorem 3 also holds for computing lower-arc cuttings. Also note that the algorithm is optimal if the subsets H_σ 's need to be computed.

To prove Theorem 3, we adapt Chazelle's algorithm for computing cuttings for hyperplanes [16]. It was stated in [7] that Chazelle's algorithm can be extended to compute such a cutting of size $O(r^{1+\delta} + \chi \cdot r^2/n^2)$ in $O(n^{1+\delta} + \chi \cdot r/n)$ time. However, no details were provided in [7]. For completeness and also for helping the reader to better understand our cutting, we present the algorithm details in the full paper, where we actually give a more general algorithm that also works for other curves in the plane (e.g., circles or circular arcs of different radii, pseudo-lines, line segments, etc.). Note that our result reduces the factor $n^{1+\delta}$ in the above time complexity of [7] to nr^δ .

The weighted case. To adapt the simplex range searching algorithms in [11, 26, 27], we will need to compute cuttings for a weighted set H of spanning upper arcs in $\overline{C'}$, where each arc $h \in H$ has a nonnegative weight $w(h)$. The hierarchical $(1/r)$ -cutting can be naturally generalized to the weighted case (i.e., the interior of each pseudo-trapezoid in a $(1/r)$ -cutting can be intersected by upper arcs of H of total weight at most $w(H)/r$, where $w(H)$ is the total weight of all arcs of H). By a method in [25], any algorithm computing a hierarchical $(1/r)$ -cutting for a set of hyperplanes can be converted to the weighted case with only a constant factor overhead. We can use the same technique to extend any algorithm computing a hierarchical $(1/r)$ -cutting for a set of upper arcs to the weighted case.

2.4 Test Set Lemma

A critical component in all simplex range searching algorithms in [11, 26, 27] is a Test Set Lemma. Using the duality, we obtain a similar result for our problem in the following lemma, whose proof is in the full paper. For any pseudo-trapezoid σ in $\overline{C'}$, we say that an upper arc h *crosses* σ if h intersects the interior of σ .

► **Lemma 4 (Test Set Lemma).** *For any parameter $r \leq n$, there exists a set Q of at most r spanning upper arcs in $\overline{C'}$, such that for any collection Π of interior-disjoint upper-arc pseudo-trapezoids in $\overline{C'}$ satisfying that each pseudo-trapezoid contains at least $n/(c \cdot r)$ points of P for some constant $c > 0$, the following holds: if κ is the maximum number of pseudo-trapezoids of Π crossed by any upper arc of Q , then the maximum number of pseudo-trapezoids of Π crossed by any upper arc in $\overline{C'}$ is at most $O(\kappa + \sqrt{r})$.*

With our Cutting Theorem and the Test Set Lemma, we proceed to adapt the simplex range searching algorithms in [11, 26, 27] to our problem in the following subsections.

2.5 A data structure based on pseudo-trapezoidal partitions

We first extend the simplicial partition for hyperplanes in [26] to our problem, which we rename *pseudo-trapezoidal partition*. A *pseudo-trapezoidal partition* for P is a collection $\Pi = \{(P_1, \sigma_1), \dots, (P_m, \sigma_m)\}$, where the P_i 's are pairwise disjoint subsets forming a partition of P , and each σ_i is a relatively open upper-arc pseudo-trapezoid in $\overline{C'}$ containing all points of P_i . The pseudo-trapezoidal partition we will compute has the following additional property: $\max_{1 \leq i \leq m} |P_i| < 2 \cdot \min_{1 \leq i \leq m} |P_i|$, i.e., all subsets have roughly the same size. Note that the trapezoids σ_i 's may overlap. The subsets P_i 's are called *classes* of Π .

For any upper arc h in $\overline{C'}$, we define its *crossing number* with respect to Π as the number of pseudo-trapezoids of Π crossed by h . The crossing number of Π is defined as the maximum crossing numbers of all upper arcs h in $\overline{C'}$. The following Theorem 5 corresponds

to Theorem 3.1 [26]. Its proof, which is in the full paper, is similar to Theorem 3.1 in [26], with our Test Set Lemma and our Cutting Theorem. Note that similar result as the theorem is already known for pseudo-lines with respect to points [7].

► **Theorem 5 (Partition Theorem).** *Let s be an integer $2 \leq s < n$ and $r = n/s$. There exists a pseudo-trapezoidal partition Π for P , whose classes P_i satisfy $s \leq |P_i| < 2s$, and whose crossing number is $O(\sqrt{r})$.*

Lemma 6, which corresponds to Theorem 4.7(i) [26], computes a pseudo-trapezoidal partition and will be used in the algorithm for Theorem 7. The proof is in the full paper.

► **Lemma 6.** *For any fixed $\delta > 0$, if $s \geq n^\delta$, then a pseudo-trapezoidal partition as in the Partition Theorem (whose classes $|P_i|$ satisfy $s \leq |P_i| < 2s$ and whose crossing number is $O(\sqrt{r})$) can be constructed in $O(n \log r)$ time, where $r = n/s$.*

Using Lemma 6, we can obtain the following theorem, whose proof is in the full paper.

► **Theorem 7.** *A data structure of $O(n)$ space can be built in $O(n \log n)$ time, so that given a disk D centered in C , the number of points of P in D can be computed in $O(\sqrt{n}(\log n)^{O(1)})$ time.*

► **Remark.** It is easy to modify the algorithm to answer the *outside-disk queries*: compute the number of points of P outside any query disk, with asymptotically the same complexities. This is also the case for other data structures given later, e.g., Theorems 8, 9, 10.

2.6 A data structure based on hierarchical cuttings

Using our Cutting Theorem and the Test Set Lemma, we can adapt the techniques of Matoušek [27] to our problem. We have the following theorem, whose proof is in the full paper.

► **Theorem 8.** *We can build an $O(n)$ space data structure for P in $O(n^{1+\delta})$ time for any small constant $\delta > 0$, such that given any disk D whose center is in C , the number of points of P in D can be computed in $O(\sqrt{n})$ time.*

2.7 A randomized result

We have the following theorem by adapting the randomized result of Chan [11].

► **Theorem 9.** *We can build an $O(n)$ space data structure for P in $O(n \log n)$ expected time by a randomized algorithm, such that given any disk D whose center is in C , the number of points of P in D can be computed in $O(\sqrt{n})$ time with high probability.*

The data structure is a partition tree, denoted by T , obtained by recursively subdividing $\overline{C'}$ into cells each of which is an upper-arc pseudo-trapezoid. Each node v of T corresponds to a cell, denoted by σ_v . If v is the root, then σ_v is $\overline{C'}$. If v is not a leaf, then v has $O(1)$ children whose cells form a disjoint partition of σ_v . Define $P_v = P \cap \sigma_v$. The set P_v is not explicitly stored at v unless v is a leaf, in which case $|P_v| = O(1)$. The cardinality $|P_v|$ is stored at v . The height of T is $O(\log n)$. If κ is the maximum number of pseudo-trapezoids of T that are crossed by any upper arc in $\overline{C'}$, then $\kappa = O(\sqrt{n})$ holds with high probability. The partition tree T can be built by a randomized algorithm of $O(n \log n)$ expected time. The space of T is $O(n)$. More details for Theorem 9 are in the full paper.

2.8 Trade-offs

Using cuttings and the results of Theorems 8 and 9, we can obtain the following trade-offs between preprocessing and query time by standard techniques [2,27]. The proof is in the full paper.

► **Theorem 10.**

1. We can build an $O(nr)$ space data structure for P in $O(nr(n/r)^\delta)$ time, such that given any query disk D whose center is in C , the number of points of P in D can be computed in $O(\sqrt{n/r})$ time, for any $1 \leq r \leq n/\log^2 n$.
2. We can build an $O(nr)$ space data structure for P in $O(nr \log(n/r))$ expected time, such that given any query disk D whose center is in C , the number of points of P in D can be computed in $O(\sqrt{n/r})$ time with high probability, for any $1 \leq r \leq n/\log^2 n$.

In particular, for the large space case, i.e., $r = n/\log^2 n$, we can obtain the following corollary by Theorem 10(1) (a randomized result with slightly better preprocessing time can also be obtained by Theorem 10(2)).

► **Corollary 11.** We can build an $O(n^2/\log^2 n)$ space data structure for P in $O(n^2/\log^{2-\delta} n)$ time, such that given any query disk D whose center is in C , the number of points of P in D can be computed in $O(\log n)$ time.

2.9 Wrapping things up

All above results on P are for a pair of cells (C, C') such that all points of P are in C' and centers of query disks are in C . Combining the above results with Lemma 1, we can obtain our results for the general case where points of P and query disk centers can be anywhere in the plane. The proof of Corollary 12 summarizes the overall algorithm.

► **Corollary 12.** We have the following results for the unit-disk range counting problem.

1. An $O(n)$ space data structure can be built in $O(n \log n)$ time, with $O(\sqrt{n}(\log n)^{O(1)})$ query time.
2. An $O(n)$ space data structure can be built in $O(n^{1+\delta})$ time for any small constant $\delta > 0$, with $O(\sqrt{n})$ query time.
3. An $O(n)$ space data structure can be built in $O(n \log n)$ expected time by a randomized algorithm, with $O(\sqrt{n})$ query time with high probability.
4. An $O(n^2/\log^2 n)$ space data structure can be built in $O(n^2/\log^{2-\delta} n)$ time, with $O(\log n)$ query time.
5. An $O(nr)$ space data structure can be built in $O(nr(n/r)^\delta)$ time, with $O(\sqrt{n/r})$ query time, for any $1 \leq r \leq n/\log^2 n$.
6. An $O(nr)$ space data structure can be built in $O(nr \log(n/r))$ expected time by a randomized algorithm, with $O(\sqrt{n/r})$ query time with high probability, for any $1 \leq r \leq n/\log^2 n$.

Proof. In the preprocessing, we compute the information and data structure in Lemma 1, which takes $O(n \log n)$ time and $O(n)$ space. For each pair of cells (C, C') with $C \in \mathcal{C}$ and $C' \in N(C)$, we construct the data structure on $P(C')$, i.e., $P \cap C'$, with respect to query disks centered in C , e.g., those in Theorems 7, 8, 9, and 10. As discussed before, due to property (5) of \mathcal{C} , the total preprocessing time and space is the same as those in the above theorems. Given a query disk D with center q , by Lemma 1(2), we determine whether q is in a cell C of \mathcal{C} in $O(\log n)$ time. If no, then $D \cap P = \emptyset$ and thus we simply return 0. Otherwise, the data structure returns $N(C)$. Then, for each $C' \in N(C)$, we use the data structure constructed for (C, C') to compute $|P(C') \cap D|$. We return $|P \cap D| = \sum_{C' \in N(C)} |P(C') \cap D|$. As $|N(C)| = O(1)$, the total query time is as stated in the above theorems. ◀

► Remark. As in [11, 26, 27], all above results can be extended to the weighted case (or the more general semigroup model) where each point of P has a weight.

3 Applications

In this section, we show that our techniques for the disk range searching problem can be used to solve several other problems. More specifically, our techniques yield improved results for three classical problems: batched range counting, distance selection, and discrete 2-center.

3.1 Batched unit-disk range counting

Let P be a set of n points and \mathcal{D} be a set of m (possibly overlapping) unit disks in the plane. The *batched unit-disk range counting* problem (also referred to as *offline range searching* in the literature) is to compute for each disk $D \in \mathcal{D}$ the number of points of P in D .

Let Q denote the set of centers of the disks of \mathcal{D} . For each point $q \in Q$, we use D_q to denote the unit disk centered at q .

We first apply Lemma 1 on P . For each point $q \in Q$, by Lemma 1(2), we first determine whether q is in a cell C of \mathcal{C} . If no, then D_q does not contain any point of P and thus it can be ignored for the problem; without loss of generality, we assume that this case does not happen to any disk of \mathcal{D} . Otherwise, let C be the cell of \mathcal{C} that contains q . By Lemma 1(2), we further find the set $N(C)$ of C . In this way, in $O((n+m)\log n)$ time, we can compute $Q(C)$ for each cell C of \mathcal{C} , where $Q(C)$ is the subset of points of Q in C . Define $\mathcal{D}(C)$ as the set of disks of \mathcal{D} whose centers are in $Q(C)$. Let $P(C) = P \cap C$.

In what follows, we will consider the problem for $P(C')$ and $\mathcal{D}(C)$ for each pair (C, C') of cells with $C \in \mathcal{C}$ and $C' \in N(C)$. Combining the results for all such pairs leads to the result for P and \mathcal{D} (the details on this will be discussed later). To simplify the notation, we assume that $P(C') = P$ and $\mathcal{D}(C) = \mathcal{D}$ (thus $Q(C) = Q$). Hence, our goal is to compute $|P \cap D|$ for all disks $D \in \mathcal{D}$.

If $C = C'$, then all points of P are in D for each disk $D \in \mathcal{D}$ and thus the problem is trivial. Below we assume $C \neq C'$. Without loss of generality, we assume that C' and C are separated by a horizontal line and C' is above the line. We assume that each point of P defines a lower arc in \overline{C} since otherwise the point can be ignored. We also assume that the boundary of each disk of \mathcal{D} intersects $\overline{C'}$, i.e., each point q of Q is dual to an upper arc h_q in $\overline{C'}$, since otherwise the disk can be ignored. Observe that a point p is in D_q if and only if p is below the upper arc h_q (we say that p is below h_q if p is below the upper half boundary of D_q), for any $p \in P$ and $q \in Q$. Hence, the problem is equivalent to computing the number of points of P below each upper arc of H , where $H = \{h_q \mid q \in Q\}$.

Given a set of n points and a set of m lines in the plane, Chan and Zheng [13] recently gave an $O(m^{2/3}n^{2/3} + n \log m + m \log n)$ time algorithm to compute the number of points below each line (alternatively, compute the number of points inside the lower half-plane bounded by each line). We can easily adapt their algorithm to solve our problem. Indeed, the main techniques of Chan and Zheng's algorithm we need to adapt to our problem are the hierarchical cuttings and duality. Using our Cutting Theorem and our definition of duality, we can apply the same technique and solve our problem in $O(m^{2/3}n^{2/3} + n \log m + m \log n)$ time, with $n = |P|$ and $m = |H| = |\mathcal{D}|$. Thus we have the following theorem; the proof is in the full paper.

► **Theorem 13.** *We can compute, for all disks $D \in \mathcal{D}$, the number of points of P in D in $O(m^{2/3}n^{2/3} + n \log m + m \log n)$, with $n = |P|$ and $m = |\mathcal{D}|$.*

Let χ denote the number of intersections of the arcs of H , and thus $\chi = O(m^2)$. Using our Cutting Theorem and Theorem 13, we further improve the algorithm for small χ .

► **Theorem 14.** *We can compute, for all disks $D \in \mathcal{D}$, the number of points of P in D in $O(n^{2/3}\chi^{1/3} + m^{1+\delta} + n \log n)$ time, with $n = |P|$ and $m = |\mathcal{D}|$.*

Proof. We start with computing a hierarchical $(1/r)$ -cutting Ξ_0, \dots, Ξ_k for H , where $r = \min\{m/8, (m^2/\chi)^{1/(1-\delta)}\}$ and δ refers to the parameter in the Cutting Theorem. By our Cutting Theorem, the size of the cutting, denoted by K , is bounded by $O(r^\delta + \chi \cdot r^2/m^2)$ and the time for computing the cutting is $O(mr^\delta + \chi \cdot r/m)$. Since the parameter r depends on χ , which is not available to us, we can overcome the issue by using the standard trick of doubling. More specifically, initially we set χ to a constant. Then we run the algorithm until it exceeds the running time specified based on the guessed value of χ . Next, we double the value χ and run the algorithm again. We repeat this process until when the algorithm finishes before it reaches the specified running time for a certain value of χ . In this way, we run the cutting construction algorithm at most $O(\log \chi)$ time. Therefore, the total time for constructing the desired cutting is $O((mr^\delta + \chi \cdot r/m) \log \chi)$.

Next, we reduce the problem into $O(K)$ subproblems and then solve each subproblem by Theorem 13, which will lead to the theorem.

For each point $p \in P$, we find the cell σ of Ξ_i that contains p and we store p in a *canonical subset* $P(\sigma)$ of P (which is initially \emptyset), for all $0 \leq i \leq k$, i.e., $P(\sigma) = P \cap \sigma$; in fact, we only need to store the cardinality of $P(\sigma)$. For ease of exposition, we assume that no point of P lies on the boundary of any cell of Ξ_i for any i .

For each disk $D \in \mathcal{D}$, our goal is to compute the number of points of P in D , denoted by n_D . We process D as follows. We initialize $n_D = 0$. Let h be the upper arc of H defined by D , i.e., $h = \partial D \cap \overline{C'}$. Starting from $\Xi_0 = \overline{C'}$. Suppose σ is a cell of Ξ_i crossed by h (initially, $i = 0$ and σ is $\overline{C'}$) and $i < k$. For each child cell σ' of σ in Ξ_{i+1} , if σ' is contained in D , then we increase n_D by $|P(\sigma')|$ because all points of $P(\sigma')$ are contained in D . Otherwise, if h crosses σ' , then we proceed on σ' . In this way, the points of $P \cap D$ not counted in n_D are those contained in cells $\sigma \in \Xi_k$ that are crossed by h . To count those points, we perform further processing as follows.

For each cell σ in Ξ_k , if $|P_\sigma| > n/K$, then we arbitrarily partition $P(\sigma)$ into subsets of size between $n/(2K)$ and n/K , called *standard subsets* of $P(\sigma)$. As Ξ_k has $O(K)$ cells and $|P| = n$, the number of standard subsets of all cells of Ξ_k is $O(K)$. Denote by \mathcal{D}_σ the subset of disks of \mathcal{D} whose boundaries cross σ . Our problem is to compute for all disks $D \in \mathcal{D}_\sigma$ the number of points of $P(\sigma)$ contained in D , for all cells $\sigma \in \Xi_k$. To this end, for each cell σ of Ξ_k , for each standard subset $P'(\sigma)$ of $P(\sigma)$, we solve the batched unit-disk range counting problem on the point set $P'(\sigma)$ and the disk set \mathcal{D}_σ by Theorem 13. Note that $|\mathcal{D}_\sigma| \leq m/r$. As Ξ_k has $O(K)$ cells, we obtain $O(K)$ subproblems of size $(n/K, m/r)$ each. As discussed above, solving these subproblems also solves our original problem. It remains to analyze the time complexity of the algorithm, which can be found in the full paper. ◀

The general problem. The above results are for the case where points of P are in the square cell C' while centers of \mathcal{D} are all in C . For solving the general problem where both P and \mathcal{D} can be anywhere in the plane, as discussed before, we reduce the problem to the above case by Lemma 1. The properties of the set \mathcal{C} guarantee that the complexities for the general problem are asymptotically the same as those in Theorem 13. To see this, we consider all pairs (C, C') with $C \in \mathcal{C}$ and $C' \in N(C)$. For the i -th pair (C, C') , let $n_i = |P(C')|$ and $m_i = |\mathcal{D}(C)|$. Then, solving the problem for the i -th pair (C, C') takes $O(n_i^{2/3} m_i^{2/3} + m_i \log n_i + n_i \log m_i)$

time by Theorem 13. Due to the properties (4) and (5) of \mathcal{C} , $\sum_i n_i = O(n)$ and $\sum_i m_i = O(m)$. Therefore, by Hölder's Inequality, $\sum_i n_i^{2/3} m_i^{2/3} \leq n^{1/3} \cdot \sum_i n_i^{1/3} m_i^{2/3} \leq n^{2/3} m^{2/3}$, and thus the total time for solving the problem for all pairs of cells is $O(n^{2/3} m^{2/3} + m \log n + n \log m)$. Similarly, the complexity of Theorem 14 also holds for the general problem, with χ as the number of pairs of disks of \mathcal{D} that intersect.

Computing incidences between points and circles. It is easy to modify the algorithm to solve the following problem: Given n points and m unit circles in the plane, computing (either counting or reporting) the incidences between points and unit circles. The runtime is $O(n^{2/3} m^{2/3} + m \log n + n \log m)$ or $O(n^{2/3} \chi^{1/3} + m^{1+\delta} + n \log n)$, where χ is the number of intersecting pairs of the unit circles. Although the details were not given, Agarwal and Sharir [7] already mentioned that an $n^{2/3} m^{2/3} 2^{O(\log^*(n+m))} + O((m+n) \log(m+n))$ time algorithm can be obtained by adapting Matoušek's technique [27]. (The same problem for circles of arbitrary radii is considered in [7]. Refer to [30] for many other incidence problems.) Our result further leads to an $O(n^{4/3})$ -time algorithm for the *unit-distance detection* problem: Given n points in the plane, is there a pair of points at unit distance? Erickson [21] gave a lower bound of $\Omega(n^{4/3})$ time for the problem in his partition algorithm model.

3.2 The distance selection problem

Given a set P of n points in the plane and an integer k in the range $[0, n(n-1)/2]$, the distance selection problem is to compute the k -th smallest distance among the distances of all pairs of points of P . Let λ^* denote the k -th smallest distance to be computed. Given a value λ , the *decision problem* is to decide whether $\lambda \geq \lambda^*$. Using our batched range counting algorithm, we can easily obtain the following lemma.

► **Lemma 15.** *Given a value λ , whether $\lambda \geq \lambda^*$ can be decided in $O(n^{4/3})$ time.*

Proof. We can use our algorithm for the batched unit-disk range counting problem. Indeed, let \mathcal{D} be the set of congruent disks centered at the points of P with radius λ . By Theorem 13, we can compute in $O(n^{4/3})$ time the cardinality $|\Pi|$, where Π is the set of all disk-point incidences (D, p) , where $D \in \mathcal{D}$, $p \in P$, and D contains p . Observe that for each pair of points (p_i, p_j) of P whose distance is at most λ , it introduces two pairs in Π . Also, each point p_i introduces one pair in Π because p_i is contained in the disk of \mathcal{D} centered at p_i . Hence, the number of pairs of points of P whose distances are at most λ is equal to $(|\Pi| - n)/2$. Clearly, $\lambda \geq \lambda^*$ if and only if $(|\Pi| - n)/2 \geq k$. ◀

Plugging Lemma 15 into a randomized algorithm of Chan [10] (i.e., Theorem 5 [10]), λ^* can be computed in $O(n \log n + n^{2/3} k^{1/3} \log n)$ expected time.

3.3 The discrete 2-center problem

Let P be a set of n points in the plane. The discrete 2-center problem is to find two smallest congruent disks whose centers are in P and whose union covers P . Let λ^* be the radius of the disks in an optimal solution. Given λ , the *decision problem* is to decide whether $\lambda \geq \lambda^*$.

Agarwal, Sharir, and Welzl [8] gave an $O(n^{4/3} \log^5 n)$ time algorithm by solving the decision problem first. A *key subproblem* in their decision algorithm [8] is: Preprocess P to compute a collection \mathcal{P} of *canonical subsets* of P , $\{P_1, P_2, \dots\}$, so that given a query point p in the plane, the set F_p of points of P outside the unit disk centered at p can be represented as the union of a sub-collection \mathcal{P}_p of canonical subsets and \mathcal{P}_p can be found efficiently (it suffices to give the “names” of the canonical subsets of \mathcal{P}_p). Note that here the radius of unit disks is λ .

Roughly speaking, suppose we can solve the above key subproblem with preprocessing time T such that $\sum_{P_i \in \mathcal{P}} |P_i| = M$ and $|\mathcal{P}_p|$ for any query point p is bounded by $O(\tau)$ (and $|\mathcal{P}_p|$ can be found in $O(\tau)$ time); then the algorithm of [8] can solve the decision problem in $O(T + M \log n + \tau \cdot n \log^3 n)$ time. The optimal radius λ^* can thus be found by binary search on all pairwise distances of P (in each iteration, find the k -th smallest distance using a distance selection algorithm); the total time is $O((T_1 + T_2) \log n)$, where T_1 is the time of the distance selection algorithm and T_2 is the time of the decision algorithm.

Note that the logarithmic factor of $M \log n$ in the above running time of the decision algorithm of [8] is due to that for each canonical subset $P_i \in \mathcal{P}$, we need to compute the common intersection of all unit disks centered at the points of P_i , which takes $O(|P_i| \log n)$ time [23]. However, if all points of P_i are sorted (e.g., by x -coordinate or y -coordinate), then the common intersection can be computed in $O(|P_i|)$ time [31]. Therefore, if we can guarantee that all canonical subsets are sorted, then the runtime of the decision algorithm of [8] can be bounded by $O(T + M + \tau \cdot n \log^3 n)$.

Using our techniques for unit-disk range searching, we present new solutions to the above key subproblem. We show that after $T = O(n^{4/3} \log^2 n (\log \log n)^{1/3})$ expected time preprocessing by a randomized algorithm, we can compute $M = O(n^{4/3} \log^2 n / (\log \log n)^{2/3})$ sorted canonical subsets of P so that $\tau = O(n^{1/3} (\log \log n)^{1/3} / \log n)$ holds with high probability. Consequently, the decision problem can be solved in $O(n^{4/3} \log^2 n (\log \log n)^{1/3})$ expected time, and thus λ^* can be computed in $O(n^{4/3} \log^3 n (\log \log n)^{1/3})$ expected time if we use the $O(n^{4/3} \log^2 n)$ time distance selection algorithm in [24]. We also have another slightly slower deterministic result. After $T = O(n^{4/3} \log^{7/3} n (\log \log n)^{1/3})$ time preprocessing algorithm, we can compute $M = O(n^{4/3} \log^{7/3} n / (\log \log n)^{2/3})$ sorted canonical subsets of P so that $\tau = O(n^{1/3} (\log \log n)^{O(1)} / \log^{2/3} n)$. Consequently, the decision problem can be solved in $O(n^{4/3} \log^{7/3} n (\log \log n)^{O(1)})$ time, and thus λ^* can be computed in $O(n^{4/3} \log^{10/3} n (\log \log n)^{O(1)})$ time.

► **Remark.** It is straightforward to modify our algorithms to achieve the same results for the following *inside-disk* problem: represent the subset of points of P *inside* D as a collection of pairwise-disjoint canonical sets for any query disk D .

In what follows, we present our solutions to the above subproblem. We apply Lemma 1 on the set P to compute the set \mathcal{C} of square cells. As before, we first reduce the problem to the same problem with respect to pairs of cells (C, C') of \mathcal{C} , by using Lemma 1 as well as the following lemma (whose proof is in the full paper, by modifying of the algorithm for Lemma 1); then we will solve the problem using our techniques for disk range searching.

► **Lemma 16.** *We can compute in $O(n \log n)$ time a collection of $O(n)$ sorted canonical subsets of P whose total size is $O(n \log n)$, such that for any cell C of \mathcal{C} , there are $O(\log n)$ pairwise-disjoint canonical subsets whose union consists of the points of P that are not in the cells of $N(C)$, and we can find those canonical subsets in $O(\log n)$ time.*

Let D_p be the unit disk centered at a point p in the plane. If p is not in any cell of \mathcal{C} , then $D_p \cap P = \emptyset$ and thus we can return the entire set P as a canonical subset. Henceforth, we only consider the case where p is in a cell C of \mathcal{C} . According to Lemma 16, it suffices to find canonical subsets to cover all points of $P \cap C'$ not in D_p for all cells $C' \in N(C)$. As $|N(C)| = O(1)$, it suffices to consider one such cell $C' \in N(C)$. Hence, as before, the problem reduces to a pair of square cells (C, C') of \mathcal{C} with $C' \in N(C)$. If $C' = C$, then we know that all points of $P \cap C'$ are in D_p . Hence, we assume that $C' \neq C$. Without loss of generality, we assume that C' and C are separated by a horizontal line and C is below the line. The problem is to process all points of $P \cap C'$, such that given any query disk D_p whose center p is in C , we can find a collection of disjoint canonical subsets whose union is the set of points of $P \cap C'$ not in D_p . To simplify the notation, we assume that all n points of P are in C' .

Our data structure combines some techniques for the disk range searching problem. As remarked before, all our results on disk range searching with respect to (C, C') can be applied to find the number of points of P outside any query disk D whose center is in C (indeed, the disk D defines a spanning upper arc h in $\overline{C'}$, and points in D lie on one side of h while points outside D lie on the other side of h). Hence, our main idea is to examine our disk range searching data structures and define canonical subsets of P in these data structures. For each query disk D , we apply the query algorithm on D , which will produce a collection of canonical subsets. The crux is to carefully design the disk range searching data structure (e.g., by setting parameters to some appropriate values) so that the following are as small as possible (tradeoffs are needed): the preprocessing time, the total size of all canonical subsets of the data structure, which is M , and the total number of canonical subsets for each query disk D , which is τ . In the following, whenever we say “apply our query algorithm on D ”, we mean “finding points outside D ”. We will present two results, a randomized result based on Chan’s partition trees [11] and a slightly slower deterministic result.

3.3.1 The randomized result

Our data structure has three levels. We will present them from the lowest level to the highest one. We start with the lowest level in the following lemma, which relies on the partition tree T built in Theorem 9. The proof is in the full paper. For any disk D , we use $P \setminus D$ to refer to the subset of the points of P not in D .

► **Lemma 17.** *We can compute in $O(n \log n)$ expected time a data structure with $O(n)$ sorted canonical subsets of P whose total size is $O(n \log n)$, so that for any disk D whose center is in C , we can find in $O(\kappa)$ time $O(\kappa)$ pairwise-disjoint canonical sets whose union is $P \setminus D$, where $\kappa = O(\sqrt{n})$ holds with high probability.*

In the next lemma we add the second level to the data structure of Lemma 17. The proof is in the full paper.

► **Lemma 18.** *We can compute in $O(n^2 \log \log n / \log^2 n)$ expected time a data structure with $O(n^2 / \log^2 n)$ sorted canonical subsets of P whose total size is $O(n^2 \log \log n / \log^2 n)$, so that for any disk D whose center is in C , we can find in $O(\kappa)$ time $O(\kappa)$ pairwise-disjoint canonical sets whose union is $P \setminus D$, where $\kappa = O(\log n)$ holds with high probability.*

We finally add the top-level data structure in Lemma 19, whose proof is in the full paper.

► **Lemma 19.** *For any $r < n / \log^{\omega(1)} n$, we can compute in $O(n \log n + nr \log \log r / \log^2 r)$ expected time a data structure with $O(nr / \log^2 r)$ sorted canonical subsets of P whose total size is $O(n \log(n/r) + nr \log \log r / \log^2 r)$, so that for any disk D whose center is in C , we can find in $O(\kappa)$ time $O(\kappa)$ pairwise-disjoint canonical sets whose union is $P \setminus D$, where $\kappa = O(\sqrt{n/r} \log r)$ holds with high probability.*

By setting $r = n^{1/3} \log^4 n / (\log \log n)^{2/3}$ in Lemma 19, we can obtain the following result.

► **Corollary 20.** *We can compute in $O(n^{4/3} \log^2 n (\log \log n)^{1/3})$ expected time by a randomized algorithm a data structure with $O(n^{4/3} \log^2 n / (\log \log n)^{2/3})$ sorted canonical subsets of P whose total size is $O(n^{4/3} \log^2 n (\log \log n)^{1/3})$, so that for any disk D whose center is in C , we can find in $O(\kappa)$ time $O(\kappa)$ pairwise-disjoint canonical sets whose union is $P \setminus D$, where $\kappa = O(n^{1/3} (\log \log n)^{1/3} / \log n)$ holds with high probability.*

As discussed before, plugging our above result in the algorithm of [8], we can solve the decision version of the discrete 2-center problem in $O(n^{4/3} \log^2 n (\log \log n)^{1/3})$ expected time. Using the decision algorithm and the $O(n^{4/3} \log^2 n)$ -time distance selection algorithm in [24], the discrete 2-center problem can be solved in $O(n^{4/3} \log^3 n (\log \log n)^{1/3})$ expected time.

► **Theorem 21.** *Given a set P of n points in the plane, the discrete 2-center problem can be solved in $O(n^{4/3} \log^3 n (\log \log n)^{1/3})$ expected time by a randomized algorithm.*

3.3.2 The deterministic result

The deterministic result also has three levels, which correspond to Lemmas 17, 18, and 19, respectively, but instead uses deterministic techniques. More specifically, we use the partition tree of Theorem 7 to obtain the lowest level data structure; the second level follows the same algorithm as Lemma 18 with the deterministic lowest level structure; the top level structure makes use of a partial half-space decomposition scheme of [27]. The next three lemmas present the three data structures, respectively, with their proofs in the full paper.

► **Lemma 22.** *We can compute in $O(n \log n)$ time a data structure with $O(n)$ sorted canonical subsets of P whose total size is $O(n \log \log n)$, so that for any disk D whose center is in C , we can find in $O(\sqrt{n} (\log n)^{O(1)})$ time $O(\sqrt{n} (\log n)^{O(1)})$ pairwise-disjoint canonical sets whose union is $P \setminus D$.*

► **Lemma 23.** *We can compute in $O(n^2 \cdot \log \log n / \log^2 n)$ time a data structure with $O(n^2 / \log^2 n)$ sorted canonical subsets of P whose total size is $O(n^2 \log \log \log n / \log^2 n)$, so that for any disk D whose center is in C , we can find in $O(\log n (\log \log n)^{O(1)})$ time $O(\log n (\log \log n)^{O(1)})$ pairwise-disjoint canonical sets whose union is $P \setminus D$.*

► **Lemma 24.** *For any $r \leq n$, we can compute in $O(n\sqrt{r} + n \log n + r^2 + (n^2/r) \cdot \log r \cdot \log \log(n/r) / \log^2(n/r))$ time a data structure with $O(r \log r + (n^2/r) \log r / \log^2(n/r))$ sorted canonical subsets of P whose total size is $O(n \log^2 r + (n^2/r) \log r \log \log \log(n/r) / \log^2(n/r))$, so that for any disk D whose center is in C , we can find in $O(\sqrt{r} \log(n/r) (\log \log(n/r))^{O(1)})$ time $O(\sqrt{r} \log(n/r) (\log \log(n/r))^{O(1)})$ pairwise-disjoint canonical sets whose union is $P \setminus D$.*

By setting $r = n^{2/3} (\log \log n)^{2/3} / \log^{10/3} n$ in the preceding lemma, we obtain the following result.

► **Corollary 25.** *We can compute in $O(n^{4/3} \log^{7/3} n (\log \log n)^{1/3})$ time a data structure with a total of $O(n^{4/3} \log^{7/3} n / (\log \log n)^{2/3})$ sorted canonical subsets of P whose total size is upper-bounded by $O(n^{4/3} \log^{7/3} n \log \log \log n / (\log \log n)^{2/3})$, so that for any disk D whose center is in C , we can find in $O(n^{1/3} (\log \log n)^{O(1)} / \log^{2/3} n)$ time $O(n^{1/3} (\log \log n)^{O(1)} / \log^{2/3} n)$ pairwise-disjoint canonical sets whose union is $P \setminus D$.*

According to our discussion before, plugging our above result in the algorithm of [8], we can solve the decision version of the discrete 2-center problem in $O(n^{4/3} \log^{7/3} n (\log \log n)^{O(1)})$ time. Using the decision algorithm and the $O(n^{4/3} \log^2 n)$ -time distance selection algorithm in [24], the discrete 2-center problem can be solved in $O(n^{4/3} \log^{10/3} n (\log \log n)^{O(1)})$ time.

► **Theorem 26.** *Given a set P of n points in the plane, the discrete 2-center problem can be solved in $O(n^{4/3} \log^{10/3} n (\log \log n)^{O(1)})$ time.*

4 Concluding remarks

Our techniques are likely to find other applications. Generally speaking, our techniques may be useful for solving problems involving a set of congruent disks in the plane. Our paper demonstrates that well-studied techniques for arrangements of lines may be adapted to solving problems involving arrangements of congruent disks. The general idea is to first reduce the problem to the same problem with respect to a pair of square cells using an algorithm like Lemma 1. Then, to tackle the problem on a pair of square cells (C, C') , we need to deal with an arrangement of spanning upper arcs in $\overline{C'}$ such that the centers of the underlying disks of these arcs are all in C . The properties of spanning upper arcs (e.g., Observation 2), along with the duality between the upper arcs in $\overline{C'}$ and the points in C , make an upper-arc arrangement “resemble” a line arrangement so that many algorithms and techniques on line arrangements may be easily adapted to the upper-arc arrangements.

References

- 1 Pankaj K. Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry*, Csaba D. Tóth, Joseph O’Rourke, and Jacob E. Goodman (eds.), pages 1057–1092. CRC Press, 3rd edition, 2017.
- 2 Pankaj K. Agarwal. Simplex range searching and its variants: a review. In *A Journey Through Discrete Mathematics*, pages 1–30. Springer, 2017.
- 3 Pankaj K. Agarwal, Boris Aronov, Micha Sharir, and Subhash Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.
- 4 Pankaj K. Agarwal and Jiří Matoušek. On range searching with semialgebraic sets. *Discrete and Computational Geometry*, 11:393–418, 1994.
- 5 Pankaj K. Agarwal, Jiří Matoušek, and Micha Sharir. On range searching with semialgebraic sets. II. *SIAM Journal on Computing*, 42:2039–2062, 2013.
- 6 Pankaj K. Agarwal, Marco Pellegrini, and Micha Sharir. Counting circular arc intersections. *SIAM Journal on Computing*, 22:778–793, 1993.
- 7 Pankaj K. Agarwal and Micha Sharir. Pseudoline arrangements: Duality, algorithms, and applications. *SIAM Journal on Computing*, 34:526–552, 2005.
- 8 Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. The discrete 2-center problem. *Discrete and Computational Geometry*, 20:287–305, 1998.
- 9 Jon L. Bentley and Hermann A. Maurer. A note on Euclidean near neighbor searching in the plane. *Information Processing Letters*, 8:133–136, 1979.
- 10 Timothy M. Chan. On enumerating and selecting distances. *International Journal of Computational Geometry and Application*, 11:291–304, 2001.
- 11 Timothy M. Chan. Optimal partition trees. *Discrete and Computational Geometry*, 47:661–690, 2012.
- 12 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete and Computational Geometry*, 56:866–881, 2016.
- 13 Timothy M. Chan and Da Wei Zheng. Hopcroft’s problem, log-star shaving, 2D fractional cascading, and decision trees. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 190–210, 2022.
- 14 Bernard Chazelle. An improved algorithm for the fixed-radius neighbor problem. *Information Processing Letters*, 16:193–198, 1983.
- 15 Bernard Chazelle. New techniques for computing order statistics in Euclidean space. In *Proceedings of the 1st Annual Symposium on Computational Geometry (SoCG)*, pages 125–134, 1985.
- 16 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993.

- 17 Bernard Chazelle, Richard Cole, Franco P. Preparata, and Chee-Keng Yap. New upper bounds for neighbor searching. *Information and Control*, 68:105–124, 1986.
- 18 Bernard Chazelle and Herbert Edelsbrunner. Optimal solutions for a class of point retrieval problems. *Journal of Symbolic Computation*, 1:47–56, 1985.
- 19 Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete and Computational Geometry*, 4(5):467–489, 1989.
- 20 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Heidelberg, 1987.
- 21 Jeff Erickson. New lower bounds for Hopcroft’s problem. *Discrete and Computational Geometry*, 16:389–418, 1996.
- 22 Michael T. Goodrich. Geometric partitioning made easier, even in parallel. In *Proceedings of the 9th Annual Symposium on Computational Geometry (SoCG)*, pages 73–82, 1993.
- 23 John Hershberger and Subhash Suri. Finding tailored partitions. *Journal of Algorithms*, 3:431–463, 1991.
- 24 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.
- 25 Jiří Matoušek. Cutting hyperplane arrangement. *Discrete and Computational Geometry*, 6:385–406, 1991.
- 26 Jiří Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8(3):315–334, 1992.
- 27 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10(1):157–182, 1993.
- 28 Jiří Matoušek. Geometric range searching. *ACM Computing Survey*, 26:421–461, 1994.
- 29 Jiří Matoušek and Zuzana Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete and Computational Geometry*, 54:22–41, 2015.
- 30 Micha Sharir. Computational geometry column 65. *SIGACT News*, 48:68–85, 2017.
- 31 Haitao Wang. On the planar two-center problem and circular hulls. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG)*, pages 68:1–68:14, 2020.
- 32 Frances F. Yao. A 3-space partition and its applications. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 258–263, 1983.

Space-Efficient Data Structure for Posets with Applications

Tatsuya Yanagita ✉

The University of Tokyo, Japan

Sankardeep Chakraborty ✉

The University of Tokyo, Japan

Kunihiko Sadakane ✉ 

The University of Tokyo, Japan

Srinivasa Rao Satti ✉ 

Norwegian University of Science and Technology, Trondheim, Norway

Abstract

Space efficient data structures for partial ordered sets or posets are well-researched field. It is known that a poset with n elements can be represented in $n^2/4 + o(n^2)$ bits [30] and can also be represented in $(1 + \epsilon)n \log n + 2nk + o(nk)$ bits [19] where k is width of the poset. In this paper, we make the latter data structure occupy $2n(k - 1) + o(nk)$ bits by considering topological labeling on the elements of posets. Also considering the topological labeling, we propose a new data structure that calculates queries on transitive reduction graphs of posets faster though queries on transitive closure graphs are computed slower. Moreover, we propose an alternative data structure for topological labeled posets that calculates both of the queries faster though it uses $3nk - 2n + o(nk)$ bits of space. Additionally, we discuss the advantage of these data structures from the perspective of an application for BlockDAG, which is a more scalable version of Blockchain.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Succinct Data Structures, Posets, Blockchain

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.33

1 Introduction

We consider space-efficient data structures for partially ordered sets (posets). Such data structures have been studied earlier. For arbitrary posets, Munro and Nicholson [30] proposed a data structure whose space complexity is $n^2/4 + o(n^2)$ bits, and this matches the lower bound $n^2/4 + 3n/2 + O(\log n)$ which is found by Kleitman and Rothschild [29]. When the poset has Dilworth's width [16] k^1 , Daskalakis et al. [15] showed that posets can be represented in $O(nk)$ words where n is the number of elements of the poset. Using their idea, Farzan and Fischer [19] developed a data structure for posets with $(1 + \epsilon)n \log n + 2nk + o(nk)$ bits, for any positive constant $\epsilon \leq 1$. It is known that

$$\frac{n!}{k!} 4^{n(k-1)} n^{-24k(k-1)} \leq N_k(n) \leq n! 4^{n(k-1)} n^{-\frac{(k-1)(k-2)}{2}} k^{\frac{k(k-1)}{2}}$$

when $N_k(n)$ is the number of posets on n elements with width k [10]. Hence, the information theoretic lower bound is $n \log n + 2n(k - 1) - \Theta(k^2 \log n)$ bits. Thus the data structure of Farzan and Fischer is succinct if $k = o(\sqrt{n})$ and $\epsilon = o(1)$.

There are a lot of other related works. It is well-known as Birkhoff's representation theorem that posets have one-to-one correspondence with distributive lattices [8]. There is a data structure for distributive lattices [32] whose size is close to the lower bound given by

¹ Hereafter we denote Dilworth's width by just *width*.



■ **Table 1** Comparison of data structures for posets. The data structure in Section 2.5 is the original version of Farzan and Fischer’s data structure [19] and the one in Section 3.2 is a modified version of that. G_C is a transitive closure graph of a poset and G_R is a transitive reduction graph of it. Here $o(nk)$ denotes $n \cdot o(k) + o(n) \cdot k$, and t denotes the size of the output of a query.

| | Section 2.5 | Section 3.2 | Section 3.3 | Section 3.4 |
|--------------------------|---|---------------------|------------------|----------------------|
| Labeling | General | Topological | Topological | Topological |
| Space [bits] | $(1 + \epsilon)n \log n$ $+ 2n(k - 1) + o(nk)$ | $2n(k - 1) + o(nk)$ | $2nk + o(nk)$ | $3nk - 2n + o(nk)$ |
| $\text{adj}_{G_C}(u, v)$ | $O(1/\epsilon)$ | $O(\log \log k)$ | $O(k^2)$ | $O(\log \log k)$ |
| $\text{succ}_{G_C}(v)$ | $O(1/\epsilon + k + t)$ | $O(k + t)$ | $O(k^2 + t)$ | $O(k + t)$ |
| $\text{pred}_{G_C}(v)$ | $O(1/\epsilon + k + t)$ | $O(k + t)$ | $O(k^2 + t)$ | $O(k + t)$ |
| $\text{adj}_{G_R}(u, v)$ | $O(1/\epsilon + k)$ | $O(k)$ | $O(\log \log k)$ | $O(\log \log k)$ |
| $\text{succ}_{G_R}(v)$ | $O(1/\epsilon + k^2)$ | $O(k^2)$ | $O(k)$ | $O(\log \log k + t)$ |
| $\text{pred}_{G_R}(v)$ | $O(1/\epsilon + k^2)$ | $O(k^2)$ | $O(k)$ | $O(k)$ |

Erne, Heitzig and Reinhold [18]. There are also succinct/compact data structures which can represent arbitrary binary relations [6, 7], interval graphs, chordal graphs and finally arbitrary graphs among many others [1, 2, 12, 13, 20, 33]. Recently, a new parameter called twin-width, which can be defined on posets, graphs and more generally matrices, was introduced by Bonnet et al. [9] and it is shown that the twin-width of posets is linear in their Dilworth’s width [5]. A compact data structure for matrices with fixed twin-width was proposed [36].

1.1 Main Results

We denote by n the number of elements in a poset and its width by k . We first show that by considering not generally labeled posets but topologically labeled posets, the space complexity of Farzan and Fischer’s data structure can be reduced to $2n(k - 1) + o(nk)$ bits (Section 3.2). This is succinct for non-constant $k = o(\frac{n}{\log n})$ (see Section 3.1). Also, we propose two alternative data structures to store topologically labeled posets. Our first data structure can support queries on transitive reduction graphs (defined in Section 2.2) faster than their data structure though queries on transitive closure graphs are slower (Section 3.3). The other data structure can support the queries on both the transitive reduction and transitive closure graphs faster though the space increases to $3nk - 2n + o(nk)$ bits (Section 3.4).

Table 1 shows the comparison of time complexity of queries among the modified version of Farzan and Fischer’s data structure in Section 3.2, the proposed data structure in Section 3.3 and another one in Section 3.4.

Our other contributions are to compress some of these data structures into less space (Section 3.5) and to make these data structures dynamic (Appendix B). The dynamic versions support the operation of adding an element to the poset.

1.2 Application

Blockchain is a technology for a public ledger of cryptocurrencies like Bitcoin, and was proposed by Satoshi Nakamoto in 2009 [34]. In Blockchain, blocks storing the data of transactions are linked in a chain, which has a scalability problem. In order to solve this problem, some protocols called *BlockDAG* are proposed in which the blocks are connected in a directed acyclic graph (DAG) instead of a chain. Typical BlockDAG protocols are PHANTOM and GHOSTDAG [37].

Let $G = (V, E)$ be a DAG where $V = \{1, 2, \dots, n\}$, $E \subseteq V \times V$. The DAG class of BlockDAG satisfies the conditions below:

1. $\forall (u, v) \in E, v < u$;
2. Only one sink (i.e., a vertex with no outgoing edges); and
3. If $(u, v_1), (u, v_2) \in E$, then there is no path from v_1 to v_2 .

In Condition 1, $<$ is the standard total order on V , i.e. $1 < 2 < \dots < n$. We call DAGs that satisfy Condition 1 as *topologically labeled DAGs*. Conditions 1 and 2 imply that the node labeled 1 is the sink and the DAG is connected. DAGs which satisfy Condition 3 are called transitive reduction graphs [3]. If we remove the sink from G , then the DAG class has one-to-one correspondence to posets labeled by a topological order.

For a Blockchain, its corresponding graph is a chain (path), which has width 1. For a BlockDAG, it is expected that the graph has small width k . It is therefore worth giving a space-efficient representation for graphs with small width.

2 Preliminaries

2.1 Chain Decomposition

A *partially ordered set* or a *poset* is a set with a binary relation which is reflexive, antisymmetric and transitive. We denote a poset by $\mathcal{P} = (V, \preceq)$ where $V = \{1, 2, \dots, n\}$ and \preceq is the relation. For any $a, b, c \in V$, the following holds:

- reflexivity: $a \preceq a$,
- antisymmetry: if $a \preceq b$ and $b \preceq a$, then $a = b$,
- transitivity: if $a \preceq b$ and $b \preceq c$, then $a \preceq c$.

When $u \preceq v$ and $u \neq v$, we denote it by $u < v$. We say that a poset is *topologically labeled* if $u \preceq v \Rightarrow u \leq v$ for any $u, v \in V$ where \leq is the standard total order on V . In 1950, Dilworth showed the duality between chains and antichains of posets [16].

► **Definition 1** (Antichain). $A \subseteq V$ is an antichain of \mathcal{P} if any two elements in A are unordered on \mathcal{P} .

► **Definition 2** (Chain). $C \subseteq V$ is a chain of \mathcal{P} if C is totally ordered on \mathcal{P} .

► **Definition 3** (Chain Decomposition). A set of disjoint sets $\{C_p\}_{p=0}^{k'-1}$ is a chain decomposition of \mathcal{P} if C_p is a chain of \mathcal{P} for all $p \in \{0, 1, \dots, k' - 1\}$ and $\bigcup_{p=0}^{k'-1} C_p = V$.

► **Theorem 4** (Dilworth's Theorem [16]). The maximum size of an antichain is equal to the minimum number of chains in any chain decomposition.

The maximum size of antichain is called *Dilworth's width* or simply *width* and we denote it by k . Fulkerson found that Dilworth's theorem is equivalent to König's theorem and the minimal chain decomposition can be obtained by solving a maximum matching on a bipartite graph [23]. It can be solved in $O(n^{2.5})$ time by using Hopcroft and Karp's algorithm [27]. The time complexity to calculate the minimal chain decomposition can be reduced to $O(kn^2)$ [14, 15].

2.2 Transitive Closure Graphs vs. Transitive Reduction Graphs

Transitive closure and transitive reduction are well-researched topics since 1970s. Given a poset \mathcal{P} , we define its transitive closure graph and transitive reduction graph as follows.

► **Definition 5** (Transitive Closure Graph). *A transitive closure graph of \mathcal{P} is a DAG $G_C = (V, E_C)$ where $E_C = \{(u, v) \in V \times V; v \prec u\}$.*

► **Definition 6** (Transitive Edge). *A transitive edge of a DAG $G = (V, E)$ is an edge $(u, v) \in E$ such that there exists a path from u to v other than the path going through the edge (u, v) .*

► **Definition 7** (Transitive Reduction Graph). *A transitive reduction graph of \mathcal{P} is a DAG $G_R = (V, E_R)$ where E_R is a set such that all the transitive edges are removed from E_C .*

Aho, Garey and Ullman generalized the transitive reduction to binary relations [3]. If the binary relation is antisymmetric, the transitive reduction graph is unique. Thus, the transitive reduction graph of a poset is unique. They also proved that both directions of the conversions between a transitive closure graph and a transitive reduction graph have algorithms of same time complexity.

The conversions between a transitive reduction graph and a transitive closure graph can be obtained in $O(n^{2.37286})$ time by using boolean matrix multiplication algorithm [4, 21]. They can also be calculated in $O(nm_R + m_C)$ time [25] where $m_R = |E_R|$ and $m_C = |E_C|$, and this time complexity is $O(kn^2)$ since $m_R \leq kn$ and $m_C \leq n(n-1)/2$.

Definition 7 is essentially equivalent to Condition 3 of the definition of BlockDAG in Section 1.2. The transitive reduction graphs are essentially same as Hasse diagrams of posets. A poset can be represented either by its transitive closure graph or its transitive reduction graphs.

2.3 Queries on Posets

In this section, we introduce the queries that are supported by our data structures for posets, or transitive closure/reduction graphs. We consider the following six queries:

- $\text{adj}_{G_C}(u, v) \cdots$ return 1 if $(u, v) \in E_C$ and otherwise 0;
- $\text{succ}_{G_C}(v) \cdots$ return the set $\{u \in V; (v, u) \in E_C\}$;
- $\text{pred}_{G_C}(v) \cdots$ return the set $\{u \in V; (u, v) \in E_C\}$;
- $\text{adj}_{G_R}(u, v) \cdots$ return 1 if $(u, v) \in E_R$ and otherwise 0;
- $\text{succ}_{G_R}(v) \cdots$ return the set $\{u \in V; (v, u) \in E_R\}$;
- $\text{pred}_{G_R}(v) \cdots$ return the set $\{u \in V; (u, v) \in E_R\}$.

In particular, $\text{adj}_{G_R}(u, v) = 1 \Rightarrow \text{adj}_{G_C}(u, v) = 1$ holds for all $u, v \in V$ and $\text{succ}_{G_R}(v) \subseteq \text{succ}_{G_C}(v)$ and $\text{pred}_{G_R}(v) \subseteq \text{pred}_{G_C}(v)$ also hold for any $v \in V$.

We also define some terminologies and utility functions. Given a chain decomposition of a poset, *node index* is the label of a node and *chain index* of a node is the pair (p, i) such that the node is the i -th node of chain p . We often specify a node not only by its node index but also by its chain index. Function $\text{node_index}(p, i)$ converts the chain index (p, i) into its node index. On the other hand, function $\text{chain_index}(v)$ converts the node index v into its chain index. The *lower bound* in chain q of a node (p, i) is the node (q, j) such that j is the maximum value which satisfies $(q, j) \prec (p, i)$ and we denote it by $lb_q(p, i)$. To the contrary, the *upper bound* in chain q of a node (p, i) is the node (q, j) such that j is the minimum value which satisfies $(p, i) \prec (q, j)$ and we denote it by $ub_q(p, i)$. When v is the node index of (p, i) , we define $lb_q(v) = lb_q(p, i)$ and $ub_q(v) = ub_q(p, i)$.

2.4 Succinct Data Structures

In this paper, we discuss data structures in the word-RAM model [22], which supports reading, writing, arithmetic operations and bitwise operations on a word of $w = \Omega(\log N)$ bits in constant time, where N is the size of data. We make use of some of the basic succinct data structures such as bitvector, string and permutation.

A bitvector data structure stores a sequence N bits, $B[1 \dots N]$ and supports random access and rank/select queries described below:

- $B[i] \dots$ return i -th element;
- $\text{rank}_c(B, i) \dots$ return the number of $j \in \{1, 2, \dots, \min\{i, n\}\}$ such that $B[j] = c$;
- $\text{select}_c(B, i) \dots$ return $\inf\{j \in \mathbb{N}; \text{rank}_c(B, j) \geq i\}$.

Note that $\text{select}_c(B, i)$ returns ∞ when i is greater than the number of c 's in B . All of these queries can be supported in constant time using a data structure whose space complexity is $N + o(N)$ bits [28]. It can also be compressed into $\log \binom{N}{M} + o(N)$ bits [11, 35] when M is the number of 1s in the bitvector.

String $S[1 \dots N]$ is a generalisation of the bitvector into a sequence of characters from the alphabet $\{0, 1, \dots, L-1\}$ instead of bits in $\{0, 1\}$. A succinct string data structure supports random access and rank/select queries for each $c \in \{0, 1, \dots, L-1\}$. Random access and rank queries can be calculated in $O(\log \log L)$ time and select query can be computed in $O(1)$ time, and the data structure can be stored in $N \log L + o(N \log L)$ bits of space [24]. Moreover, the wavelet tree [26] data structure can be used to support all three operations in $O(\log L)$ time.

A permutation data structure represents a bijection $\pi : \{1, 2, \dots, N\} \rightarrow \{1, 2, \dots, N\}$ and supports the queries $\pi(i)$ and $\pi^{-1}(i)$ for all $i = 1, 2, \dots, N$. The permutation data structure of Munro et al. [31] supports $\pi(i)$ in constant time and $\pi^{-1}(i)$ in $O(1/\epsilon)$ time using $(1 + \epsilon)N \log N + N + o(N)$ bits, for any parameter $0 < \epsilon \leq 1$.

2.5 Farzan and Fischer's Data Structure [19]

Let \mathcal{P} be a poset on n elements with Dilworth's width k , and let $\{C_p\}_{p \in \Sigma}$ be one of the minimal chain decompositions of \mathcal{P} where $\Sigma = \{0, 1, \dots, k-1\}$. Farzan and Fischer's data structure consists of a permutation π , a bitvector B and bitvectors D_{pq} for each $p, q \in \Sigma$ such that $p \neq q$. The permutation π is the bijection of $V \rightarrow V$ and it uses $(1 + \epsilon)n \log n + n + o(n)$ bits of space for any $\epsilon \in (0, 1]$. The length of the bitvector B is n and that of D_{pq} is $|C_p| + |C_q|$ for each $p, q \in \Sigma, p \neq q$. We store auxiliary structures to support rank and select queries on each of the bit vectors (B and D_{pq} , for each $p, q \in \Sigma, p \neq q$). Thus the space complexity of the whole data structure is

$$(1 + \epsilon)n \log n + n + o(n) + \left(n + \sum_{p, q \in \Sigma, p \neq q} (|C_p| + |C_q|) \right) (1 + o(1)) = (1 + \epsilon)n \log n + 2nk + o(nk)$$

bits.

Permutation π and bitvector B represent the correspondence between node indices and chain indices. We construct π so that $\pi^{-1}(v) = i + \sum_{q < p} |C_q|$ for each $v \in V$ where (p, i) is the chain index of v . In other words, π reorders the nodes from the lexicographical order of chain indices to the order of node indices. Also, we construct B in such a way that $B[j] = 1$ if and only if there exists $p \in \Sigma$ which satisfies $j = \sum_{q \leq p} |C_q|$; B essentially represents the length of the individual chains in the decomposition (note that one can store B in a compressed form, but this would not change the overall space complexity). Then, `node_index(v)` and `chain_index(p, i)` are implemented as Algorithms 1 and 2 respectively. The time complexity of `node_index(v)` is $O(1)$ and that of `chain_index(v)` is $O(1/\epsilon)$.

For each $p, q \in \Sigma, p \neq q$, D_{pq} is constructed as $D_{pq} = 0^{\delta_{pq}^0} 1 0^{\delta_{pq}^1} 1 \dots 0^{\delta_{pq}^{|C_p|-1}} 1 0^{\delta_{pq}^{|C_p|}}$ where δ_{pq}^i is an increase of the number of edges on G_C from node $(p, i+1)$ to nodes in chain q as compared with node (p, i) . We consider that there is no outgoing edge from node $(p, 0)$ and any node can be reached from node $(p, |C_p|+1)$. In particular, $\sum_{i < i'} \delta_{pq}^i = |\{j \in$

■ **Algorithm 1** convert chain index to node index.

```

1: procedure node_index( $p, i$ )
2:    $a := i + \text{select}_1(B, p)$ 
3:   return  $\pi(a)$ 

```

■ **Algorithm 2** convert node index to chain index.

```

1: procedure chain_index( $v$ )
2:    $a := \pi^{-1}(v)$ 
3:    $p := \text{rank}_1(B, a - 1)$ 
4:   return  $(p, a - \text{select}_1(B, p))$ 

```

$J; (q, j) \prec (p, i')\}$ for all $i' \in \{0, 1, \dots, |C_p|+1\}$ where $J = \{1, 2, \dots, |C_q|\}$. By using D_{pq} , we can obtain the lower bound and the upper bound in chain q of node (p, i) in constant time with Algorithm 3 and 4. An edge from node (p, i) to node (q, j) is a transitive edge when $j < \text{lower_bound}(p, i, q)$ because there exists a path which goes over $lb_q(p, i)$. Node (p, i) does not have a path to node (q, j) when $j > \text{lower_bound}(p, i, q)$ since if there is such a path, $lb_q(p, i) \prec (q, j) \prec (p, i)$ and it contradicts the definition of lower bound.

■ **Algorithm 3** return j if (q, j) is the largest node reachable from (p, i) .

```

1: procedure lower_bound( $p, i, q$ )
2:   if  $p = q$  then return  $i - 1$ 
3:   else return  $\text{select}_1(D_{pq}, i) - i$ 

```

Then, we can implement the algorithms for the six queries mentioned in Section 2.3 as follows. First, $\text{adj}_{G_C}(u, v) = 1$ if and only if $lb_q(u) \preceq v$ for such q that $v \in C_q$. Hence, all we have to do in order to compute $\text{adj}_{G_C}(u, v)$ is to check $j \leq \text{lower_bound}(p, i, q)$ when (p, i) is the chain index of u and (q, j) is that of v . The bottleneck of the computation is the conversions to chain indices and the time complexity of $\text{adj}_{G_C}(u, v)$ is $O(1/\epsilon)$. Also, $\text{succ}_{G_C}(v)$ (respectively, $\text{pred}_{G_C}(v)$) can be calculated by collecting all the nodes less (respectively, greater) than or equal to $lb_p(v)$ (respectively, $ub_p(v)$) for all $p \in \Sigma$. The time complexities of $\text{succ}_{G_C}(v)$ and $\text{pred}_{G_C}(v)$ are both $O(1/\epsilon + k + t)$ where t is the size of the output.

To compute $\text{adj}_{G_R}(u, v)$, we need to check that there is no path from u to v other than the direct edge (u, v) . If there exists $p \in \Sigma$ such that $ub_p(v) \preceq lb_p(u)$, there exists a non-direct path which goes through $ub_p(v)$ since $v \prec ub_p(v) \preceq lb_p(u) \prec u$. Thus, $\text{adj}_{G_R}(u, v) = 1$ if and only if $\text{adj}_{G_C}(u, v) = 1$ and $lb_p(u) \prec ub_p(v)$ for all $p \in \Sigma$. Checking $lb_p(u) \prec ub_p(v)$ for all $p \in \Sigma$ costs $O(1/\epsilon + k)$ time and therefore the time complexity of $\text{adj}_{G_R}(u, v)$ is also $O(1/\epsilon + k)$. It can be easily observed that $\text{succ}_{G_R}(v) = \{u \in L(v); \text{adj}_{G_R}(v, u) = 1\}$ and $\text{pred}_{G_R}(v) = \{u \in U(v); \text{adj}_{G_R}(u, v) = 1\}$ where $L(v) = \{lb_p(v); p \in \Sigma\}$ and $U(v) = \{ub_p(v); p \in \Sigma\}$. Hence, the time complexities of $\text{succ}_{G_R}(v)$ and $\text{pred}_{G_R}(v)$ are both $O(1/\epsilon + k^2)$.

3 Improved Data Structures

3.1 Lower Bound on Space

Let $Z_k(n)$ be the number of topologically labeled posets and $X_k(n)$ be the number of unlabeled posets. As written in Section 1, the number of generally labeled posets $N_k(n)$ satisfies

$$N_k(n) \geq \frac{n!}{k!} 4^{n(k-1)} n^{-24k(k-1)}.$$

■ **Algorithm 4** return j if (q, j) is the smallest node reachable to (p, i) .

```

1: procedure upper_bound( $p, i, q$ )
2:   if  $p = q$  then return  $i + 1$ 
3:   else return  $\text{select}_0(D_{qp}, i) - i + 1$ 

```

Each unlabeled posets has at most $n!$ distinct labeling, therefore $n! \cdot X_k(n) \geq N_k(n)$ holds. It is clear that $Z_k(n) \geq X_k(n)$ and the information-theoretic lower bound on the number of topologically labeled posets is

$$\log Z_k(n) \geq 2n(k-1) - \Theta(k^2 \log n)$$

bits. Thus, it is possible to compress posets into space which is linear in n with fixed k when the posets are topologically labeled.

3.2 Modified Farzan and Fischer's Data Structure

In this section, we adapt Farzan and Fischer's data structure in Section 2.5 for topologically labeled posets and make its space close to the lower bound in Section 3.1. In order to do so, we replace the permutation π and the bitvector B with a string S . Eventually, this data structure consists of the string S and the bitvectors D_{pq} ($p, q \in \Sigma, p \neq q$).

The length of S is n , its alphabet is Σ and $S[v] = p$ if $v \in C_p$. String S plays the role of storing information of the correspondence between node indices and chain indices, as well as π and B in Section 2.5 do. By using S , $\text{node_index}(p, i)$ and $\text{chain_index}(v)$ can be calculated by Algorithms 5 and 6, respectively. Since select_p takes $O(1)$ time and rank_p takes $O(\log \log k)$ time, $\text{node_index}(p, i)$ can be supported in $O(1)$ time and $\text{chain_index}(v)$ in $O(\log \log k)$ time.

■ **Algorithm 5** convert chain index to node index.

```

1: procedure node_index( $p, i$ )
2:   return  $\text{select}_p(S, i)$ 

```

■ **Algorithm 6** convert node index to chain index.

```

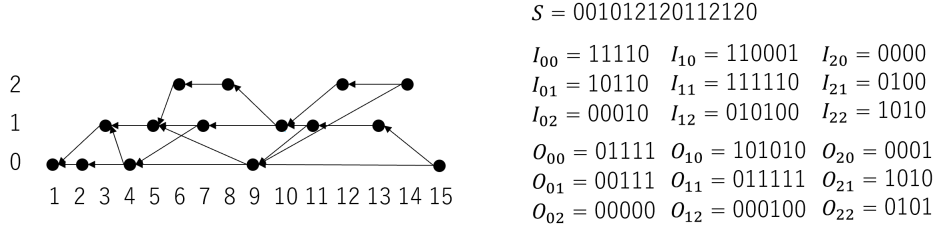
1: procedure chain_index( $v$ )
2:    $p := S[v]$ 
3:   return  $(p, \text{rank}_p(S, v))$ 

```

Then, $\text{adj}_{G_C}(u, v)$ can be computed in $O(\log \log k)$ time, $\text{succ}_{G_C}(v)$ and $\text{pred}_{G_C}(v)$ in $O(k+t)$ time where t is the size of output, $\text{adj}_{G_R}(u, v)$ in $O(k)$ time, and $\text{succ}_{G_R}(v)$ and $\text{pred}_{G_R}(v)$ in $O(k^2)$ time. Also, the total space of the data structure is

$$\left(n \log k + \sum_{p, q \in \Sigma, p \neq q} (|C_p| + |C_q|) \right) (1 + o(1)) = 2n(k-1) + o(nk)$$

bits. For non-constant $k = o(\frac{n}{\log n})$, this data structure is succinct since $\Theta(k^2 \log n) = o(nk)$.



■ **Figure 1** Example of a poset and its encoding.

3.3 Proposed Data Structure

The data structure in Section 3.2 can perform queries on G_C fast, whereas it is slow to do queries on G_R . However, when we have to run some BFS/DFS-based algorithm, such as algorithms to solve a shortest path problem on G_R or a longest path problem, it is required to do such queries on G_R faster. For this reason, we propose a new data structure which can support queries on G_R faster.

Our data structure consists of a string S and bitvectors I_{pq}, O_{pq} ($p, q \in \Sigma$). The string S is same as the one in Section 3.2. The lengths of I_{pq} and O_{pq} are both equal to $|C_p|$. Therefore, the total space of our data structure is

$$\left(n \log k + 2k \cdot \sum_{p=0}^{k-1} |C_p| \right) (1 + o(1)) = 2nk + o(nk)$$

bits. $I_{pq}[i] = 1$ when node (p, i) has an incoming edge from some node of chain q and otherwise $I_{pq}[i] = 0$. Similarly, $O_{pq}[i] = 1$ when node (p, i) has an outgoing edge to some node of chain q and otherwise $O_{pq}[i] = 0$. In Figure 1, the left graph is an example of a transitive reduction graph of a poset with $n = 15$ and $k = 3$ and its encoding is shown in the right side.

If there exists an edge $(u, v) \in E_R$ such that $u \in C_p$ and $v \in C_q$, there is no edge $(u', v') \in E_R \setminus \{(u, v)\}$ which satisfies $u' \in C_p, v' \in C_q$ and $u \preceq u', v \succeq v'$. This is because when there exists such an edge (u', v') , then $u \preceq u' \prec v' \preceq v$ and (u, v) becomes a transitive edge since $(u, v) \neq (u', v')$. Therefore, the incoming edge corresponding to the i -th 1 of I_{pq} is the same as the outgoing edge corresponding to the i -th 1 of O_{qp} .

Let $E_R(K) = \{(u, v) \in E_R; u \in C_i, v \in C_j, i, j \in K\}$ for each $K \subseteq \Sigma$. We define functions called `nearest_dst` (p, i, q) and `nearest_src` (p, i, q) which return the second element of chain index of the nearest node in chain q reachable from (respectively, reachable to) the node (p, i) through the edges in $E_R(\{p, q\})$. If there is no such node, then it returns 0 (respectively, ∞). These can be calculated by Algorithms 7 and 8 in constant time.

■ **Algorithm 7** return the nearest destination in chain q from the node (p, i) .

```

1: procedure nearest_dst( $p, i, q$ )
2:   return select1( $I_{qp}, \text{rank}_1(O_{pq}, i)$ )

```

■ **Algorithm 8** return the nearest source in chain q to the node (p, i) .

```

1: procedure nearest_src( $p, i, q$ )
2:   return select1( $O_{qp}, \text{rank}_1(I_{pq}, i - 1) + 1$ )

```

Other utility functions we defined are $\text{lower_bounds}(v)$ and $\text{upper_bounds}(v)$. These functions return an array of length k which stores the node indices of $lb_p(v)$ (respectively, $ub_p(v)$) for each $p \in \Sigma$. If there is no such node, then it stores 0 (respectively, ∞). The algorithms of $\text{lower_bounds}(v)$ and $\text{upper_bounds}(v)$ are shown in Algorithm 9 and 10. These algorithms use the data structures known as priority queue. $\text{priority_queue}_{\leq}$ and $\text{priority_queue}_{\geq}$ support $\text{push}(p, i)$ to push a new node (p, i) , $\text{pop}()$ to pop out the node of the largest (respectively, smallest) node index, $\text{top}()$ to access the node of the largest (respectively, smallest) node index and $\text{update}(p, i)$ to update the node $(p, *)$ in the priority queue to (p, i) if the node index of (p, i) is larger (respectively, smaller) than that of $(p, *)$. Incidentally, we regard the node index of $(*, 0)$ as 0 and that of $(*, \infty)$ as ∞ . Remind that the conversion from chain index to node index can be done in constant time, therefore comparing nodes by node index does not affect the time complexities. In this paper, we use Relaxed Heap [17] which supports $\text{push}(p, i)$, $\text{top}(p, i)$, and $\text{update}(p, i)$ in $O(1)$ time and $\text{pop}(p, i)$ in $O(\log N)$ time where N is the number of elements in the priority queue.

■ **Algorithm 9** return the largest nodes reachable from v in each chain.

```

1: procedure lower_bounds( $v$ )
2:   array  $R[0 \dots k - 1] := \{0, \dots, 0\}$ 
3:   priority_queue $_{\leq}$   $Q := \{(0, 0), (1, 0), \dots, (k - 1, 0)\}$ 
4:    $(p_0, i_0) := \text{chain\_index}(v)$ 
5:    $Q.\text{update}(p_0, i_0)$ 
6:   while  $Q \neq \emptyset$  do
7:      $(p, i) := Q.\text{top}()$ 
8:      $Q.\text{pop}()$ 
9:     if  $i = 0$  then break
10:     $R[p] \leftarrow \text{node\_index}(p, i)$ 
11:    for  $q = 0, 1, \dots, k - 1$  do
12:      if  $R[q] \neq 0$  then continue
13:       $j := \text{nearest\_dst}(p, i, q)$ 
14:       $Q.\text{update}(q, j)$ 
15:     $R[p_0] \leftarrow \text{node\_index}(p_0, i_0 - 1)$ 
16:  return  $R$ 

```

In Algorithms 9 and 10, the while loop iterates at most k times because priority queue Q has k elements before the while loop and the elements are popped one by one in each iteration. The bottleneck of $\text{lower_bounds}(v)$ and $\text{upper_bounds}(v)$ is the for loop. The statements in it run at most k^2 times and the time complexity of the whole algorithm is $O(k^2)$.

The correctness of Algorithm 9 can be shown as follows. Trivially, each element of R is updated in line 10 at most once before reaching line 15. Let $K \subseteq \Sigma$ be a set of the indices such that the elements of R corresponding to them have been already updated and $\bar{K} = \Sigma \setminus K$. In the while loop, assume that $R[p']$ correctly stores the largest node v' in chain p' such that $v' \preceq v$ when $p' \in K$. Let u be the node index of the node (p, i) declared in line 7. If $i = 0$, it means that the nodes in C_p cannot be reached from v and the nodes in $\bigcup_{p' \in \bar{K}} C_{p'}$ cannot be either because $Q = \{(p', 0); p' \in \bar{K}\}$ holds since the node indices of all the nodes in Q are not larger than the node index of $(p, 0)$. Otherwise, it is easy to observe that node u is the largest destination reachable from v through the edges in $E_R(K \cup \{p\})$. If there exists a node $w \in \bigcup_{p' \in \bar{K} \setminus \{p\}} C_{p'}$ such that $u \prec w \prec v$, then for some $p' \in \bar{K} \setminus \{p\}$, there

33:10 Space-Efficient Data Structure for Posets

■ **Algorithm 10** return the smallest nodes reachable to v in each chain.

```

1: procedure upper_bounds( $v$ )
2:   array  $R[0 \dots k-1] := \{\infty, \dots, \infty\}$ 
3:   priority_queue $\geq$   $Q := \{(0, \infty), (1, \infty), \dots, (k-1, \infty)\}$ 
4:    $(p_0, i_0) := \text{chain\_index}(v)$ 
5:    $Q.\text{update}(p_0, i_0)$ 
6:   while  $Q \neq \emptyset$  do
7:      $(p, i) := Q.\text{top}()$ 
8:      $Q.\text{pop}()$ 
9:     if  $i = \infty$  then break
10:     $R[p] \leftarrow \text{node\_index}(p, i)$ 
11:    for  $q = 0, 1, \dots, k-1$  do
12:      if  $R[q] \neq \infty$  then continue
13:       $j := \text{nearest\_src}(p, i, q)$ 
14:       $Q.\text{update}(q, j)$ 
15:   $R[p_0] \leftarrow \text{node\_index}(p_0, i_0 + 1)$ 
16:  return  $R$ 

```

exists at least one node $w' \in C_{p'}$ such that $u \prec w'$ and w' can be reached from v through the edges in $E_R(K \cup \{p'\})$, and there also exists a node $w'' \in C_{p'}$ which satisfies $w'' \in Q$ and $w' \preceq w''$. However, $u < w''$ if $u \prec w''$ and it contradicts with the features of priority queue Q . Therefore, node u is also the largest destination reachable from v through any edges in E_R . Hence, even if we update the value of $R[p]$ into u and K into $K \cup \{u\}$, it does not go against the assumption, and recursively, it can be confirmed that $R[p] \preceq v$ holds for each $p \in \Sigma$ just before line 15. Finally, $R[p_0]$ is modified in line 15 not to be equal to v itself, and then we obtain the correct answer. The correctness of Algorithm 10 can be proved in the same way.

Using these utility functions, we can perform the six queries shown in Section 2.3. Each algorithm of the queries are given in Algorithms 11, 12, 13, 14, 15 and 16. $\text{adj}_{G_C}(u, v)$ can be calculated in $O(k^2)$, $\text{succ}_{G_C}(v)$ and $\text{pred}_{G_C}(v)$ in $O(k^2 + t)$ where t is the size of output, $\text{adj}_{G_R}(u, v)$ in $O(1)$ and $\text{succ}_{G_R}(v)$ and $\text{pred}_{G_R}(v)$ in $O(k)$.

Note that I_{pp} and O_{pp} ($p \in \Sigma$) is not necessary to reconstruct the poset because $I_{pp}[i] = 1$ if and only if $lb_q(p, i+1) \prec ub_q(p, i)$ for all $q \in \Sigma$ and $O_{pp}[i] = 1$ if and only if $lb_q(p, i) \prec ub_q(p, i-1)$ for all $q \in \Sigma$. Remind that $\text{lower_bounds}(v)$ and $\text{upper_bounds}(v)$ can be obtained without using I_{pp} and O_{pp} ($p \in \Sigma$) since the arguments never be $p = q$ in line 13 of Algorithm 9 and 10. The total space of I_{pp} and O_{pp} ($p \in \Sigma$) is

$$\left(2 \cdot \sum_{p=0}^{k-1} |C_p| \right) (1 + o(1)) = 2n + o(n)$$

bits. Thus, when we do not store them, the space of the data structure becomes $2n(k-1) + o(nk)$ bits and this is close to the information theoretical lower bound in Section 3.1. However, the space with I_{pp} and O_{pp} ($p \in \Sigma$) is also asymptotically the same because

$$2nk + o(nk) = 2n(k-1) + 2n + o(nk) = 2n(k-1) + o(nk).$$

Therefore, we store them for the sake of fast response to the queries.

3.4 Faster Index

We also propose another data structure which is efficient to compute all six queries at the extra expense of space. The idea of this data structure is based on the modified version of Farzan and Fischer's data structure in Section 3.2. This data structure consists of three components: string S , bitvectors D_{pq} ($p, q \in \Sigma, p \neq q$) and bitvectors T_v ($v \in V$).

The string S and the bitvectors D_{pq} is same as the one in Section 3.3. The bitvectors T_v has length k for each $v \in V$ and represents whether node v has a non-transitive edge to each chain or not. If one of the edges on G_C from node v to the node in chain p is not a transitive edge, then $T_v[p] = 1$.

The string S requires $n \log k + o(n \log k)$ bits, the bitvector D_{pq} does $|C_p| + |C_q| + o(|C_p| + |C_q|)$ bits for each $p, q \in \Sigma, p \neq q$ and the bitvector T_v does $k + o(k)$ bits for each $v \in V$. Thus, the total space of the data structure is

$$\left(n \log k + \sum_{p, q \in \Sigma, p \neq q} (|C_p| + |C_q|) + \sum_{v \in V} k \right) (1 + o(1)) = 3nk - 2n + o(nk)$$

bits.

On this data structure, $\text{node_index}(p, i)$ and $\text{chain_index}(v)$ can be done by Algorithm 5 and 6. Also, $\text{adj}_{G_C}(u, v)$, $\text{succ}_{G_C}(v)$ and $\text{pred}_{G_C}(v)$ can be computed in the same way as mentioned in Section 3.2.

$\text{adj}_{G_R}(u, v)$, $\text{succ}_{G_R}(v)$ and $\text{pred}_{G_R}(v)$ can be calculated by the following ways. The edge from (p, i) to $lb_q(p, i)$ is a non-transitive edge when $T_v[q] = 1$ where v is the node index of (p, i) . Thus, $\text{adj}_{G_R}(u, v)$ returns 1 if and only if $\text{lower_bound}(p, i, q) = j$ and $T_u[q] = 1$ where (p, i) is a chain index of u and (q, j) is that of v . The bottleneck of $\text{adj}_{G_R}(u, v)$ is the conversions from node indices to chain indices and $\text{adj}_{G_R}(u, v)$ can be obtained in $O(\log \log k)$ time. $\text{succ}_{G_R}(v)$ can be computed by the process that collects the node index of $lb_q(p, i)$ for all $q \in \Sigma$ such that $T_v[q] = 1$ where (p, i) is a chain index of v . When we only iterate q which satisfies the condition $T_v[q] = 1$ by using select query on T_v , the time complexity is $O(\log \log k + t)$ where t is the size of output. $\text{pred}_{G_R}(v)$ can be computed in $O(k)$ time by the process that checks whether node $ub_q(p, i)$ is adjacent to node v on G_R for each $q \in \Sigma$ where (p, i) is a chain index of v .

3.5 Higher Order Compression

First, we consider the data structure in Section 3.3. Let m_{pq} be the number of 1s in bitvector I_{pq} for each $p, q \in \Sigma$. It is obvious that m_{pq} is also the number of 1s in bitvector O_{qp} for each $p, q \in \Sigma$ and $\sum_{p, q \in \Sigma} m_{pq} = m_R$. When we use the compressed bitvectors for each I_{pq} and O_{pq} , the space complexity of the whole data structure becomes

$$n \log k + 2 \cdot \sum_{p, q \in \Sigma} \log \binom{|C_p|}{m_{pq}} + o(nk) \leq 2 \log \binom{nk}{m_R} + o(nk)$$

bits.

The data structure in Section 3.4 can also be compressed. Let m'_v be the number of 1s in bitvectors T_v for each $v \in V$. Then, $\sum_{v \in V} m'_v = m_R$. The space complexity of the data structure becomes

$$n \log k + \sum_{p, q \in \Sigma, p \neq q} (|C_p| + |C_q|) + \sum_{v \in V} \log \binom{k}{m'_v} + o(nk) \leq \log \binom{nk}{m_R} + 2n(k-1) + o(nk)$$

bits if we use compressed bitvectors for each T_v .

4 Conclusions

On the DAGs of BlockDAG, the growth of the width is much less likely than the depth, which is the longest path of the DAG. Thus, the space to store them can be almost linear in the number of elements with the data structures we consider in Section 3. This is one of the advantages that the other existing data structures do not have. Also, we provide the trade-off among time of the queries on G_C , time on G_R and space.

There remain some open problems such as:

- Can adding operation in the dynamic data structure be faster?
- Can the time of construction of these data structures be reduced by calculating the chain decomposition approximately?

It would be also interesting to consider other queries.

References

- 1 H. Acan, S. Chakraborty, S. Jo, K. Nakashima, K. Sadakane, and S. R. Satti. Succinct representations of intersection graphs on a circle. In *31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021*, pages 123–132. IEEE, 2021.
- 2 H. Acan, S. Chakraborty, S. Jo, and S. R. Satti. Succinct data structures for families of interval graphs. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2019.
- 3 A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.
- 4 J. Alman and V. V. Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. ACM-SIAM, 2021.
- 5 J. Balabán and P. Hliněný. Twin-width is linear in the poset width. In *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 6 J. Barbay, F. Claude, and G. Navarro. Compact binary relation representations with rich functionality. *Information and Computation*, 232:19–37, 2013.
- 7 J. Barbay, M. He, J. I. Munro, and S. R. Satti. Succinct indexes for strings, binary relations and multilabeled trees. *ACM Transactions on Algorithms*, 7(4), 2011.
- 8 Garrett Birkhoff. On the structure of abstract algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 31(4):433–454, 1935.
- 9 É. Bonnet, E. J. Kim, S. Thomassé, and R. Watrigant. Twin-width I: Tractable FO model checking. *Journal of the ACM*, 69(1), 2021.
- 10 G. Brightwell and S. Goodall. The number of partial orders of fixed width. *Order*, 13(4):315–337, 1996.
- 11 A. Brodnik and J. I. Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.
- 12 S. Chakraborty, S. Jo, K. Sadakane, and S. R. Satti. Succinct data structures for series-parallel, block-cactus and 3-leaf power graphs. In *Combinatorial Optimization and Applications - 15th International Conference, COCOA 2021, Tianjin, China, December 17-19, 2021, Proceedings*, volume 13135 of *Lecture Notes in Computer Science*, pages 416–430. Springer, 2021.
- 13 S. Chakraborty, S. Jo, K. Sadakane, and S. R. Satti. Succinct data structures for small clique-width graphs. In *31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021*, pages 133–142. IEEE, 2021.
- 14 Y. Chen and Y. Chen. On the DAG decomposition. *British Journal of Mathematics & Computer Science*, 10:1–27, 2015.

- 15 C. Daskalakis, R. M. Karp, E. Mossel, S. J. Riesenfeld, and E. Verbin. Sorting and selection in posets. *SIAM Journal on Computing*, 40(3):597–622, 2011.
- 16 R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.
- 17 J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11):1343–1354, 1988.
- 18 M. Ern e, J. Heitzig, and J. Reinhold. On the number of distributive lattices. *The Electronic Journal of Combinatorics*, 9, 2002.
- 19 A. Farzan and J. Fischer. Compact representation of posets. In *Proceedings of the 22nd International Conference on Algorithms and Computation (ISAAC)*, pages 302–311, Berlin, Heidelberg, 2011. Springer-Verlag.
- 20 A. Farzan and J. I. Munro. Succinct encoding of arbitrary graphs. *Theoretical Computer Science*, 513:38–52, 2013.
- 21 M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 129–131, 1971.
- 22 M. L. Fredman and D. E. Willard. Blasting through the information theoretic barrier with fusion trees. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–7, New York, NY, USA, 1990.
- 23 D. R. Fulkerson. Note on Dilworth’s decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7(4), 1956.
- 24 A. Golynski, J. I. Munro, and S. R. Satti. Rank/select operations on large alphabets: A tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 368–373, USA, 2006. Society for Industrial and Applied Mathematics.
- 25 A. Goral ıkova and V. Koubek. A reduct-and-closure algorithm for graphs. In *Mathematical Foundations of Computer Science 1979*, pages 301–307, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.
- 26 R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–850, USA, 2003. Society for Industrial and Applied Mathematics.
- 27 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
- 28 G. J. Jacobson. *Succinct Static Data Structures*. PhD thesis, Carnegie Mellon University, USA, 1988. AAI8918056.
- 29 D. J. Kleitman and B. L. Rothschild. Asymptotic enumeration of partial orders on a finite set. *Transactions of the American Mathematical Society*, 205:205–220, 1975.
- 30 J. I. Munro and P. K. Nicholson. Succinct posets. *Algorithmica*, 76(2):445–473, 2016.
- 31 J. I. Munro, R. Raman, V. Raman, and S. R. Satti. Succinct representations of permutations and functions. *Theoretical Computer Science*, 438:74–88, 2012.
- 32 J. I. Munro and C. Sinnamon. Time and space efficient representations of distributive lattices. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 550–567. ACM-SIAM, 2018.
- 33 J. I. Munro and K. Wu. Succinct data structures for chordal graphs. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPICs*, pages 67:1–67:12. Schloss Dagstuhl - Leibniz-Zentrum fur Informatik, 2018.
- 34 S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 2009. URL: <https://bitcoin.org/bitcoin.pdf>.
- 35 R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31(2):353–363, 2001.

- 36 M. Pilipczuk, M. Sokółowski, and A. Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 37 Y. Sompolinsky, S. Wyborski, and A. Zohar. Phantom ghostdag: A scalable generalization of Nakamoto consensus: September 2, 2021. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 57–70, New York, NY, USA, 2021.

A Pseudo Codes

■ **Algorithm 11** whether $(u, v) \in E_C$.

```

1: procedure adj $_{G_C}(u, v)$ 
2:    $R := \text{lower\_bounds}(v)$ 
3:    $(p, \text{ignore}) := \text{chain\_index}(u)$ 
4:   if  $u \leq R[p]$  then return 1
5:   else return 0

```

■ **Algorithm 12** return out-neighbors of node v on G_C .

```

1: procedure succ $_{G_C}(v)$ 
2:    $T := \emptyset$ 
3:    $R := \text{lower\_bounds}(v)$ 
4:   for  $p = 0, 1, \dots, k - 1$  do
5:      $c := \text{rank}_p(S, R[p])$ 
6:     for  $i = 1, 2, \dots, c$  do
7:        $u := \text{node\_index}(p, i)$ 
8:        $T \leftarrow T \cup \{u\}$ 
9:   return  $T$ 

```

■ **Algorithm 13** return in-neighbors of node v on G_C .

```

1: procedure pred $_{G_C}(v)$ 
2:    $T := \emptyset$ 
3:    $R := \text{upper\_bounds}(v)$ 
4:   for  $p = 0, 1, \dots, k - 1$  do
5:      $c := \text{rank}_p(S, R[p])$ 
6:      $d := \text{rank}_p(S, n)$ 
7:     for  $i = c, c + 1, \dots, d$  do
8:        $u := \text{node\_index}(p, i)$ 
9:        $T \leftarrow T \cup \{u\}$ 
10:  return  $T$ 

```

■ **Algorithm 14** whether $(u, v) \in E_R$.

```

1: procedure adjGR( $u, v$ )
2:   ( $p, i$ ) := chain_index( $u$ )
3:   ( $q, j$ ) := chain_index( $v$ )
4:   if  $u \neq v$  and  $O_{pq}[i] = I_{qp}[j] = 1$  and  $\text{rank}_1(O_{pq}, i) = \text{rank}_1(I_{qp}, j)$  then
5:     return 1
6:   else
7:     return 0

```

■ **Algorithm 15** return out-neighbors of node v on G_R .

```

1: procedure succGR( $v$ )
2:    $T := \emptyset$ 
3:   ( $p, i$ ) := chain_index( $v$ )
4:   for  $q = 0, 1, \dots, k - 1$  do
5:     if  $O_{pq}[i] = 1$  then
6:        $u := \text{node\_index}(q, \text{nearest\_dst}(p, i, q))$ 
7:        $T \leftarrow T \cup \{u\}$ 
8:   return  $T$ 

```

B Dynamic Data Structures

In BlockDAG application, the required operation to modify posets is only adding to the transitive reduction graph a node with some non-transitive edges outgoing from the new node. There is an algorithm called *peeling* introduced by Daskalakis et al. [15]. Given a poset and its chain decomposition of size $k' \leq 2k$ as inputs, this algorithm reduces the size of the chain decomposition to k one by one in each iteration and returns the minimal chain decomposition. What we have to do in order to realize the adding operation is that add the new node to a new chain, run the iteration of peeling algorithm once to obtain the minimal chain decomposition and modify the data structure.

According to their paper, lookup tables of $lb_p(v)$ and $ub_p(v)$ for each $v \in V$ and $p \in \Sigma$ is required to call peeling algorithm. They named the tables *chainmerge*. Chainmerge can be constructed in $O(nk)$ time for the data structure in Sections 3.2 and 3.4 and $O(nk^2)$ time for the data structure in Section 3.3. One iteration of peeling algorithm takes $O(nk)$ time. Even if we create new strings and bitvectors of the whole data structure, it requires at most $O(nk)$ time when the minimal chain decomposition is given. Therefore, the time complexity of modifying the data structure is absorbed into that of the iteration of peeling algorithm.

■ **Algorithm 16** return in-neighbors of node v on G_R .

```

1: procedure predGR( $v$ )
2:    $T := \emptyset$ 
3:   ( $p, i$ ) := chain_index( $v$ )
4:   for  $q = 0, 1, \dots, k - 1$  do
5:     if  $I_{pq}[i] = 1$  then
6:        $u := \text{node\_index}(q, \text{nearest\_src}(p, i, q))$ 
7:        $T \leftarrow T \cup \{u\}$ 
8:   return  $T$ 

```

33:16 Space-Efficient Data Structure for Posets

■ **Algorithm 17** add to the data structure D a node with edges $(n+1, u)$ for each $u \in U$.

```

1: procedure add( $D, U$ )
2:    $C := \{C_p\}_{p=0}^{k-1}$ 
3:    $C_k := \{n+1\}$ 
4:    $C \leftarrow C \cup C_k$ 
5:   array  $LB[1 \dots n+1][0 \dots k], UB[1 \dots n+1][0 \dots k]$ 
6:   for  $v = 1, 2, \dots, n$  do
7:     for  $p = 0, 1, \dots, k-1$  do
8:        $LB[v][p] \leftarrow lb_p(v)$ 
9:        $UB[v][p] \leftarrow ub_p(v)$ 
10:    for  $p = 0, 1, \dots, k-1$  do
11:       $LB[n+1][p] \leftarrow \max(\{LB[u][p]; u \in U\} \cup (U \cap C_p))$ 
12:       $UB[n+1][p] \leftarrow (p, \infty)$ 
13:    for  $v = 1, 2, \dots, n$  do
14:       $LB[v][k] \leftarrow (k, 0)$ 
15:       $(p, i) := \text{chain\_index}(v)$ 
16:      if  $(p, i) \prec LB[n+1][p]$  then  $UB[v][k] \leftarrow (k, 1)$ 
17:      else  $UB[v][k] \leftarrow (k, \infty)$ 
18:     $LB[n+1][k] \leftarrow (k, 0)$ 
19:     $UB[n+1][k] \leftarrow (k, \infty)$ 
20:    run a peeling iteration on  $LB, UB$  and  $C$  and get a minimal chain decomposition  $C'$ 
21:    construct a new data structure  $D'$  by  $D$  and  $C'$ 
22:    replace  $D$  by  $D'$ 

```

On the data structure of Section 3.2 or Section 3.4, the time complexity of adding nodes is $O(nk)$ per one node. Consider that adding n elements to an empty poset one by one with this adding algorithm. Then, it costs $O(n^2k)$ time. This matches the time complexity to construct the data structure of the poset with n elements, which is $O(n^2k)$ time and is mainly spent by the chain decomposition and conversions between G_R and G_C when its minimal chain decomposition is not given. Algorithm 17 shows the detail of the adding operation. We construct the tables of chainmerge LB and UB from line 5 to line 19.