27th International Symposium on Algorithms and Computation

ISAAC 2016, December 12–14, 2016, Sydney, Australia

Edited by

Seok-Hee Hong

LIPICS

Editors

Seok-Hee Hong
School of Information Technology
University of Sydney, Australia
seokhee.hong@sydney.edu.au

ACM Classification 1998
E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2 Discrete Mathematics, G.4 Mathematical Software, I.1.2 Algorithms, I.2.8 Problem Solving, Control Methods, and Search, I.3.5 Computational Geometry and Object Modeling

ISBN 978-3-95977-026-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-026-2.

Publication date
December 2016, 2016

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

Digital Object Identifier: 10.4230/LIPIcs.ISAAC.2016.0

ISBN 978-3-95977-026-2 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC 2016), held in Sydney, Australia, December 12–14, 2016. ISAAC is an annual international symposium that covers the very wide range of topics in the field of algorithms and computation. The main purpose of the symposium is to provide a forum for researchers working in algorithms and theory of computation from all over the world.

In response to our call for papers, we received 155 submissions from 36 countries. Each submission was reviewed by at least three Program Committee members, possibly with the assistance of external reviewers. After an extremely rigorous review process and extensive discussion, the Program Committee selected 62 papers. Two special issues of Algorithmica and International Journal of Computational Geometry and Applications will publish selected papers from ISAAC 2016.

The best paper award was given to "Optimal Composition Ordering Problems for Piecewise Linear Functions" by Yasushi Kawase, Kazuhisa Makino and Kento Seimi. Selected from submissions authored by students only, the best student paper award was given to "Adaptivity vs. Postselection, and Hardness Amplification in Polynomial Approximation" by Lijie Chen.

In addition to selected papers, the program also included invited talks by two prominent invited speakers, Xuemin Lin, University of NSW, Australia, and Kunsoo Park, Seoul National University, Korea.

We thank all the Program Committee members and external reviewers for their professional service and volunteering their time to review the submissions under time constraints. We also thank all authors who submitted papers for consideration, thereby contributing to the high quality of the conference. We would like also to acknowledge our supporting organizations for their assistance and support, in particular the University of Sydney and the NSW Department of Industry, through the NSW Office of Science and Research. Finally, we are deeply indebted to the Organizing Committee members, Peter Eades and Amyra Meidiana, whose excellent effort and professional service to the community made the conference an unparalleled success.

December 2016 Seok-Hee Hong

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Seok-Hee Hong (chair)	University of Sydney, Australia
Ulrik Brandes	University of Konstanz, Germany
Xiaotie Deng	Shanghai Jiao Tong University, China
Thomas Erlebach	University of Leicester, UK
William Evans	UBC, Canada
Rudolf Fleischer	GUtech, Oman
Fabrizio Frati	Roma Tre University, Italy
Takuro Fukunaga	National Institute of Informatics, Japan
Serge Gaspers	UNSW Australia and Data61, CSIRO, Australia
Michael Goodrich	University of California, Irvine, USA
Hiroshi Imai	University of Tokyo, Japan
Toshimasa Ishii	Hokkaido University, Japan
Giuseppe F. Italiano	University of Rome "Tor Vergata", Italy
Takehiro Ito	Tohoku University, Japan
Ming-Yang Kao	Northwestern University, USA
Michael Kaufmann	Tübingen University, Germany
Jan Kratochvil	Charles University, Czech Republic
Minming Li	City University of Hong Kong, Hong Kong
Giuseppe Liotta	University of Perugia, Italy
Hsueh-I Lu	National Taiwan University, Taiwan
Bernard Mans	Macquarie University, Australia
Sang-Il Oum	KAIST, South Korea
Evanthia Papadopoulou	University of Lugano (USI), Switzerland
Kunsoo Park	Seoul National University, South Korea
Md. Saidur Rahman	Bangladesh University of Engineering and Technology, Bangladesh
Ignaz Rutter	Karlsruhe Institute of Technology (KIT), Germany
Kunihiko Sadakane	The University of Tokyo, Japan
Pascal Schweitzer	RWTH Aachen University, Germany
Shin-Ichi Tanigawa	Kyoto University, Japan
Takeshi Tokuyama	Tohoku University, Japan
Takeaki Uno	National Institute of Informatics, Japan
Yushi Uno	Osaka Prefecture University, Japan
Osamu Watanabe	Tokyo Institute of Technology, Japan
Anthony Wirth	The University of Melbourne, Australia
Hsu-Chun Yen	National Taiwan University, Taiwan

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Abboud, Amir
Adnan, Muhammad Abdullah
Agrawal, Akanksha
Angelini, Patrizio
Angelopoulos, Spyros
Auletta, Vincenzo
Banik, Aritra
Barba, Luis
Bei, Xiaohui
Bekos, Michael
Belmonte, Rémy
Bhaskara, Aditya
Binucci, Carla
Bläsius, Thomas
Chang, Yi-Jun
Chau, Vincent
Chen, Ruiwen
Chen, Hao
Chen, Ho-Lin
Chen, Li
Chiu, Man Kwun
Das, Shantanu
Datta, Samir
Dehghani, Sina
Didimo, Walter
Donato, Valentino Di
Donoso, Yago Diez
Durocher, Stephane
Engels, Christian
Fagerberg, Rolf
Franciosa, Paolo
Fujiwara, Hiroshi
Gajarský, Jakub
Georgiadis, Loukas
Giacomo, Emilio Di
Göös, Mika
Grilli, Luca
Han, Xin
Haraguchi, Kazuya
Hasunuma, Toru
Hatanaka, Tatsuhiko
Heinsohn, Niklas
Higashikawa, Yuya
Hirahara, Shuichi
Hon, Wing-Kai
Huang, Shenwei
Huang, Shang-En
Jones, Mark
Kakimura, Naonori
Kawachi, Akinori
Kawamura, Yasuyuki
Kawase, Yasushi
Keil, Mark
Kijima, Shuji
Kiyomi, Masashi
Koabayashi, Yusuke
Kobayashi, Yusuke
Koike, Atsushi
Korman, Matias
Koshiba, Takeshi
Kriege, Nils
Křivka, Zbyněk
Kucera, Petr
Kumar, Nirman
Lai, Kai-Yuan
Lampis, Michael
Liao, Chung-Shou
Liu, Chih-Hung
Loitzenbauer, Veronika
Lozzo, Giordano Da
Mathieson, Luke
Mccauley, Samuel
Mchedlidze, Tamara
Mehta, Aranyak
Meijer, Henk
Misra, Neeldhara
Mnich, Matthias
Mondal, Debajyoti
Montecchiani, Fabrizio
Moriyama, Sonoko
Mouawad, Amer
Mulzer, Wolfgang
Naamad, Yonatan
Näher, Stefan
Najeebullah, Kamran
Nguyen, Trung Thanh
Niedermann, Benjamin
Nilsson, Bengt J.
Nishat, Rahnuma Islam
Okamoto, Yoshio

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ono, Hirotaka
Onodera, Taku
Otachi, Yota
Ozeki, Kenta
Palios, Leonidas
Parotsidis, Nikos
Pasquale, Francesco
Poon, Sheung-Hung
Radermacher, Marcel
Rahman, Atif
Roeloffzen, Marcel
Saitoh, Toshiki
Satti, Srinivasa Rao
Sauerhoff, Martin
Schnoor, Henning
Sedeño-Noda, Antonio
Shen, Liang-Hsin
Shibuya, Tetsuo
Shioura, Akiyoshi
Shurbevski, Aleksandar
Silveira, Rodrigo
Singh, Shikha
Strash, Darren
Su, Hsin-Hao
Suzuki, Akira

Takazawa, Kenjiro
Talebanfard, Navid
Tamaki, Suguru
Teague, Vanessa
Thaler, Justin
Thankachan, Sharma V.
Thierauf, Thomas
van Bevern, René
van Leeuwen, Erik Jan
van Renssen, André
Varadarajan, Kasturi
Wang, Haitao
Wang, Bow-Yaw
Wasa, Kunihiro
Watson, Thomas
Wilmes, John
Wismath, Steve
Wu, Weiwei
Yamanaka, Katsuhisa
Yasunaga, Kenji
Zhang, Jialin
Zhang, Lele
Zhao, Yingchao
Zündorf, Tobias

Towards Processing of Big Graphs: from Theory, Algorithm to System

Xuemin Lin

University of New South Wales, Sydney, Australia
lxue@cse.unsw.edu.au

Abstract

Graphs are very important parts of Big Data and widely used for modelling complex structured data with a broad spectrum of applications such as bioinformatics, web search, social network, road network, etc. Over the last decade, tremendous research efforts have been devoted to many fundamental problems in managing and analysing graph data. In this talk, I will present some of our recent research efforts in processing big graphs including scalable processing theory and techniques, distributed computation, and system framework.

1998 ACM Subject Classification H.2.8 Database Applications, H.2.4 Systems

Keywords and phrases Graph Processing, Big Data, Cloud Computing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.1

Category Invited Talk

Compressed and Searchable Indexes for Highly Similar Strings[*]

Kunsoo Park

Dept. of Computer Science and Engineering, Seoul National University, Korea
kpark@theory.snu.ac.kr

Abstract

The collection indexing problem is defined as follows: Given a collection of highly similar strings, build a compressed index for the collection of strings, and when a pattern is given, find all occurrences of the pattern in the given strings. Since the index is compressed, we also need a separate operation which retrieves a specified substring of one of the given strings.

Such a collection of highly similar strings can be found in genome sequences of a species and in documents stored in a version control system. Many indexes for the collection indexing problem have been developed, most of which use classical compression schemes such as run-length encoding and Lempel-Ziv compressions to exploit the similarity of the given strings.

We introduce a new index for highly similar strings, called FM index of alignment. We start by finding common regions and non-common regions of highly similar strings. We need not find a multiple alignment of non-common regions. Finding common and non-common regions is much easier and simpler than finding a multiple alignment. Then we make a transformed alignment of the given strings, where gaps in a non-common region are put together into one gap. We define a suffix array of alignment on the transformed alignment, and the FM index of alignment is an FM index of this suffix array of alignment. The FM index of alignment supports the LF mapping and backward search, the key functionalities of the FM index. The FM index of alignment takes less space than other indexes and its pattern search is also fast.

1998 ACM Subject Classification E.1 [Data Structures] Arrays, Tables, F.2 Analysis of Algorithms and Problem Complexity, F.2.2 [Nonnumerical Algorithms and Problems] Pattern matching

Keywords and phrases Index for similar strings, FM index, Suffix array, Alignment

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.2

Category Invited Talk

[*] This research was supported by the Bio & Medical Technology Development Program of the NRF funded by the Korean government, MSIP (NRF-2014M3C9A3063541).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Streaming Verification of Graph Properties[*][†]

Amirali Abdullah[1], Samira Daruki[2], Chitradeep Dutta Roy[3], and Suresh Venkatasubramanian[4]

1 Department of Mathematics, University of Michigan, USA
2 School of Computing, University of Utah, USA
3 School of Computing, University of Utah, USA
4 School of Computing, University of Utah, USA

Abstract

Streaming interactive proofs (SIPs) are a framework for outsourced computation. A computationally limited streaming client (the verifier) hands over a large data set to an untrusted server (the prover) in the cloud and the two parties run a protocol to confirm the correctness of result with high probability. SIPs are particularly interesting for problems that are hard to solve (or even approximate) well in a streaming setting. The most notable of these problems is finding maximum matchings, which has received intense interest in recent years but has strong lower bounds even for constant factor approximations. In this paper, we present efficient streaming interactive proofs that can verify maximum matchings *exactly*. Our results cover all flavors of matchings (bipartite/non-bipartite and weighted). In addition, we also present streaming verifiers for approximate metric TSP. In particular, these are the first efficient results for weighted matchings and for metric TSP in any streaming verification model.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases streaming interactive proofs, verification, matching, travelling salesman problem, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.3

1 Introduction

The shift from direct computation to outsourcing in the cloud has led to new ways of thinking about massive scale computation. In the *verification* setting, computational effort is split between a computationally weak client (the *verifier*) who owns the data and wants to solve a desired problem, and a more powerful server (the *prover*) which performs the computations. Here the client has only limited (streaming) access to the data, as well as a bounded ability to talk with the server (measured by the amount of communication), but wishes to verify the correctness of the prover's answers. This model can be viewed as a streaming modification of a classic interactive proof system (a streaming IP, or SIP), and has been the subject of a number of papers [26, 47, 23, 17, 22, 16, 38, 39] that have established sublinear (verifier) space and communication bounds for classic problems in streaming and data analysis.

In this paper, we present streaming interactive proofs for graph problems that are traditionally hard for streaming, such as for the maximum matching problem (in bipartite and general graphs, both weighted and unweighted) as well for approximating the traveling

[*] A full version of the paper is available at `http://arxiv.org/abs/1602.08162`.
[†] This research was supported in part by the NSF under grants IIS-1251049, CNS-1302688.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 3; pp. 3:1–3:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Our Results. All bounds expressed in bits, upto constant factors. For the matching results, $\rho = \min(n, C)$ where C is the cardinality of the optimal matching (weighted or unweighted). Note that for the MST, the verification is for a $(1+\epsilon)$-approximation. For the TSP, the verification is for a $(3/2+\varepsilon)$-approximation. (*) γ' is a linear function of γ and is strictly more than 1 as long as γ is a sufficiently large constant.

	$\log n$ rounds		$\gamma = O(1)$ rounds	
Problem	Verifier Space	Communication	Verifier Space	Communication
Triangle Counting	$\log^2 n$	$\log^2 n$	$\log n$	$n^{1/\gamma} \log n$
Matchings (all versions)	$\log^2 n$	$(\rho + \log n) \log n$	$\log n$	$(\rho + n^{1/\gamma'}) \log n$ (*)
Connectivity	$\log^2 n$	$n \log n$	$\log n$	$n \log n$
Minimum Spanning Tree	$\log^2 n$	$n \log^2 n/\varepsilon$	$\log n$	$n \log^2 n/\varepsilon$
Travelling Salesperson	$\log^2 n$	$n \log^2 n/\varepsilon$	$\log n$	$n \log^2 n/\varepsilon$

salesperson problem. In particular, we present protocols that verify a matching *exactly* in a graph using polylogarithmic space and polylogarithmic communication apart from the matching itself. In all our results, we consider the input in the *dynamic streaming model*, where graph edges are presented in arbitrary order in a stream and we allow both deletion and insertion of edges. All our protocols use either $\log n$ rounds of communication or (if the output size is sufficiently large or we are willing to tolerate superlogarithmic communication) constant rounds of communication.

To prove the above results, we also need SIPs for sub-problems like connectivity, minimum spanning tree and triangle counting. While it is possible to derive similar (and in some cases better) results for these subroutines using known techniques [30], we require explicit protocols that return structures that can be used in the computation pipeline for the TSP. Furthermore, our protocols for these problems are much simpler than what can be obtained by techniques in [30], which require some effort to obtain precise bounds on the size and depth of the circuits corresponding to more complicated parallel algorithms. We summarize our results in Table 1. Due to space constraints subproblems like triangle counting, connectivity, bipartiteness and MST are presented in the full version [1].

Significance of our Results

While the streaming model of computation has been extremely effective for processing numeric and matrix data, its ability to handle large graphs is limited, even in the so-called *semi-streaming* model where the streaming algorithm is permitted to use space quasilinear in the number of vertices. Recent breakthroughs in graph sketching [43] have led to space-efficient approximations for many problems in the semi-streaming model but canonical graph problems like matchings have been shown to be provably hard.

It is known [36] that no better than a $1 - 1/e$ approximation to the maximum cardinality matching is possible in the streaming model, even with space $\tilde{O}(n)$. It was also known that even allowing limited communication (effectively a single message from the prover) required a space-communication product of $\Omega(n^2)$ [16, 22]. Our results show that even allowing a few more rounds of communication dramatically improves the space-communication tradeoff for matching, as well as yielding *exact* verification. We note that streaming algorithms for matching vary greatly in performance and complexity depending in whether the graph is weighted or unweighted, bipartite or nonbipartite. In contrast, our results apply to all forms of matching. Interestingly, the special case of *perfect matching*, by virtue of being in RNC [37], admits an efficient SIP via results by Goldwasser, Kalai and Rothblum [30] and Cormode, Thaler and Yi [23]. Similarly for triangle counting, the best streaming algorithm [5] yields an additive εn^3 error estimate in polylogarithmic space, and again in the annotation model (effectively a single round of communication) the best result yields a space-communication tradeoff of $n^2 \log^2 n$, which is almost exponentially worse than the bound we obtain. We note that counting triangles is a classic problem in the sublinear algorithms literature, and identifying optimal space and communication bounds for this problem was posed as an open problem by Graham Cormode in the Bertinoro sublinear algorithms workshop [21]. Our bound for verifying a $3/2 + \epsilon$ approximation for the TSP in dynamic graphs is also interesting: a trivial 2-approximation in the semi-streaming model follows via the MST, but it is open to improve this bound (even on a grid) [46].

In general, our results can be viewed as providing further insight into the tradeoff between space and communication in sublinear algorithms. The annotation model of verification provides $\Omega(n^2)$ lower bounds on the space-communication product for the problems we consider: in that light, the fact that we can obtain polynomially better bounds with only constant number of rounds demonstrates the power of just a few rounds of interaction. We note that as of this paper, virtually all of the canonical hard problems for streaming algorithms (Index [17], Disjointness [9, 10], Boolean Hidden Matching [28, 15, 40]) admit efficient SIPs. A SIP for Index was presented in [17] and we present SIPs for Disjointness and Boolean Hidden Matching in the full version [1]. Our model is also different from a standard multi-pass streaming framework, since communication must remain sublinear in the input and in fact in all our protocols the verifier still reads the input exactly once.

From a technical perspective, our work continues the *sketching* paradigm for designing efficient graph algorithms. All our results proceed by building linear sketches of the input graph. The key difference is that our sketches are not approximate but algebraic: based on random evaluation of polynomials over finite fields. Our sketches use higher dimensional linearization ("tensorization") of the input, which might itself be of interest. They also compose: indeed, our solutions are based on building a number of simple primitives that we combine in different ways.

2 Related Work

Outsourced computation

Work on outsourced computation comes in three other flavors in addition to SIPs: firstly, there is work on reducing the verifier and prover complexity without necessarily making the verifier a sublinear algorithm [30, 29, 35], in some cases using cryptographic assumptions to achieve their bounds. Another approach is the idea of *rational proofs* [8, 19, 32, 31], in which the verifier uses a payment function to give the prover incentive to be honest. Moving to sublinear verifiers, there has been research on designing SIPs where the verifier runs in sublinear *time* [33, 45].

Streaming Graph Verification

All prior work on streaming graph verification has been in the annotation model, which in practice resembles a 1-round SIP (a single message from prover to verifier after the stream has been read). In recent work, Thaler [47] gives protocols for counting triangles, and computing maximum cardinality matching with both $n \log n$ space and communication cost. For matching, Chakrabarti *et al.* [16] show that any annotation protocol with space cost $O(n^{1-\delta})$ requires communication cost $\Omega(n^{1+\delta})$ for any $\delta > 0$. They also show that any annotation protocol for graph connectivity with space cost $O(n^{1-\delta})$ requires communication cost $\Omega(n^{1+\delta})$ for any $\delta > 0$.

It is also proved that every protocol for this problem in the annotation model requires $\Omega(n^2)$ product of space and communication. This is optimal upto logarithmic factors. Furthermore, they conjecture that achieving smooth tradeoffs between space and communication cost is impossible, i.e. it is not known how to reduce the space usage to $o(n \log n)$ without blowing the communication cost up to $\Omega(n^2)$ or vice versa [16, 47]. Note that in all our protocols, the product of space and communication is $O(n \operatorname{poly} \log n)$.

Streaming Graph Algorithms

In the general dynamic streaming model, $\operatorname{poly} \log 1/\varepsilon$-pass streaming algorithms [2, 3] give $(1+\varepsilon)$-approximate answers and require $\tilde{O}(n)$ space in one pass. The best results for matching are [20] (a parametrized algorithm for computing a maximal matching of size k using $\tilde{O}(nk)$ space) and [7, 42] which gives a streaming algorithm for recovering an n^{ϵ}-approximate maximum matching by maintaining a linear sketch of size $\tilde{O}(n^{2-3\epsilon})$ bits. In the single-pass insert-only streaming model, Epstein et al. [27] give a constant (4.91) factor approximation for weighted graphs using $O(n \log n)$ space. Crouch and Stubbs [24] give a $(4+\epsilon)$-approximation algorithm which is the best known result for weighted matchings in this model. Triangle counting in streams has been studied extensively [11, 13, 14, 34, 44]. For dynamic graphs, the most space-efficient result is the one by [5] that provides the aforementioned additive εn^3 bound in polylogarithmic space. The recent breakthrough in sketch-based graph streaming [4] has yielded $\tilde{O}(n)$ semi-streaming algorithms [43] for computing the connectivity, bipartiteness and minimum spanning trees of dynamic graphs.

3 Preliminaries

We will work in the *streaming interactive proof* (SIP) model first proposed by Cormode et al. [23]. In this model, there are two players, the prover P and the verifier V. The input consists of a *stream* τ of items from a universe $\mathcal{U}$. Let f be a function mapping τ to any finite set $\mathcal{S}$. A k-message SIP for f works as follows:

1. V and P read the input stream and perform some computation on it.
2. V and P then exchange k messages, after which V either outputs a value in $\mathcal{S} \cup \{\perp\}$, where $\perp$ denotes that V is not convinced that the prover followed the prescribed protocol.

V is randomized. There must exist a prover strategy that causes the verifier to output $f(\tau)$ with probability $1 - \varepsilon_c$ for some $\varepsilon_c \leq 1/3$. Similarly, for all prover strategies, V must output a value in $\{f(\tau), \perp\}$ with probability $1 - \varepsilon_s$ for some $\varepsilon_s \leq 1/3$. The values ε_c and ε_s are respectively referred to as the completeness and soundness errors of the protocol. The protocols we design here will have perfect completeness ($\varepsilon_c = 0$).[1] We note that the annotated stream model of Chakrabarti et al. [16] essentially corresponds to one-message SIPs.[2]

[1] The constant 1/3 appearing in the completeness and soundness requirements is chosen by convention [6]. The constant 1/3 can be replaced with any other constant in $(0, 1)$ without affecting the theory in any way.

[2] Technically, the annotated data streaming model allows the annotation to be interleaved with the stream updates, while the SIP model does not allow the prover and verifier to communicate until after the stream has passed. However, almost all known annotated data streaming protocols do not utilize the ability to interleave the annotation with the stream, and hence are actually 1-message SIPs, but without any interaction from the verifier to prover side.

Input Model

We will assume the input is presented as stream updates to a vector. In general, each element of this stream is a tuple (i, δ), where each i lies in a universe $\mathcal{U}$ of size u, and $\delta \in \{+1, -1\}$. The data stream implicitly defines a frequency vector $\mathbf{a} = (a_1, \ldots, a_u)$, where a_i is the sum of all δ values associated with i in the stream. The stream update (i, δ) is thus the implicit update $\mathbf{a}[i] \leftarrow \mathbf{a}[i] + \delta$. In this paper, the stream consists of edges drawn from $\mathcal{U} = [n] \times [n]$ along with weight information as needed. As is standard, we assume that edge weights are drawn from $[n^c]$ for some constant c. We allow edges to be inserted and deleted but the final edge multiplicity is 0 or 1, and also mandate that the length of the stream is polynomial in n. Finally, for weighted graphs, we further constrain that the edge weight updates be atomic, i.e. that an edge along with its full weight be inserted or deleted at each step.

There are three parameters that control the complexity of our protocols: the vector length u, the length of stream s and the maximum size of a coordinate $M = max_i \mathbf{a}_i$. In the protocols discussed in this paper M will always be upper bounded by some polynomial in u, i.e. $\log M = O(\log u)$. All algorithms we present use linear sketches, and so the stream length s only affects verifier running time. In full version of the paper [1] we discuss how to reduce the verifier update time to polylogarithmic on each step.

Costs

A SIP has two costs: the verifier space, and the total communication, expressed as the number of bits exchanged between V and P. We will use the notation (A, B) to denote a SIP with verifier space $O(A)$ and total communication $O(B)$. We will also consider the number of rounds of communication between V and P. The basic versions of our protocols will require $\log n$ rounds, and we later show how to improve this to a constant number of rounds while maintaining the same space and similar communication cost otherwise.

4 Overview of our Techniques

For all the problems that we discuss the input is a data stream of edges of a graph where for an edge e an element in the stream is of the form (i, j, Δ). Now all our protocols proceed as follows. We define a domain $\mathcal{U}$ of size u and a frequency vector $\mathbf{a} \in \mathbb{Z}^u$ whose entries are indexed by elements of $\mathcal{U}$. A particular protocol might define a number of such vectors, each over a different domain. Each stream element will trigger a set of indices from $\mathcal{U}$ at which to update $\mathbf{a}$. For example in case of matching, we derive this constraint universe from the LP certificate, whereas for counting triangles our universe is derived from all $O(n^3)$ possible three-tuples of the vertices.

The key idea in all our protocols is that since we cannot maintain $\mathbf{a}$ explicitly due to limited space, we instead maintain a linear *sketch* of $\mathbf{a}$ that varies depending on the problem being solved. This sketch is computed as follows. We will design a polynomial that acts as a *low-degree* extension of f over an extension field $\mathbb{F}$ and can be written as $p(x_1, \ldots, x_d) = \sum_{u \in \mathcal{U}} a[u] g_u(x_1, x_2, \ldots, x_d)$. The crucial property of this polynomial is that it is *linear* in the entries of $\mathbf{a}$. This means that polynomial evaluation at any fixed point $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ is easy in a stream: when we see an update $a[u] \leftarrow a[u] + \Delta$, we merely need to add the expression $\Delta g_u(\mathbf{r})$ to a running tally. Our sketch will always be a polynomial evaluation at a *random* point $\mathbf{r}$. Once the stream has passed, V and the prover P will engage in a conversation that might involve further sketches as well as further updates to the current sketch. In our descriptions, we will use the imprecise but convenient shorthand "increment

$\mathbf{a}[u]$" to mean "update a linear sketch of some low-degree extension of a function of $\mathbf{a}$". It should be clear in each context what the specific function is.

As mentioned earlier, a single stream update of the form (i, j, Δ) might trigger updates in many entries of $\mathbf{a}$, each of which will be indexed by a multidimensional vector. We will use the wild-card symbol '$*$' to indicate that all values of that coordinate in the index should be considered. For example, suppose $\mathcal{U} \subseteq [n] \times [n] \times [n]$. The instruction "update $\mathbf{a}[(i, *, j)]$" should be read as "update all entries $\mathbf{a}[t]$ where $t \in \{(i, s, j) \mid s \in [n], (i, s, j) \in \mathcal{U}\}$". We show later how to do these updates implicitly, so that verifier time remains suitably bounded.

5 Some Useful Protocols

We will make use of two basic tools in our algorithms: Reed-Solomon fingerprints for testing vector equality, and the streaming SumCheck protocol of Cormode et al. [23]. We summarize the main properties of these protocols here: for more details, the reader is referred to the original papers.

Multi-Set Equality (MSE)

We are given streaming updates to the entries of two vectors $\mathbf{a}, \mathbf{a}' \in \mathbb{Z}^u$ and wish to check $\mathbf{a} = \mathbf{a}'$. Reed-Solomon fingerprinting is a standard technique to solve MSE using only logarithmic space.

▶ **Theorem 1** (MSE, [22]). *Suppose we are given stream updates to two vectors $\mathbf{a}, \mathbf{a}' \in \mathbb{Z}^u$ guaranteed to satisfy $|\mathbf{a}_i|, |\mathbf{a}'_i| \leq M$ at the end of the data stream. Let $t = \max(M, u)$. There is a streaming algorithm using $O(\log t)$ space, satisfying the following properties: (i) If $\mathbf{a} = \mathbf{a}'$, then the streaming algorithm outputs 1 with probability 1. (ii) If $\mathbf{a} \neq \mathbf{a}'$, then the streaming algorithm outputs 0 with probability at least $1 - 1/t^2$.*

The SumCheck Protocol

We are given streaming updates to a vector $\mathbf{a} \in \mathbb{Z}^u$ and a univariate polynomial $h \colon \mathbb{Z} \to \mathbb{Z}$. The Sum Check problem (SumCheck) is to verify a claim that $\sum_i h(\mathbf{a}_i) = K$.

▶ **Lemma 2** (SumCheck, [23]). *There is a SIP to verify that $\sum_{i \in [u]} h(\mathbf{a}_i) = K$ for some claimed K. The total number of rounds is $O(\log u)$ and the cost of the protocol is $(\log(u) \log |\mathbb{F}|, deg(h) \log(u) \log |\mathbb{F}|)$.*

Here are the two other protocols that act as building blocks for our graph verification protocols.

Inverse Protocol (Finv)

Let $\mathbf{a} \in \mathbb{Z}^u$ be a (frequency) vector. The *inverse frequency* function F_k^{-1} for a fixed k is the number of elements of $\mathbf{a}$ that have frequency k: $F_k^{-1}(\mathbf{a}) = |\{i \mid \mathbf{a}_i = k\}|$. Let $h_k(i) = 1$ for $i = k$ and 0 otherwise. We can then define $F_k^{-1}(\mathbf{a}) = \sum_i h_k(\mathbf{a}_i)$. Note that the domain of h_k is $[M]$ where $M = \max_i \mathbf{a}_i$. We will refer to the problem of verifying a claimed value of F_k^{-1} as Finv. By using Lemma 2, there is a simple SIP for Finv. We restate the related results here [23].

▶ **Lemma 3** (Finv, [23]). *Given stream updates to a vector $\mathbf{a} \in \mathbb{Z}^u$ such that $\max_i \mathbf{a}_i = M$ and a fixed integer k there is a SIP to verify the claim $F_k^{-1}(\mathbf{a}) = K$ with cost $(\log^2 u, M \log^2 u)$ in $\log u$ rounds.*

Remark 1. Note that the same result holds if instead of verifying an inverse query for a single frequency k, we wish to verify it for a set of frequencies. Let $S \subset [M]$ and let $F_S^{-1} = |\{i|\mathbf{a}_i \in S\}|$. Then using the same idea as above, there is a SIP for verifying a claimed value of F_S^{-1} with costs given by Lemma 3.

Remark 2. Note that in the protocols presented in this paper later, the input to the FINV is not the graph edges itself, but instead the FINV is applied to the derived stream updates triggered by each input stream elements. As stated before, a single stream update of the form (i, j, Δ) might trigger updates in many entries of vector $\mathbf{a}$, which is defined based on the problem.

Subset Protocol

We now present a new protocol for a variant of the vector equality test described in Theorem 1. While this problem has been studied in the annotation model, it requires space-communication product of $\Omega(u^2)$ communication in that setting.

▶ **Lemma 4** (SUBSET). *Let $E \subset [u]$ be a set of elements, and let $S \subset [u]$ be another set owned by* P*. There is a SIP to verify a claim that $S \subset E$ with cost $(\log^2 u, (|S| + \log u) \log u)$ in $\log u$ rounds.*

Proof. Consider a vector $\bar{\mathbf{a}}$ with length u, in which the verifier does the following updates: for each element in set E, increment the corresponding value in vector $\bar{\mathbf{a}}$ by $+1$ and for each element in set S, decrements the corresponding value in vector $\bar{\mathbf{a}}$ by -1. Let the vector $\mathbf{a} \in \{0, 1\}^u$ be the characteristic vector of E, and let $\mathbf{a}'$ be the characteristic vector of S. Thus, $\bar{\mathbf{a}} = \mathbf{a} - \mathbf{a}'$. By applying F_{-1}^{-1} protocol on $\bar{\mathbf{a}}$, verifier can determine if $S \subset E$ or not. Note that in vector $\bar{\mathbf{a}}$, $M = 1$. Then the protocol cost follows by Lemma 3. ◀

6 SIP for MAX-MATCHING in Bipartite Graphs

We now present a SIP for maximum cardinality matching in bipartite graphs. The prover P needs to generate two certificates: an actual matching, and a proof that this is optimal. By König's theorem [41], a bipartite graph has a maximum matching of size k if and only if it has a minimum vertex cover of size k. Therefore, P's proof consists of two parts:

(a) Send the claimed optimal matching $M \subset E$ of size k.
(b) Send a vertex cover $S \subset V$of size k.

V has three tasks:

(i) Verify that M is a matching and that $M \subset E$.
(ii) Verify that S covers all edges in E.
(iii) Verify that $|M| = |S|$.

We describe protocols for first two tasks and the third task is trivially solvable by counting the length of the streams and can be done in $\log n$ space. V will run the three protocols in parallel.

Verifying a Matching

Verifying that $M \subset E$ can be done by running the SUBSET protocol from Lemma 4 on E and the claimed matching M. A set of edges M is a matching if each vertex has degree at most 1 on the subgraph defined by M. Interpreted another way, let τ_M be the stream of endpoints of edges in M. Then each item in τ_M must have frequency 1. This motivates the following protocol, based on Theorem 1. V treats τ_M as a sequence of updates to a frequency vector $\mathbf{a} \in \mathbb{Z}^{|V|}$ counting the number of occurrences of each vertex. V then asks P to send a stream of all the vertices incident on edges of M as updates to a different frequency vector $\mathbf{a}'$. V then runs the MSE protocol to verify that these are the same.

Verifying that S is a Vertex Cover

The difficulty with verifying a vertex cover is that V no longer has streaming access to E. However, we can once again reformulate the verification in terms of frequency vectors. S is a vertex cover if and only if each edge of E is incident to some vertex in S. Let $\mathbf{a}, \mathbf{a}' \in \mathbb{Z}^{\binom{n}{2}}$ be vectors indexed by $\mathcal{U} = \{(i,j), i,j \in V, i < j\}$. On receiving the input stream edge $e = (i,j,\Delta), i < j$, V increments $\mathbf{a}[(i,j)]$ by Δ.

For each vertex $i \in S$ that P sends, we increment all entries $\mathbf{a}'[(i,*)]$ and $\mathbf{a}'[(*,i)]$. Now it is easy to see that S is a vertex cover if and only there are no entries in $\mathbf{a} - \mathbf{a}'$ with value 1 (because these entries correspond to edges that have *not* been covered by a vertex in S). This yields the following verification protocol.

1. V processes the input edge stream for the F_1^{-1} protocol, maintaining updates to a vector $\mathbf{a}$.
2. P sends over a claimed vertex cover S of size c^* one vertex at a time. For each vertex $i \in S$, V *decrements* all entries $\mathbf{a}[(i,*)]$ and $\mathbf{a}[(*,i)]$.
3. V runs FINV to verify that $F_1^{-1}(\mathbf{a}) = 0$.

The bounds for this protocol follow from Lemmas 3, 4 and Theorem 1.

▶ **Theorem 5.** *Given an input bipartite graph with n vertices, there exists a streaming interactive protocol for verifying the maximum-matching with $\log n$ rounds of communication, and cost $(\log^2 n, (c^* + \log n)\log n)$, where c^* is the size of the optimal matching.*

7 SIP for Maximum-Weight-Matching in General Graphs

We now turn to the most general setting: of maximum weight matching in general graphs and the bipartite case is moved to the full version [1]. This of course subsumes the easier case of maximum cardinality matching in general graphs, and while there is a slightly simpler protocol for that problem based on the Tutte-Berge characterization of maximum cardinality matchings [48, 12], we will not discuss it here.

We will use the odd-set based LP-duality characterization of maximum weight matchings due to Cunningham and Marsh. Let $\mathbb{O}(V)$ denote the set of all odd-cardinality subsets of V. Let $y_i \in [n^c]$ define non-negative integral weight on vertex v_i, $z_U \in [n^c]$ define a non-negative integral weight on an *odd-cardinality* subset $U \in \mathbb{O}(V)$, $w_{ij} \in [n^c]$ define the weight of an edge $e = (i,j)$ and $c^* \in [n^{c+1}]$ be the weight of a maximum weight matching on G. We define y and z to be *dual feasible* if $y_i + y_j + \sum_{\substack{U \in \mathbb{O}(V) \\ i,j \in U}} z_U \geq w_{i,j}, \forall i,j$.

A collection of sets is said to be *laminar*, if any two sets in the collection are either disjoint or nested (one is contained in the other). Note that such a family must have size linear in the size of the ground set. Standard LP-duality and the Cunningham-Marsh theorem state that:

▶ **Theorem 6** ([25]). *For every integral set of edge weights W, and choices of dual feasible integral vectors y and z, $c^* \leq \sum_{v \in V} y_v + \sum_{U \in \mathbb{O}(V)} z_U \left\lfloor \frac{1}{2}|U| \right\rfloor$. Furthermore, there exist vectors y and z that are dual feasible such that $\{U : z_U > 0\}$ is laminar and for which the above upper bound achieves equality.*

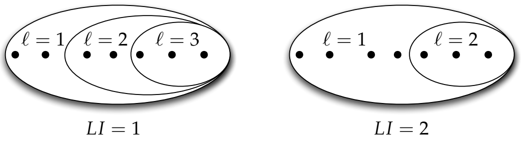


Figure 1 A Laminar family.

We design a protocol that will verify that each dual edge constraint is satisfied by the dual variables. The laminar family $\{U : z_U > 0\}$ can be viewed as a collection of nested subsets (each of which we call a *claw*) that are disjoint from each other. Within each claw, a set U can be described by giving each vertex v in order of increasing *level* $\ell(v)$: the number of sets v is contained in (see Figure 1).

The prover will describe a set U and its associated z_U by the tuple $(LI, \ell, r_U, \partial U)$, where $1 \leq LI \leq n$ is the index of the claw U is contained in, $\ell = \ell(U)$, $r_U = \sum_{U' \supseteq U'} z_{U'}$ and $\partial U = U \setminus \cup_{U'' \subset U} U''$. For an edge $e = (i, j)$ let $r_e = \sum_{i,j \in U, U \in \mathbb{O}(V)} z_U$ represent the weight assigned to an edge by weight vector z on the laminar family. Any edge whose endpoints lie in different claws will have $r_e = 0$. For a vertex v, let $r_v = \min_{v \in U} r_U$. For an edge $e = (v, w)$ whose endpoints lie in the same claw, it is easy to see that $r_e = \min(r_v, r_w)$, or equivalently that $r_e = r_{\arg\min(\ell(v), \ell(w))}$. For such an edge, let $\ell_{e,\downarrow} = \min(\ell(u), \ell(v))$ and $\ell_{e,\uparrow} = \max(\ell(u), \ell(v))$. We will use $LI(e) \in [n]$ to denote the index of the claw that the endpoints of e belong to.

The Protocol

V prepares to make updates to a vector $\mathbf{a}$ with entries indexed by $\mathcal{U} = \mathcal{U}_1 \cup \mathcal{U}_2$. $\mathcal{U}_1$ consists of all tuples of the form $\{(i, j, w, y, y', LI, \ell, \ell', r)\}$ and $\mathcal{U}_2$ consists of all tuples of the form $\{(i, j, w, y, y', 0, 0, 0, 0)\}$ where $i < j, i, j, LI, \ell, \ell' \in [n]$, $y, y', r, w \in [n^c]$ and tuples in $\mathcal{U}_1$ must satisfy 1) $w \leq y + y' + r$ and 2) it is *not* simultaneously true that $y + y' \geq w$ and $r > 0$. Note that $\mathbf{a} \in \mathbb{Z}^u$ where $u = O(n^{4c+5})$ and all weights are bounded by n^c.

1. V prepares to process the stream for an F_5^{-1} query. When V sees an edge update of form (e, w_e, Δ), it updates all entries $\mathbf{a}[(e, w_e, *, *, *, *, *, *)]$.
2. P sends a list of vertices (i, y_i) in order of increasing i. For each (i, y_i), V increments by 1 the count of all entries $\mathbf{a}[(i, *, *, y_i, *, *, *, *, *]$ and $\mathbf{a}[(*, i, *, *, y_i, *, *, *, *)]$ with indices drawn from $\mathcal{U}_1$. Note that P only sends vertices with nonzero weight, but since they are sent in increasing order, V can infer the missing entries and issue updates to $\mathbf{a}$ as above. V also maintains the sum of all y_i.
3. P sends the description of the laminar family in the form of tuples $(LI, \ell, r_U, \partial U)$, sorted in lexicographic order by LI and then by ℓ. V performs the following operations.
   a. V increments all entries of the form $(i, *, *, y_i, *, 0, 0, 0, 0)$ or $(*, i, *, *, y_i, 0, 0, 0, 0)$ by 2 to account for edges which are satisfied by only vector y.
   b. V maintains the sum Σ_R of all r_U seen thus far. If the tuple is deepest level for a given claw (easily verified by retaining a one-tuple lookahead) then V adds r_U to a running sum $\Sigma_{\max}$.
   c. V verifies that the entries appear in sorted order and that r_U is monotone increasing.
   d. V updates the fingerprint structure from Theorem 1 with each vertex in ∂U.
   e. For each $v \in \partial U$, V increments (subject to our two constraints on the universe) all entries of $\mathbf{a}$ indexed by tuples of the form $(e, w_e, *, *, LI, *, \ell, *)$ and all entries indexed by tuples of the form $(e, w_e, *, *, LI, \ell, *, r_U)$, where e is any edge containing v as an endpoint.

f. V ensures all sets presented are odd by verifying that for each LI, all $|\partial U|$ except the last one are even.

4. P sends V all vertices participating in the laminar family in ascending order of vertex label. V verifies that the fingerprint constructed from this stream matches the fingerprint constructed earlier, and hence that all the claws are disjoint.

5. V runs a verification protocol for $F_5^{-1}(\mathbf{a})$ and accepts if $F_5^{-1}(\mathbf{a}) = m$, returning Σ_r and $\Sigma_{\max}$.

Define c^s as the certificate size, which is upper bounded by the matching cardinality. Then:

▶ **Theorem 7.** *Given dynamic updates to a weighted graph on n vertices with all weights bounded polynomially in n, there is a SIP with cost $(\log^2 n, (c^s + \log n) \log n)$, where c^s is the cardinality of maximum matching, that runs in $\log n$ rounds and verifies the size of a maximum weight matching.*

Proof. In parallel, V and P run protocols to verify a claimed matching as well as its optimality. The correctness and resource bounds for verifying the matching follow from Section 6. We now turn to verifying the optimality of this matching. The verifier must establish the following facts:

(i) P provides a valid laminar family of odd sets.
(ii) The lower and upper bounds are equal.
(iii) All dual constraints are satisfied.

Since the verifier fingerprints the vertices in each claw and then asks P to replay all vertices that participate in the laminar structure, it can verify that no vertex is repeated and therefore that the family is indeed laminar. Each ∂U in a claw can be written as the difference of two odd sets, except the deepest one (for which $\partial U = U$. Therefore, the cardinality of each ∂U must be even, except for the deepest one. V verifies this claim, establishing that the laminar family comprises odd sets.

Consider the term $\sum_U z_U \lfloor |U|/2 \rfloor$ in the dual cost. Since each U is odd, this can be rewritten as $(1/2)(\sum_u z_u|U| - \sum_U z_U)$. Consider the odd sets $U_0 \supset U_1 \supset \ldots \supset U_l$ in a single claw. We have $r_{U_j} = \sum_{i \leq j} z_{U_i}$, and therefore $\sum_j r_{U_j} = \sum_j \sum_{i \leq j} z_{U_i}$. Reordering, this is equal to $\sum_{i \leq j} \sum_j z_{U_i} = \sum_i z_{U_i}|U_i|$. Also, $r_{U_l} = \sum_i z_{U_i}$. Summing over all claws, $\Sigma_r = \sum_U z_U|U|$ and $\Sigma_{\max} = \sum_U z_U$. Therefore, $\sum_i y_i + \Sigma_r - \Sigma_{\max}$ equals the cost of the dual solution provided by P.

Finally we turn to validating the dual constraints. Consider an edge $e = (i, j)$ whose dual constraints are satisfied: i.e. P provides y_i, y_j and z_U such that $y_i + y_j + r_{ij} \geq w_e$. Firstly, consider the case when $r_{ij} > 0$. In this case, the edge belongs to some claw LI. Let its lower and upper endpoints vertex levels be s, t, corresponding to odd sets U_S, U_t. Consider now the entry of $\mathbf{a}$ indexed by $(e, y_i, y_j, LI, s, t, r_{ij})$. This entry is updated when e is initially encountered and ends up with a net count of 1 at the end of input processing. It is incremented twice when P sends the (i, y_i) and (j, y_j). When P sends U_s this entry is incremented because $r_{ij} = r_{U_s} = \min(r_{U_S}, r_{U_t})$ and when P sends U_t this entry is incremented because U_t has level t, returning a final count of 5. If $r_{ij} = 0$ (for example when the edge crosses a claw), then the entry indexed by $(e, w_e, y_i, y_j, 0, 0, 0, 0)$ is incremented when e is read. It is not updated when P sends (i, y_i) or (j, y_j). When P sends the laminar family, V increments this entry by 2 twice (one for each of i and j) because we know that $y_i + y_j \geq w_e$. In this case, the entry indexed by $(i, j, w_e, y_i, y_j, 0, 0, 0, 0)$ will be exactly 5. Thus, for each satisfied edge there is exactly one entry of $\mathbf{a}$ that has a count of 5.

Conversely, suppose e is not satisfied by the dual constraints, for which a necessary condition is that $y_i + y_j < w_e$. Firstly, note that any entry indexed by $(i, j, w_e, *, *, 0, 0, 0, 0)$ will receive only two increments: one from reading the edge, and another from one of y_i and y_j but not both. Secondly, consider any entry with an index of the form $(i, j, w_e, *, *, LI, *, *, *)$ for $LI > 0$. Each such entry gets a single increment from reading e and two increments when P sends (i, y_i) and (j, y_j). However, it will not receive an increment from the second of the two updates in Step 3(e), because $y_i + y_j + r_{ij} < w_e$ and so its final count will be at most 4. The complexity of the protocol follows from the complexity for Finv, Subset and the matching verification described in Section 6. ◀

8 Streaming Interactive Proofs for Approximate Metric TSP

We can apply our protocols to another interesting graph streaming problem: that of computing an approximation to the min cost travelling salesman tour. The input here is a weighted complete graph of distances. We briefly recall the Christofides heuristic: compute a MST T on the graph and add to T all edges of a min-weight perfect matching on the odd-degree vertices of T. The classical Christofides result shows that the sum of the costs of this MST and induced min-weight matching is a 3/2 approximation to the TSP cost. In the SIP setting, we have protocols for both of these problems. The difficulty however is in the dependency: the matching is built on the odd-degree vertices of the MST, and this would seem to require the verifier to maintain much more states as in the streaming setting. We show that this is not the case, and in fact we can obtain an efficient SIP for verifying a $(3/2+\epsilon)$-approximation to the TSP. To summarize our SIP for verifying an approximate MST, here first we state two main results which are used as subroutine: Connectivity and Bipartiteness.

For establishing the number of connected components in graph G, we devise SIPs to verify spanning trees, as well as the disjointness and maximality of any claimed connected components by prover. We show the results here and move full details of the protocols to the full version [1].

▶ **Lemma 8.** *Given an input graph G with n vertices, there exists a SIP protocol for verifying the number of connected components G_i with $(\log n)$ rounds of communication, and $(\log^2 n, n \log n)$ cost.*

By applying the verification protocol for connectivity on both the input graph G and the *bipartite double cover* [4] of G we obtain the following results for testing bipartiteness:

▶ **Lemma 9.** *Given an input graph G with n vertices, there exists a SIP protocol for testing bipartiteness on G with $(\log n)$ rounds of communication, and $(\log^2 n, n \log n)$ cost.*

Now for MST protocol, we follow a reduction for stratifying graph edges by weight and counting the number of connected components at each level introduced in [18] and later generalized to streaming setting [4]. This finally yields us:

▶ **Theorem 10.** *Given a weighted graph with n vertices, there is a SIP protocol for verifying MST within $(1 + \epsilon)$-approximation with $(\log n)$ rounds of communication, and $(\log^2 n, n \log^2 n/\epsilon)$ cost.*

We note here that while we could have used known parallel algorithms for connectivity and MST combined with the protocol of Goldwasser et al. [30] and the technique of Cormode, Thaler and Yi [23] to obtain similar results, we need an explicit and simpler protocol with an output that we can fit into the overall TSP protocol.

What remains is how we verify a min-cost perfect matching on the odd-degree nodes of the spanning tree. We employ the procedure described in Section 7 for maximum weight matching along with a standard equivalence to min-cost perfect matching. In addition to validating all the LP constraints, we also have to make sure that they pertain solely to vertices in ODD. We do this as above by using the fingerprint for ODD to ensure that we only count satisfied constraints on edges in ODD. We present the details of TSP in the full version of the paper [1]. Finally, the approximate TSP cost is the sum of the min-weight perfect matching on ODD and the MST cost on the graph.

▶ **Theorem 11.** *Given a weighted complete graph with n vertices, in which the edge weights satisfy the triangle inequality, there exists a streaming interactive protocol for verifying optimal TSP cost within $(\frac{3}{2}+\epsilon)$-approximation with $(\log n)$ rounds of communication, and $(\log^2 n, n\log^2 n/\varepsilon)$ cost.*

References

1 A. Abdullah, S. Daruki, C. D. Roy, and S. Venkatasubramanian. Streaming verification of graph properties. *CoRR*, abs/1602.08162, 2016. URL: http://arxiv.org/abs/1602.08162.

2 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *arXiv preprint arXiv:1307.4359*, 2013.

3 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467. SIAM, 2012.

5 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.

6 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

7 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Tight bounds for linear sketches of approximate matchings. *arXiv preprint arXiv:1505.01467*, 2015.

8 Pablo Daniel Azar and Silvio Micali. Rational proofs. In *STOC*, pages 1017–1028. ACM, 2012.

9 Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

10 Ziv Bar-Yossef. *The complexity of massive data set computations.* PhD thesis, University of California at Berkeley, 2002.

11 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

12 C. Berge. Sur le couplage maximum d'un graphe. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 247:258–259, 1958.

13 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Automata, Languages, and Programming*, pages 244–254. Springer, 2013.

14 Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the 25th ACM*

SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pages 253–262. ACM, 2006.

15 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. *arXiv preprint arXiv:1505.02019*, 2015.

16 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Annotations in data streams. In *Automata, Languages and Programming*, pages 222–234. Springer, 2009.

17 Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and arthur–merlin communication. In *CCC*, 2015.

18 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005.

19 Jing Chen, Samuel McCauley, and Shikha Singh. Rational proofs with multiple provers. *CoRR*, abs/1504.08361, 2015. URL: `http://arxiv.org/abs/1504.08361`.

20 Rajesh Chitnis, Graham Cormode, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: maximal matching and vertex cover. In *SODA*, pages 1234–1251. SIAM, 2015.

21 Graham Cormode. Bertinoro workshop 2011, problem 47. URL: `http://sublinear.info/index.php?title=Open_Problems:47`.

22 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.

23 Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proceedings of the VLDB Endowment*, 5(1):25–36, 2011.

24 Michael Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. *APPROX/RANDOM*, 28:96–104, 2014.

25 W. H. Cunningham and A. B. Marsh. A primal algorithm for optimum matching. In *Polyhedral Combinatorics*, pages 50–72. Springer, 1978.

26 Samira Daruki, Justin Thaler, and Suresh Venkatasubramanian. Streaming verification in data analysis. In *Algorithms and Computation*, pages 715–726. Springer Berlin Heidelberg, 2015.

27 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.

28 Hossein Esfandiari, Mohammad T Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *SODA*, pages 1217–1233. SIAM, 2015.

29 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:177, 2015. URL: `http://eccc.hpi-web.de/report/2015/177`.

30 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC'08*, pages 113–122, New York, NY, USA, 2008. ACM.

31 Siyao Guo, Pavel Hubácek, Alon Rosen, and Margarita Vald. Rational arguments: single round delegation with sublinear verification. In *Proc. ITCS*, pages 523–540. ACM, 2014.

32 Siyao Guo, Pavel Hubácek, Alon Rosen, and Margarita Vald. Rational sumchecks. In *Theory of Cryptography*, pages 319–351. Springer, 2016.

33 Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In Tim Roughgarden, editor, *Proc. ITCS*, pages 133–142. ACM, 2015.

34 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics*, pages 710–716. Springer, 2005.

35 Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: The power of no-signaling proofs. Cryptology ePrint Archive, Report 2013/862, 2013. URL: `http://eprint.iacr.org/`.

36 Michael Kapralov. Better bounds for matchings in the streaming model. In *SODA*, pages 1679–1697. SIAM, 2013.

37 R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, January 1986.

38 Hartmut Klauck. On arthur merlin games in communication complexity. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 189–199. IEEE, 2011.

39 Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *Automata, Languages, and Programming*, pages 919–930. Springer, 2014.

40 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proc. ITCS*, pages 367–376. ACM, 2015.

41 D. König. Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére. *Matematikai és Természettudományi Értesít*o, 34:104–119, 1916.

42 Christian Konrad. Maximum matching in turnstile streams. *arXiv preprint arXiv:1505.01460*, 2015.

43 Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

44 Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.

45 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proc. STOC*, 2016.

46 Christian Sohler. Dortmund workshop on streaming algorithms, problem 52. 2012. URL: `http://sublinear.info/index.php?title=Open_Problems:52`.

47 Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *Proc. ICALP*, 2016.

48 W. T. Tutte. The factorization of linear graphs. *The Journal of the London Mathematical Society, Ser. 1*, 22(2):107–111, 1947.

Building Clusters with Lower-Bounded Sizes

Faisal Abu-Khzam[1], Cristina Bazgan[*2], Katrin Casel[†3], and Henning Fernau[‡4]

1 **Lebanese American University, Beirut, Lebanon**
`faisal.abukhzam@lau.edu.lb`

2 **Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, Paris, France**
`bazgan@lamsade.dauphine.fr`

3 **Fachbereich 4, Informatikwissenschaften, Universität Trier, Germany**
`casel@uni-trier.de`

4 **Fachbereich 4, Informatikwissenschaften, Universität Trier, Germany**
`fernau@uni-trier.de`

Abstract

Classical clustering problems search for a partition of objects into a fixed number of clusters. In many scenarios however the number of clusters is not known or necessarily fixed. Further, clusters are sometimes only considered to be of significance if they have a certain size. We discuss clustering into sets of minimum cardinality k without a fixed number of sets and present a general model for these types of problems. This general framework allows the comparison of different measures to assess the quality of a clustering. We specifically consider nine quality-measures and classify the complexity of the resulting problems with respect to k. Further, we derive some polynomial-time solvable cases for $k = 2$ with connections to matching-type problems which, among other graph problems, then are used to compute approximations for larger values of k.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.1.2 Approximation, I.5.3 Clustering

Keywords and phrases Clustering, Approximation Algorithms, Complexity, Matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.4

1 Introduction

Clustering problems arise in different areas in very diverse forms with the only common objective of finding a partition of a given set of objects into, by some measure, similar parts. Most models consider variants of the classical k-MEANS or k-MEDIAN problem in the sense that k is a fixed given integer which determines the number of clusters one searches for. In some applications however it is not necessary to compute a partition with exactly k parts, sometimes it is not even known which number for k would be a reasonable choice. We want to discuss a clustering model which does not fix the number of clusters but instead requires that each cluster contains at least k objects. This constraint can be seen as searching for

[*] Institut Universitaire de France.

[†] K. Casel gratefully acknowledges the support by the Deutsche Forschungsgemeinschaft, grant FE 560/6-1.

[‡] H. Fernau gratefully acknowledges the support by the Deutsche Forschungsgemeinschaft, grant FE 560/6-1.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 4; pp. 4:1–4:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a clustering into parts of a specified minimum significance. For general classification or compression tasks, one might consider small clusters as disposable outliers.

One concrete scenario for this type of partitioning is Load Balanced Facility Location [11], a variant of the facility location problem where one is only interested in building profitable facilities. In this scenario a facility is not measured by the initial cost of building it but by its profitability once it is opened. Consequently, it is only reasonable to build a facility if there are enough (but maybe not too many) customers who use it but aside from this constraint it is possible to build an unrestricted number of facilities. The considered cardinality-constraint also models the basic principle of “hiding in a crowd” introduced by the concept of k-*anonymity* [14] which introduces formal problems such as r-Gather [1] and k-Member Clustering [4]. A cluster in this scenario is a collection of personal records which has to have a certain minimum cardinality in order to be considered anonymous.

We want to consider the general task of computing a clustering into sets of minimum cardinality $k \in \mathbb{N}$ with the objective to introduce an abstract framework to model such types of problems. For this purpose, we define the generic problem $(\|\cdot\|, f)$-k-cluster and specifically discuss nine variants of it, characterised via three different choices for each f and $\|\cdot\|$; a detailed description of these variants follows in Section 2. Our main contributions are the abstract model and the complexity- and approximation-results which become more apparent due to this model, as they are derived mostly via similarities to other graph problems. Section 3 compares the nine problem variants with respect to structural differences. In Section 4 and 5, we classify the complexity for small values of k by identifying polynomial-time solvable cases with connections to matching-type problems and deriving (also improving known) NP-hardness results for the remaining cases. Section 6 uses a large variety of connections to other graph problems, including the results from Section 4, to develop approximation-algorithms. A more detailed description of the results as well as the comparison to results from related work follows in the respective sections and is summarised in the conclusions.

2 General Abstract Model

In the following, we consider the general task of partitioning a set of n given objects into sets of cardinality at least k. Our model represents the n input-objects as vertices of an undirected graph $G = (V, E)$. A feasible solution is any partitioning $P_1, \dots, P_s$ of V such that $|P_i| \geq k$ for all $i \in \{1, \dots, s\}$, in the following we will refer to such a partition as *k-cluster*. Recall that in contrast to the classical clustering problems like s-means or s-median, the number of clusters s is not necessarily part of the input. Of course, one does not search for just any k-cluster but for a partitioning which preferably only combines objects which are in some sense “close”. This similarity can be very hard to capture and the appropriate way to measure it highly depends on the clustering-task and the structure of the input. We therefore consider an arbitrary given distance function $d\colon V^2 \to \mathbb{R}_+$ which for any two objects $u, v \in V$ represents the distortion which is caused by combining u and v. This general view allows to simultaneously study many different measures for dissimilarity.

In our model, the distance d is defined via a given edge-weight function $w_E\colon E \to \mathbb{R}_+$. For two vertices $u, v \in V$ we define $d(u, v) := w_E(\{u, v\})$ if $\{u, v\} \in E$, and if $\{u, v\} \notin E$, the distance $d(u, v)$ is defined by the shortest path from u to v in G. We will say that d satisfies the *triangle inequality* (and hence is a metric) if $d(u, v) \leq d(u, w) + d(w, v)$ for all $u, v, w \in V$. Observe that our definition allows for distances d which do not satisfy this property, a simple example is the complete graph over $V = \{u, v, w\}$ with $w_E(\{u, v\}) = w_E(\{u, w\}) = 1$ and

$w_E(\{v,w\}) = 3$. Distances which are defined directly via an edge are the only possible 'non-metric' distances. Edges hence do not necessarily imply similarity but can reflect a difference greater than the shortest path between two objects and make it more unattractive to cluster them together; very different from the multiedges introduced in the hypergraph-model for k-anonymous clustering from [17], where hyperedges reflect similar groups.

The overall cost of a partitioning $P_1, \dots, P_s$ is always in some sense proportional to the dissimilarities within each set or *cluster* P_i. On an abstract level, the *global cost* induced by a partitioning $P_1, \dots, P_s$ is calculated by first computing the *local cost* of each cluster and second by combining all this individual information. In this paper, we discuss three different measures for the local cost caused by a cluster P_i:

Radius: $\text{rad}(P_i) := \min_{x \in P_i} \max_{y \in P_i} d(x,y)$.
Diameter: $\text{diam}(P_i) := \max_{x \in P_i} \max_{y \in P_i} d(x,y)$.
Average Distortion: $\text{avg}(P_i) := \frac{1}{|P_i|} \cdot \min_{x \in P_i} \sum_{y \in P_i} d(x,y)$.

The overall cost of a k-cluster $P_1, \dots, P_s$ is then given by a certain combination of the local costs $f(P_1), \dots, f(P_s)$ with $f \in \{\text{rad}, \text{diam}, \text{avg}\}$. In order to model the most common problem-versions we consider the following three possibilities:

Worst Local Cost: The maximum cost of an individual cluster: $\max_{1 \leq i \leq s} f(P_i)$. Because of its structure with respect to the values $f(P_1), \dots, f(P_s)$, denoted by $\|\cdot\|_\infty$.

Worst Weighted Local Cost: The maximum cost of an individual cluster, weighted by its size: $\max_{1 \leq i \leq s} |P_i| f(P_i)$, denoted by $\|\cdot\|_\infty^w$.

Accumulated Local Cost: The sum of the distortion for each cluster, denoted by $\|\cdot\|_1^w$, with respect to the cost of the individual clusters computed by: $\sum_{i=1}^s |P_i| f(P_i)$.

Any combination of $f \in \{\text{rad}, \text{diam}, \text{avg}\}$ with $\|\cdot\| \in \{\|\cdot\|_1^w, \|\cdot\|_\infty^w, \|\cdot\|_\infty\}$ yields a different problem. (Structural properties discussed in Section 3 will explain why we do not consider the unweighted 1-norm.) For a fixed $k \in \mathbb{N}$, the general optimisation-problem is given by:

$(\|\cdot\|, f)$-k-CLUSTER
Input: Graph $G = (V,E)$ with edge-weight function $w_E : E \to \mathbb{R}_+$, $k \in \mathbb{N}$.
Output: k-cluster $P_1, \dots, P_s$ of V for some $s \in \mathbb{N}$, which minimises $\|(f(P_1), \dots, f(P_s))\|$.

$(\|\cdot\|_\infty, \text{rad})$-$k$-CLUSTER, for example, searches for a k-CLUSTER which minimises:

$$\max_{1 \leq i \leq s} \min_{x \in P_i} \max_{y \in P_i} d(x,y).$$

Some of the variants of $(\|\cdot\|, f)$-k-CLUSTER are already known under different names. The variant $(\|\cdot\|_1^w, \text{diam})$-$k$-CLUSTER is also known as k-MEMBER CLUSTERING [4] and with d chosen as the Euclidean distance, $(\|\cdot\|_\infty, \text{rad})$-$k$-CLUSTER is the so-called r-GATHER problem [1] (with $r = k$). Variant $(\|\cdot\|_1^w, \text{avg})$-$k$-CLUSTER is LOAD BALANCED FACILITY LOCATION [11] with unit demands and without facility costs and, with Euclidean distance, also models MICROAGGREGATION [6].

Choosing between the cluster-measures and norms allows adjustment for specific types of objects and different forms of output representation. The norm decides if the desired output has preferably uniformly structured clusters with or without uniform cardinalities (∞-norms) or builds clusters of object-specific irregular structure (1-norm). For cohesive clustering, the diameter-measure is more suitable for the choice of f. Average distortion is best used when the output chooses one representative of each cluster and projects all other objects in this cluster to it; a scenario which for example occurs for facility-location type problems. If the output does not project to one representative but considers clusters as circular areas, the radius measure is the most reasonable choice for f. Optimal k-clusters may differ for different choices of $\|\cdot\|$ and/or f as we will discuss in the next section. Still, we will see that there are also very useful similarities.

3 Structural Properties of Optimal Partitions

The diverse behaviour for different choices of f and $\|\cdot\|$ is nicely displayed in the cluster-cardinalities of optimal solutions. For the example $V := \{c, v_1, v_2, \dots, v_n\}$ with $w_E(c, v_i) := 1$ for all i, we find that for radius and average distortion, the single cluster V is *the* optimal solution with $\|\cdot\|_\infty$ or $\|\cdot\|_1^w$. If $w_E(v_i, v_j) := D$ for some large value D, any k-cluster with more than one set is arbitrarily worse. For the diameter-measure however we know that in general $\mathrm{diam}(S) \le \mathrm{diam}(P)$ for all sets $S \subseteq P$, which immediately yields:

▶ **Proposition 1.** *For any $k \in \mathbb{N}$ and any $\|\cdot\| \in \{\|\cdot\|_1^w, \|\cdot\|_\infty^w, \|\cdot\|_\infty\}$, optimal solutions $P_1, \dots, P_s$ for $(\|\cdot\|, diam)$-k-CLUSTER can be assumed to satisfy $|P_i| < 2k$ for all $1 \le i \le s$.*

For radius we only have the weaker property that $\mathrm{rad}(S) \le \mathrm{rad}(P)$ for all sets $S \subseteq P$ such that the center of P is contained in S. Average distortion lacks such monotone behaviour entirely. Observe that a large cardinality of a cluster can sort of "smooth over" some larger distances, for example for three vertices u, v, w with $w_E(u, v) := 3$ and $w_E(u, w) := 1$, adding w to the cluster $\{u, v\}$ decreases the average distortion from $\frac{3}{2}$ to $\frac{4}{3}$. Examples like this show that, even with triangle inequality for d, we can not in general restrict the maximum cluster-cardinality for $(\|\cdot\|_\infty, \mathrm{avg})$-$k$-CLUSTER, which is a bit unsetteling, given that most applications also like to have some natural upper bound on the cardinality (not too many customers). In a realistic scenario, we encounter sets of cardinality $2k$ or larger in optimal solutions for $(\|\cdot\|_\infty, \mathrm{avg})$-$k$-CLUSTER, if they contain an object (often called outlier) which has a large distance from all objects. Deleting such outliers before computing clusters is generally a reasonable pre-processing step, which makes large clusters in $(\|\cdot\|_\infty, \mathrm{avg})$-$k$-CLUSTER unlikely.

In general, we would like the computation of global cost to somehow favour finer partitions in order to exploit the difference to clustering models which bound the number of sets. This is the reason why we do not consider the unweighted 1-norm, formally computed by $\|(f(P_1), \dots, f(P_s))\|_1 := \sum_{i=1}^{s} f(P_i)$. For the example $V = \{v_i^1, v_i^2 \colon 1 \le i \le n\}$ with $w_E(\{v_i^1, v_i^2\}) = 1$ for $i \in \{1, \dots, n\}$ and $w_E(\{v_i^h, v_j^k\}) = n - 1$ for $i, j \in \{1, \dots, n\}$ with $i \ne j$ and $h, k \in \{1, 2\}$, the best 2-clustering w.r.t. $\|\cdot\|_1$ with any choice for f is V itself, while the most reasonable 2-clustering for most applications one can think of for this graph is obviously $\{\{v_i^1, v_i^2\} \colon 1 \le i \le n\}$. This makes $\|\cdot\|_1$ very unattractive for our clustering-purposes, observe that triangle inequality does not improve this behaviour, since the distance d for this example satisfies it. Triangle inequality however makes a big difference for the worst-case example in the beginning of the section and allows to conclude:

▶ **Theorem 2.** *If d satisfies the triangle inequality, the restriction to partitions into sets of cardinality at most $2k - 1$ yields a 2-approximation for $(\|\cdot\|_\infty, rad)$-, $(\|\cdot\|_1^w, rad)$- and $(\|\cdot\|_1^w, avg)$-k-CLUSTER and is optimal for $(\|\cdot\|_\infty^w, avg)$- and $(\|\cdot\|_\infty^w, rad)$-$k$-CLUSTER.*

As we will look at the cases $k = 2$ and $k = 3$ in the next section, we further conclude:

▶ **Corollary 3.** *If d satisfies the triangle inequality, sets in partitions for $(\|\cdot\|_1, avg)$-2-CLUSTER can be assumed to have cardinality two or three.*

Proof. For a cluster $S := \{x_1, x_2 \dots, x_r\}$ with center x_1 and $r > 3$, a further partitioning into $\{x_{2i}, x_{2i+1}\}$ for $i \in \{1, \dots, z - 1\}$ with $z = \lfloor \frac{r}{2} \rfloor$ and $\{x_1, x_{2z}, x_r\}$ does not increase the global cost for ($\|\cdot\|_1$, avg)-2-CLUSTER, since:

$$\begin{aligned}&|S|\text{avg}(S)\\ =&\sum_{i=1}^{r} d(x_i,x_r) \leq (r-2z)d(x_{2z},x_r)+d(x_r,x_{2z-1})+\sum_{i=1}^{z-1} d(x_{2i},x_r)+d(x_{2i+1},x_r)\\ \leq&|\{x_1,x_{2z},x_r\}|\text{avg}(\{x_1,x_{2z},x_r\})+\sum_{i=1}^{z-1}2\text{avg}(\{x_{2i},x_{2i+1}\})\,.\end{aligned}$$ ◀

4 Connections to Matching Problems

The graph-representation we chose to define ($\|\cdot\|, f$)-k-CLUSTER reveals relations to other well studied graph problems, interestingly in case of $k = 2$ not to classical clustering but to matching problems. Some variants can be reduced to finding a minimum weight edge cover, a problem which can be reduced to finding a minimum weight perfect matching (a simple reduction is described, e.g., in the first volume of Schrijver's monograph [[15], Section 19.3]). As a consequence, a minimum weight edge cover can be found in $O(n^3)$ time by the results of Edmonds and Johnson [8].

▶ **Theorem 4.** *($\|\cdot\|_1^w$, avg)-2-CLUSTER can be solved in $\mathcal{O}(n^3)$ time.*

Proof. ($\|\cdot\|_1^w$, avg)-2-CLUSTER searches for a 2-cluster $P_1,\dots,P_s$ minimising:

$$\sum_{i=1}^{s}\min\{\sum_{y\in P_i} d(x,y)\colon x\in P_i\}.$$

In other words, for any graph $G = (V, E)$, the global cost is the weight of the cheapest edge-set $E' \subset V^2$ for which the graph $G' := (V, E')$ has s connected components $P_1,\dots,P_s$ with at least 2 vertices such that the induced subgraph of each P_i is a star-graph. This property is equivalent to E' being a minimum weight edge cover for the complete graph on V with edge-weights equal to the distance d; observe that the graph (V, E') is a forest without isolates and without paths of length three for every minimum weight edge cover E' which means that its connected components are star-graphs. ◀

▶ **Theorem 5.** *($\|\cdot\|_\infty$, rad)-2-CLUSTER can be solved in $\mathcal{O}(n^2)$ time.*

Proof. For a graph $G = (V, E)$, first check all vertices in V and find the smallest value $c > 0$ such that each vertex v has distance at most c from at least one other vertex. This c is obviously a general lower bound on the global cost, since each vertex needs at least one partner. For $k = 2$, this c is also the optimal value since any minimal edge cover for the graph $G' := (V, E')$ with $E' := \{(u, v)\colon 0 < d(u, v) \leq c\}$ yields a 2-cluster for G with radius at most c for each cluster. ◀

With respect to diameter, this edge-cover strategy is not applicable for clusters of cardinality larger than two. Even for $k = 2$ there are cases for which clusters of cardinality three are required in every optimal solution. It seems difficult to define a correct way to compute the diameter of a cluster by summing up certain edge-weights. We therefore consider the following matching problem which is more involved but still solvable in $\mathcal{O}(n^3m^2\log n)$ [2]:

SIMPLEX MATCHING
Input: Hypergraph $H = (V, F)$ with $F \subseteq (V^2 \cup V^3)$ and cost-function $c : F \to \mathbb{R}$ satisfying:

1. $\{\{u,v\},\{v,w\},\{u,w\}\} \subset F$ for all $\{u,v,w\} \in F$. (*subset cond.*)
2. $c(\{u,v\}) + c(\{v,w\}) + c(\{u,w\}) \leq 2c(\{u,v,w\})$ for all $\{u,v,w\} \in F$. (*simplex cond.*)

Output: A perfect matching of H (that is a collection S of hyperedges such that every vertex in V appears in exactly one hyperedge of S) of minimal cost.

▶ **Corollary 6.** $(\|\cdot\|_1^w$, *diam*)-2-CLUSTER *can be solved in* $\mathcal{O}(n^9 \log n)$ *time.*

Proof. Let $G = (V, E)$ be an input graph for $(\|\cdot\|_1^w, \text{diam})$-2-CLUSTER. The corresponding input for SIMPLEX MATCHING is the hypergraph $H = (V, V^2 \cup V^3)$ which obviously satisfies the subset condition. By Proposition 1, there exists an optimal solution for $(\|\cdot\|_1^w, \text{diam})$-2-CLUSTER among the perfect matchings for H. According to the original problem, the cost-function c for any $u, v, w \in V$ is defined as: $c(\{u,v\}) := 2d(u,v)$ and $c(\{u,v,w\}) := 3\max\{d(u,v), d(v,w), d(u,w)\}$ and hence satisfies the simplex condition. Since this complete hypergraph has $\mathcal{O}(n^3)$ hyperedges, the overall running-time is in $\mathcal{O}(n^9 \log n)$. ◀

Diameter combined with the ∞-norms can be solved using Corollary 6 by fixing some maximum diameter D and multiplying all hyperedge-costs which exceed D with a large value C, say $C = n\max\{d(u,v) \colon u,v \in V\}$. This does not violate the simplex condition for the cost-function and there exists a solution for $(\|\cdot\|_\infty, \text{diam})$-2-CLUSTER of value D for the original graph if and only if the hypergraph with adjusted costs has a $(\|\cdot\|_1^w, \text{diam})$-2-CLUSTER solution of value less than C. Relating to an easier problem, we can do a little better. If we remove the hyperedges which exceed D instead of changing their cost, we arrive at a hypergraph which still satisfies the subset condition ($\text{diam}(\{u,v\}) \leq \text{diam}(\{u,v,w\})$ for any $u,v,w \in V$) and we are only interested in any perfect matching, regardless of its weight. The computation of such a perfect matching is the problem called SIMPLEX COVER [19][1]. The augmenting-path strategy from [16] for 2-GATHERING[2], can be used to solve SIMPLEX COVER in time $\mathcal{O}(m^2)$, where m is the number of hyperedges of the input graph.

▶ **Corollary 7.** $(\|\cdot\|_\infty$, *diam*)- *and* $(\|\cdot\|_\infty^w$, *diam*)-2-CLUSTER *and if d satisfies the triangle inequality also* $(\|\cdot\|_\infty^w$, *avg*)-2-CLUSTER *can be solved in* $\mathcal{O}(n^6 \log n)$ *time.*

▶ Remark. We would like to point out that SIMPLEX MATCHING is also an interesting way to solve a sort of geometric version of $(\|\cdot\|_1^w, \text{avg})$-2-CLUSTER, originally introduced as MICROAGGREGATION in [6], which considers clustering a set of vectors in $\mathbb{R}^d$ and measures local cost for a cluster $\{x_1, \ldots, x_t\}$ by $\sum_{i=1}^t ||x_i - x||_2^2$ where x is the centroid $\frac{1}{t}(x_1 + \cdots + x_t)$. With the hypergraph $(V, V^2 \cup V^3)$ with $V = \{v_1, \ldots, v_n\}$ representing $\{x_1, \ldots, x_n\}$ and the cost-function c defined by: $c(\{v_i, v_j, v_k\}) := \sum_{h \in \{i,j,k\}} ||x_h - \frac{1}{3}(x_i + x_j + x_k)||_2^2$ for all $1 \leq i < j < k \leq n$ and $c(\{v_i, v_j\}) := \frac{1}{2}||x_i - x_j||_2^2$ for all $1 \leq i < j \leq n$, the simplex condition holds, since $2c(\{v_i, v_j, v_k\}) = \frac{4}{3}(c(\{v_i, v_j\}) + c(\{v_j, v_k\}) + c(\{v_i, v_k\}))$. This construction gives a polynomial-time algorithm to solve 2-MICROAGGREGATION which improves on the 2-approximation from [7].

As powerful as SIMPLEX MATCHING may seem, the estimated worst-case running-time is fairly large. We believe that an augmenting path strategy which is specifically tailored to the above problems can yield significant improvement. Observe that similar construction for $(\|\cdot\|_1^w, \text{rad})$-2-CLUSTER does not work, since the cluster-cardinality is not bounded by three. Also, even if d satisfies the triangle inequality, the corresponding cost-function c

[1] This covering problem is equivalent to $\{K_2, K_3\}$-PACKING an old, well studied generalisation of the classical matching problem [5].

[2] Confusingly, 2-GATHERING in [16] is not equivalent to the r-GATHERING problem from [1] with $r = 2$.

would not satisfy the simplex condition, since for the small example of three vertices u, v, w with $d(u, v) = d(u, w) = 1$ and $d(v, w) = 2$, the cost with respect to radius would give $1 = c(\{u, v, w\}) < \frac{1}{2}(c(\{u, v\}) + c(\{u, w\}) + c(\{v, w\})) = 2$. Similar problems arise for the other so far unresolved variants of $(\|\cdot\|, f)$-2-CLUSTER.

5 Complexity Results

In [1], the problem r-GATHER, which is $(\|\cdot\|_\infty, \mathrm{rad})$-$k$-CLUSTER with $r = k$ with Euclidean distance, was shown to be NP-complete for $k \geq 7$. In [3] this result was strengthened by a reduction from EXACT-t-COVER to $k \geq 3$, however for a type of problem where the cluster-center exists as an input vertex but is assigned to a different cluster (i.e., with the radius of a cluster P_i calculated by: $\min_{x \in V} \max_{y \in P_i} d(x, y)$) which is not allowed in our formal definition. We establish in the following a different reduction from the EXACT-t-COVER problem which shows NP-hardness for all our variants of k-cluster and extends for all measures f which are strictly monotone with respect to radius, diameter or average distortion. With EXACT-t-COVER we refer to the problem of deciding for a given collection $C = \{S_1, \dots, S_r\}$ of subsets of a universe $X = \{x_1, \dots, x_n\}$ with $|S_i| = t$ for all i, if there exists $C' \subset C$ such that $|C'| = n/t$ and $\bigcup_{S \in C'} S = X$, which is NP-hard for all $t \geq 3$ [9].

▶ **Theorem 8.** *All variants of $(\|\cdot\|, f)$-k-CLUSTER are NP-hard for $k \geq 3$ even with the restriction to distances d which satisfy the triangle inequality.*

Proof (Sketch). We reduce from EXACT-t-COVER with $t = (k-1)^2$. Let $S_1, \dots, S_r$ be subsets of $\{x_1, \dots, x_n\}$, with $|S_i| = t$. The graph G for $(\|\cdot\|, f)$-k-CLUSTER only contains edges of weight one and vertices $u_1, \dots, u_n$ representing $x_1, \dots, x_n$ and, for all $i \in \{1, \dots, r\}$, we have vertices $w_i^1, \dots, w_i^{k-1}$ representing an arbitrary fixed partition $P_1^i, \dots, P_{k-1}^i$ of S_i with $|P_{i_j}| = k-1$ for all j, and some additional vertices v_j for sets which are not in the cover. Edges connect u_j to with w_{i_z} if $u_j \in P_z^i$. Other edges are included depending on f. We want a solution $C \subset \{S_1, \dots, S_r\}$ with $|C| = n/t$ for EXACT-t-COVER to translate to the k-sets of vertices $\{w_i^z, u_j : x_j \in P_z^i\}$ for all i with $S_i \in C$. Assigning v_j to the set $\{w_i^1, \dots, w_i^{k-1}\}$ for i with $S_i \notin C$ then partitions the remaining vertices. There is a k-clustering which only uses these types of clusters for w_i^z if and only if $S_1, \dots, S_r$ is an exact cover.

For $f = \mathrm{diam}$, we use $\ell := r - \frac{n}{t}$ vertices $v_1, \dots, v_\ell$ and turn each of the sets $\{u_1, \dots, u_n\}$ and $w_i^1, \dots, w_i^{k-1}$ for $i \in \{1, \dots, r\}$ into a clique, and connect each v_h with $h \in \{1, \dots, \ell\}$ to all w_i^z ($i \in \{1, \dots, r\}$ and $z \in \{1, \dots, k\}$). With this, there exists an exact cover for $S_1, \dots, S_r$ if and only if a there exists a k-cluster of maximum diameter one.

For $f \in \{\mathrm{rad}, \mathrm{avg}\}$, we use r vertices $v_1, \dots, v_r$ and edges (v_i, w_i^z) for $i \in \{1, \dots, r\}$ and $z \in \{1, \dots, k-1\}$ and further include vertices y_i^j for $i \in \{1, \dots, \frac{n}{t}\}$ and $j \in \{1, \dots, k-1\}$ with edges (y_1^i, y_h^i) and (y_1^i, v_j) for each $i \in \{1, \dots, \frac{n}{t}\}$, $h \in \{2, \dots, k-1\}$ and $j \in \{1, \dots, r\}$. With this construction there exists an exact cover for $S_1, \dots, S_r$ if and only if there is a clustering such that all clusters have cardinality k and radius one.

In particular, there exists an exact cover for $S_1, \dots, S_r$ if and only if there exists a k-cluster with global cost 1, k and $2n + (k-1)r + \frac{n}{k-1}$ for radius with norm $\|\cdot\|_\infty, \|\cdot\|_\infty^w$ and $\|\cdot\|_1^w$, respectively and $\frac{k-1}{k}$, $k-1$ and $2n + \frac{1}{k}(tr - n)$ for average distortion with norm $\|\cdot\|_\infty, \|\cdot\|_\infty^w$ and $\|\cdot\|_1^w$, respectively. ◀

The previous section only provided polynomial-time solvability for roughly half of the variants of $(\|\cdot\|, f)$-2-CLUSTER. We will now complete the complexity-picture for $k = 2$.

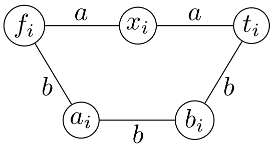
▶ **Theorem 9.** *$(\|\cdot\|_1^w, \mathit{rad})$-2-CLUSTER is APX-hard, even with the restriction to distances d which satisfy the triangle inequality.*

Proof (Sketch). We reduce from VERTEX COVER restricted to cubic graphs which is APX-hard by [13]. Let $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$ be the input for VERTEX COVER, we define $G' = (V', E')$ by $V' := \{v_i^1, v_i^2 \colon 1 \le i \le n\} \cup \{v_e \colon e \in E\}$ and $E' = \{\{v_i^1, v_i^2\} \colon 1 \le i \le n\} \cup \{\{v_i^1, v_e\} \colon v_i \in e\}$ with weights $w_E(\{v_i^1, v_i^2\}) = 1$ and $w_E(\{v_i^1, v_e\}) = 2$. With these definitions, G has a vertex cover of cardinality k if and only if there exists a solution for $(\|\cdot\|_1^w, \text{rad})$-2-CLUSTER with global cost $2n + 2k + 2m$. Since $m = 3n/2$ and $k \ge n/2$ for a cubic graph, this reduction preserves non-approximability. ◀

The reduction above can not be altered for the cases of $(\|\cdot\|, f)$-2-CLUSTER with some ∞-norm which were not shown to be polynomial-time solvable so far. We therefore consider a completely different problem for these cases to show:

▶ **Theorem 10.** *$(\|\cdot\|_\infty^w, avg)$-, $(\|\cdot\|_\infty, avg)$- and $(\|\cdot\|_\infty^w, rad)$-2-CLUSTER are all NP-hard, for the latter two even with the restriction to distances d which satisfy the triangle inequality.*

Proof (Sketch). Reduction from $(3, 3)$-SAT, i.e., satisfiability with at most three variables in each clause and where each variable occurs (positively or negatively) in at most three clauses, which remains NP-hard by [18]. Let $v_1, \dots, v_n$ be the variables and $c_1, \dots, c_m$ be the clauses. We construct G by introducing for each v_i the subgraph displayed on the below.



For each clause c_j we introduce a vertex y_j connected with edges of weight b to t_i if v_i is a literal in c_j and to f_i if $\bar{v}_i$ is a literal in c_j. With $a = \frac{1}{2}$, $b = \frac{1}{3}$ for $(\|\cdot\|_\infty, \text{rad})$-, $a = 2, b = \frac{3}{2}$ for $(\|\cdot\|_\infty, \text{avg})$- and $a = 1$, $b = \frac{1}{2}$ and also additional edges $\{y_i, y_j\}$ for all $i \ne j$ of weight one for $(\|\cdot\|_\infty^w, \text{avg})$-2-CLUSTER, the clause is satisfiable if and only if the clustering-problem has a solution of global cost one. ◀

6 Approximation results

We will only consider the case where d satisfies the triangle inequality in this section. This restriction is not just reasonable but in some sense necessary to achieve any kind of approximation. If we reconsider the reduction from Theorem 8 and turn the constructed graph G into a complete graph with additional edges of a large weight w, the difference in global cost in case of "yes"- or "no"-instance of EXACT-t-COVER increases with w, which implies:

▶ **Proposition 11.** *If d violates the triangle inequality, there is no constant-factor approximation for $(\|\cdot\|, f)$-k-CLUSTER in time polynomial in $|V|$, unless $P = NP$.*

A closer look at the metric given by the shortest paths for the original construction from Theorem 8, reveals that the global cost differs by a factor of two between "yes"- and "no"-instance for some problem-variants. Explicitly this means:

▶ **Proposition 12.** *There is no $(2 - \varepsilon)$-approximation in polynomial time for $(\|\cdot\|, f)$-k-CLUSTER with $f \in \{$rad, diam$\}$ and $\|\cdot\| \in \{\|\cdot\|_\infty, \|\cdot\|_\infty^w\}$ for any $\varepsilon > 0$ unless $P = NP$, even if d satisfies triangle inequality.*

Known approximation results for clustering with size constraints include a 9-approximation from [3] for Load Balanced Facility Location without facility cost, which is related to $(\|\cdot\|_1^w, \mathrm{avg})$-$k$-cluster here, but with the additional constraint that at each customer should be assigned to the nearest open facility. The techniques used for this result highly rely on the additional constraint, which unfortunately means that they can not be applied here. Other approximations for this problem instead relax the constraint that each cluster needs to contain at least k vertices; [11] for example presents a $2k$-approximation which constructs clusters of cardinality at least $k/3$. We will see that for our problem such an approximation factor can be achieved without relaxing the cardinality constraints. In general, our results however do not extend to Load Balanced Facility Location, since the addition of facility-costs yields a very different type of problem; we especially lose the upper bound of $2k-1$ on the cardinality of clusters in an optimal solution from Theorem 2.

Other known approximation results however also apply here and can even be altered to yield results for other problem-variants. The problem $(\|\cdot\|_\infty, \mathrm{rad})$-$k$-cluster is discussed under the name r-gather in [1], where r takes the role of k. The concept for the 2-approximation presented there can be altered, even simplified, and also used to compute a 2-approximation for $(\|\cdot\|_\infty, \mathrm{diam})$-$k$-cluster.

▶ **Theorem 13.** *$(\|\cdot\|_\infty, rad)$- and $(\|\cdot\|_\infty, diam)$-k-cluster are 2-approximable for all $k \geq 2$.*

Proof (Sketch). We try all values D that occur as pairwise distances $d(u,v)$ for $u,v \in V$ for the following greedy strategy: Start with $V_1 := V$ and iteratively, until $V_i = \emptyset$, choose $c_i \in V_i$, build clusters $P(c_i) := \{v \in V_i \colon d(c_i,v) \leq D\}$ and set $V_{i+1} = V_i \backslash P(c_i)$. This yields a partition of V into a finite number of clusters $P(c_i)$. If some cluster $P(c_i)$ has less than k vertices, consider $S(i,j) = \{v \in P(c_j) \setminus \{c_j\} \colon d(v,c_i) \leq D\}$ and move $\min\{|S(i,j)|, |P(c_j)| - k\}$ vertices from $S(i,j)$ to $P(c_i)$ for each $j \in \{1,\ldots,i-1\}$ until $|P(c_i)| \geq k$. If this procedure is successful, we arrive at a k-cluster for V with maximum radius D and maximum diameter $2D$. This procedure is successful for $D = 2r^*$ and $D = D^*$ if r^* and D^* are optimal values for $(\|\cdot\|_\infty, \mathrm{rad})$- and $(\|\cdot\|_\infty, \mathrm{diam})$-$k$-cluster respectively. ◀

▶ Remark. A greedy procedure for $(\|\cdot\|_\infty, \mathrm{avg})$-$k$-cluster could build up the sets $P(c_i)$ by successively adding $\mathrm{argmin}\{d(v,c_i) \colon v \in V_i \setminus P(c_i)\}$ until $\mathrm{avg}(P(c_i))$ exceeds D but moving vertices from $S(i,j)$ to $P(c_i)$ could unfortunately increase the average distortion of $P(c_j)$.

In [12] results from [10] for the so-called Proper Constraint Forest Problem are used to compute an $8(k-1)$-approximation for Microaggregation. We will use a different result from [10]: a 2-approximation for Lower Capacitated Tree Partitioning with capacity k which is the problem of computing a spanning forest of minimal cost for which each connected component has cardinality at least k. A spanning forest is characterised by a set of edges and its cost is defined as the sum of the weights of these edges.

▶ **Corollary 14.** *$(\|\cdot\|_1^w, avg)$-k-cluster is $2k$-approximable for all $k \geq 2$.*

▶ Remark. For $k = 2$, Theorem 4 showed that $(\|\cdot\|_1^w, \mathrm{avg})$-$k$-cluster can be solved in polynomial time which also translates to Lower Capacitated Tree Partitioning with capacity $k = 2$; tree partitioning with capacity two is equivalent to weighted edge-cover.

Essential for the result above is the fact that components of a minimal spanning forest do not contain paths of length $2k$ or more. This property implies the existence of a central vertex which can reach all vertices in its component in at most k steps and allows to bound the average distortion. This property does not prevent a component from containing arbitrarily many vertices. An algorithm for $(\|\cdot\|_1^w, \mathrm{diam})$- or $(\|\cdot\|_1^w, \mathrm{rad})$-$k$-cluster requires such an upper bound on the cardinality to prove an approximation factor. We therefore consider Lower Capacitated Path Partitioning, the restriction of Lower Capacitated Tree Partitioning to paths as connected components. With triangle inequality, [10] provides a 4-approximation for this problem and it is clear that minimal solutions can be assumed to have connected components with at most $2k-1$ vertices each, which yields:

▶ **Corollary 15.** *$(\|\cdot\|_1^w, diam)$-k-cluster is $(8k-7)$-approximable for all $k \geq 2$.*

One advantage of the unified model for $(\|\cdot\|, f)$-k-cluster is that if d satisfies the triangle inequality, the different measures relate in the following way:

$$\mathrm{avg}(P_i) \leq \mathrm{rad}(P_i) \leq \mathrm{diam}(P_i) \leq 2\mathrm{rad}(P_i)\,. \tag{1}$$

This relation with Corollary 15 immediately yields:

▶ **Proposition 16.** *$(\|\cdot\|_1^w, rad)$-k-cluster is $(16k-14)$-approximable for all $k \geq 2$.*

By definition, the two ∞-norms also relate optimal values in the following way for every choice of $f \in \{\mathrm{rad,diam,avg}\}$, where we denote by $\mathrm{opt}(G, d, \|\cdot\|, f, k)$ the global cost of an optimal solution for $(\|\cdot\|, f)$-k-cluster on G with distance d:

$$\mathrm{opt}(G, d, f, \|\cdot\|_\infty^w, k) \geq k \cdot \mathrm{opt}(G, d, f, \|\cdot\|_\infty, k)\,. \tag{2}$$

This equation is helpful to derive approximations for the weighted ∞-norm:

▶ **Corollary 17.** *$(\|\cdot\|_\infty^w, diam)$-$k$-cluster is 4-approximable and $(\|\cdot\|_\infty^w, rad)$-$k$-cluster is 8-approximable for all $k \geq 2$.*

For $(\|\cdot\|_\infty^w, \mathrm{avg})$-$k$-cluster we do not have a result for $(\|\cdot\|_\infty, \mathrm{avg})$-$k$-cluster to transfer. Interestingly, a variant with different norm and measure can be used instead:

▶ **Corollary 18.** *$(\|\cdot\|_\infty^w, avg)$-$k$-cluster is $(4k-2)$-approximable for all $k \geq 2$.*

Proof. We first show that $\mathrm{opt}(G, d, \mathrm{avg}, \|\cdot\|_\infty^w, k) \geq \mathrm{opt}(G, d, \mathrm{diam}, \|\cdot\|_\infty, k)$. Consider any set P in an optimal solution for $(\|\cdot\|_\infty, \mathrm{avg})$-$k$-cluster. Triangle inequality yields:

$$|P|\mathrm{avg}(P) = \min_{c\in P}\sum_{p\in P} d(c,p) \geq \min_{c\in P}\max_{u,v\in P} d(u,c) + d(v,c) \geq \max_{u,v\in P} d(u,v) = \mathrm{diam}(P)\,.$$

Theorem 13 and Proposition 1 produce a 2-approximation for $(\|\cdot\|_\infty, \mathrm{diam})$-$k$-cluster for which each set contains at most $2k-1$ vertices. The weighted ∞-norm of the average distortion of this partition is at most $2(2k-1)\cdot\mathrm{opt}(G, d, \mathrm{diam}, \|\cdot\|_\infty, k)$, and hence yields a $(4k-2)$-approximation for $(\|\cdot\|_\infty^w, \mathrm{avg})$-$k$-cluster. ◀

At last, we want to present an approximation which exploits the unified model in an even more surprising way. The solutions for $k=2$ derived in Section 4 for two different problem-variants are combined to compute an approximate solution for $k=4$. Explicitly, we will combine the Simplex Matching approach for $(\|\cdot\|_1^w, \mathrm{diam})$-2-cluster and the Edge Cover approach for $(\|\cdot\|_1, \mathrm{avg})$-2-cluster.

▶ **Theorem 19.** *The problem $(\|\cdot\|_1^w, diam)$-4-cluster can be approximated in polynomial time within a factor of $\frac{35}{6}$.*

Proof (Sketch). Consider as input any graph $G = (V, E)$ with induced distances d. First, compute an optimal solution $P_1, \dots, P_s$ for $(\|\cdot\|_1^w, \mathrm{diam})$-2-CLUSTER, for which the optimal value $\|(\mathrm{diam}((P_1), \dots, \mathrm{diam}(P_s))\|_1^w$ is at most $D^* := \mathrm{opt}(G, d, \mathrm{diam}, \|\cdot\|_1^w, 4)$, simply because any 4-cluster is also a 2-cluster. Next, consider the complete graph $G' = (P, P^2)$ with vertices $P := \{p_1, \dots, p_s\}$ and edge-weights w defined by $w(p_i, p_j) := \min\{d(u, v) \colon u \in P_i, v \in P_j\}$. It can be shown that $D^* \geq 3 \cdot \mathrm{opt}(G', w, \mathrm{avg}, \|\cdot\|_1^w, 2)$ and use an optimal solution $S_1, \dots, S_q$ for $(\|\cdot\|_1^w, \mathrm{avg})$-2-CLUSTER on G', such that $|S_i| \leq 3$ for all i by Corollary 3. The partition $S = \{\bigcup_{p_i \in S_j} P_i \colon 1 \leq j \leq q\}$ is a 4-cluster for G. If $S_q = \{p_i, p_j, p_k\}$ with center p_i for some $i, j, k \in \{1, \dots, s\}$ with $|P_j| = 3$, we replace the cluster $P = P_i \cup P_j \cup P_k$ in S by the two clusters $P' := P_j \cup \{u_i\}$ and $P'' := P \setminus P'$, where we choose $u_i \in P_i$ such that $w(p_i, p_j) = \min\{d(u_i, v) \colon v \in P_j\}$. These new clusters satisfy:

$$|P'|\mathrm{diam}(P') \leq 4(\mathrm{diam}(P_j) + w(p_i, p_j)) < 2|P_j|\mathrm{diam}(P_j) + 4w(p_i, p_j) \quad \text{and}$$

$$|P''|\mathrm{diam}(P'') \leq \tfrac{5}{2}|P_i|\mathrm{diam}(P_i) + \tfrac{5}{2}|P_k|\mathrm{diam}(P_k) + 5w(p_i, p_k)$$

Consider any set $R \in S$ which is not the result of splitting up a cluster. Worst case is $R = P_i \cup P_j \cup P_k$ with p_i as center of $S_q = \{p_i, p_j, p_k\}$, we know that $|R| \leq 7$ and $\mathrm{diam}(R) \leq \mathrm{diam}(P_i) + \mathrm{diam}(P_j) + \mathrm{diam}(P_k) + w(p_i, p_j) + w(p_i, p_k)$, hence:

$$|R|\mathrm{diam}(R) \leq \tfrac{7}{2}(\||P_i|\mathrm{diam}(P_i) + |P_j|\mathrm{diam}(P_j) + |P_k|\mathrm{diam}(P_k)) + 7(w(p_i, p_j) + w(p_i, p_k))\,.$$

Overall, this yields:

$$\sum_{R \in S} |R|\mathrm{diam}(R) \leq \tfrac{7}{2} \sum_{i=1}^{r} |P_i|\mathrm{diam}(P_i) + 6 \sum_{R \subset P_i \cup P_j} w(p_i, p_j) + 7 \sum_{R = P_i \cup P_j \cup P_k} w(p_i, p_j) + w(p_i, p_k)$$

$$\leq \tfrac{7}{2} \|(\mathrm{diam}((P_1), \dots, \mathrm{diam}(P_s))\|_1^w + 7 \sum_{i=1}^{q} |S_i|\mathrm{avg}(S_i) \leq \tfrac{7}{2} D^* + \tfrac{7}{3} D^* = \tfrac{35}{6} D^*\,. \qquad \blacktriangleleft$$

▶ Remark. Equation 1 translates the above result to a $\frac{35}{3}$-approximation for $(\|\cdot\|_1^w, \mathrm{rad})$-4-CLUSTER. Since the approximation-ratios from Theorem 19 are significantly better than the path-partitioning approximation from Corollary 15 (factor 25 and 50 respectively), it would be interesting to nest this construction further and extend it for larger values of k.

7 Conclusions

We have introduced and discussed the general problem $(\|\cdot\|, f)$-k-CLUSTER in order to model clustering-tasks which do not fix the number of clusters but require each cluster to contain at least k objects. The nine chosen problem-variants in this paper generalise many previous models but, of course, do not capture every possible way to measure the quality of the clustering. We however tried to cover many previous models while maintaining a clear framework in which similarities turned out to be quite fruitful.

Our NP-hardness result for $k = 3$ for all variants of $(\|\cdot\|, f)$-k-CLUSTER generalises all known complexity-results for these types of problems. Further, we completely characterise the complexity with respect to k with the following results for $(\|\cdot\|, f)$-2-CLUSTER:

$k = 2$	rad	diam	avg
$\|\cdot\|_\infty$	in P (EDGE COVER) Th.5	in P (SIMPLEX COVER) Cor.7	NP-complete Th.10
$\|\cdot\|_\infty^w$	NP-complete Th.10	in P (SIMPLEX COVER) Cor.7	NP-complete Th.10
$\|\cdot\|_1^w$	APX-hard Th. 9	in P (SIMPLEX MATCHING) Cor.6	in P (WEIGHTED EDGE COVER) Th.4

The restriction to distances d which satisfy the triangle inequality already simplified exact solvability for the general NP-hard problem $(\|\cdot\|_\infty^w, \text{avg})$-2-CLUSTER which turned out to be solvable with SIMPLEX COVER in this case. We further showed that this restriction is necessary for approximations in time polynomial in the number of objects and derived a number of approximation strategies, mostly based on different other graph-problems. Our approximation-ratios (which are the best and/or only ones known) are:

	rad	diam	avg
$\|\cdot\|_\infty$	2 Th.13	2 Th.13	?
$\|\cdot\|_\infty^w$	8 Cor.17	4 Cor.17	$4k-2$ Cor.18
$\|\cdot\|_1^w$	$16k-14$ Prop.16	$8k-7$ Cor.15	$2k$ Cor. 14

An interesting open question is whether $(\|\cdot\|_\infty, \text{avg})$-$k$-CLUSTER can be approximated within some constant ratio or at least within some ratio in $\mathcal{O}(k)$. The lack of monotonicity for average distortion makes this measure the most challenging for approximation.

References

1 G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Transactions on Algorithms*, 6(3), 2010.

2 E. Anshelevich and A. Karagiozova. Terminal Backup, 3D Matching, and Covering Cubic Graphs. *SIAM J. Comput.*, 40(3):678–708, 2011.

3 A. Armon. On min-max r-gatherings. *Theoretical Computer Science*, 412(7):573–582, 2011.

4 J.-W. Byun, A. Kamra, E. Bertino, and N. Li. Efficient k-anonymization using clustering techniques. In R. Kotagiri, P. R. Krishna, M. Mohania, and E. Nantajeewarawat, editors, *Advances in Databases: Concepts, Systems and Applications*, volume 4443 of *LNCS*, pages 188–200. Springer, 2007.

5 G. Cornuéjols, D. Hartvigsen, and W. Pulleyblank. Packing subgraphs in a graph. *Operations Research Letters*, 1(4):139–143, 1982.

6 J. Domingo-Ferrer and J. M. Mateo-Sanz. Practical Data-Oriented Microaggregation for Statistical Disclosure Control. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):189–201, 2002.

7 J. Domingo-Ferrer and F. Sebé. Optimal Multivariate 2-Microaggregation for Microdata Protection: A 2-Approximation. In J. Domingo-Ferrer and L. Franconi, editors, *Privacy in Statistical Databases, PSD'06*, volume 4302 of *LNCS*, pages 129–138. Springer, 2006.

8 J. Edmonds and E. L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.

9 F. Ergün, R. Kumar, and R. Rubinfeld. Fast approximate pcps. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 41–50, 1999.

10 M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

11 S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *In Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science, FOCS'00*, pages 603–612. IEEE Computer Society, 2000.

12 M. Laszlo and S. Mukherjee. Approximation Bounds for Minimum Information Loss Microaggregation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1643–1647, 2009.

13 C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

14 P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, November 2001.

15 A. Schrijver. *Combinatorial Optimization.* Springer, 2003.

16 A. Shalita and U. Zwick. Efficient algorithms for the 2-gathering problem. *ACM Transactions on Algorithms*, 6(2), 2010.

17 K. Stokes. On computational anonymity. In *Privacy in Statistical Databases – UNESCO Chair in Data Privacy, International Conference, PSD 2012, Palermo, Italy, September 26-28, 2012. Proceedings*, pages 336–347, 2012.

18 C. Tovey. A Simplified NP-complete Satisfiability Problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

19 D. Xu, E. Anshelevich, and M. Chiang. On survivable access network design: Complexity and algorithms. In *INFOCOM 2008. 27th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 13-18 April 2008, Phoenix, AZ, USA*, pages 186–190, 2008.

Simultaneous Feedback Edge Set: A Parameterized Perspective*

Akanksha Agrawal[1], Fahad Panolan[2], Saket Saurabh[3], and Meirav Zehavi[4]

1 **Department of Informatics, University of Bergen, Norway**
akanksha.agrawal@uib.no

2 **Department of Informatics, University of Bergen, Norway**
fahad.panolan@uib.no

3 **Department of Informatics, University of Bergen, Norway; and The Institute of Mathematical Sciences, HBNI, Chennai, India**
saket@imsc.res.in

4 **Department of Informatics, University of Bergen, Norway**
zehavimeirav@gmail.com

Abstract

In a recent article Agrawal et al. (STACS 2016) studied a simultaneous variant of the classic Feedback Vertex Set problem, called Simultaneous Feedback Vertex Set (Sim-FVS). In this problem the input is an n-vertex graph G, an integer k and a coloring function $\mathsf{col} : E(G) \rightarrow 2^{[\alpha]}$, and the objective is to check whether there exists a vertex subset S of cardinality at most k in G such that for all $i \in [\alpha]$, $G_i - S$ is acyclic. Here, $G_i = (V(G), \{e \in E(G) \mid i \in \mathsf{col}(e)\})$ and $[\alpha] = \{1, \ldots, \alpha\}$. In this paper we consider the edge variant of the problem, namely, Simultaneous Feedback Edge Set (Sim-FES). In this problem, the input is same as the input of Sim-FVS and the objective is to check whether there is an edge subset S of cardinality at most k in G such that for all $i \in [\alpha]$, $G_i - S$ is acyclic. Unlike the vertex variant of the problem, when $\alpha = 1$, the problem is equivalent to finding a maximal spanning forest and hence it is polynomial time solvable. We show that for $\alpha = 3$ Sim-FES is NP-hard by giving a reduction from Vertex Cover on cubic-graphs. The same reduction shows that the problem does not admit an algorithm of running time $\mathcal{O}(2^{o(k)}n^{\mathcal{O}(1)})$ unless ETH fails. This hardness result is complimented by an FPT algorithm for Sim-FES running in time $\mathcal{O}(2^{\omega k\alpha + \alpha \log k}n^{\mathcal{O}(1)})$, where ω is the exponent in the running time of matrix multiplication. The same algorithm gives a polynomial time algorithm for the case when $\alpha = 2$. We also give a kernel for Sim-FES with $(k\alpha)^{\mathcal{O}(\alpha)}$ vertices. Finally, we consider the problem Maximum Simultaneous Acyclic Subgraph. Here, the input is a graph G, an integer q and, a coloring function $\mathsf{col} : E(G) \rightarrow 2^{[\alpha]}$. The question is whether there is a edge subset F of cardinality at least q in G such that for all $i \in [\alpha]$, $G[F_i]$ is acyclic. Here, $F_i = \{e \in F \mid i \in \mathsf{col}(e)\}$. We give an FPT algorithm for Maximum Simultaneous Acyclic Subgraph running in time $\mathcal{O}(2^{\omega q\alpha}n^{\mathcal{O}(1)})$. All our algorithms are based on parameterized version of the Matroid Parity problem.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases parameterized complexity, feedback edge set, α-matroid parity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.5

* The research leading to these results has received funding from the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 5; pp. 5:1–5:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Deleting at most k vertices or edges from a given graph G, so that the resulting graph belongs to a particular family of graphs ($\mathcal{F}$), is an important research direction in the fields of graph algorithms and parameterized complexity. For a family of graphs $\mathcal{F}$, given a graph G and an integer k, the $\mathcal{F}$-DELETION (EDGE $\mathcal{F}$-DELETION) problem asks whether we can delete at most k vertices (edges) in G so that the resulting graph belongs to $\mathcal{F}$. The $\mathcal{F}$-DELETION (EDGE $\mathcal{F}$-DELETION) problems generalize many of the NP-hard problems like VERTEX COVER, FEEDBACK VERTEX SET, ODD CYCLE TRANSVERSAL, EDGE BIPARTIZATION, etc. Inspired by applications, Cai and Ye introduced variants of $\mathcal{F}$-DELETION (EDGE $\mathcal{F}$-DELETION) problems on edge colored graph [7]. Edge colored graphs are studied in graph theory with respect to various problems like MONOCHROMATIC AND HETEROCHROMATIC SUBGRAPHS [15], ALTERNATING PATHS [6, 8, 20], Homomorphism in edge-colored graphs [3], GRAPH PARTITIONING in 2-edge colored graphs [5] etc. One of the natural generalization to the classic $\mathcal{F}$-DELETION (EDGE $\mathcal{F}$-DELETION) problems on edge colored graphs is the following. Given a graph G with a coloring function $\mathsf{col} : E(G) \to 2^{[\alpha]}$, and an integer k, we want to delete a set S of at most k edges/vertices in G so that for each $i \in [\alpha]$, $G_i - S$ belongs to $\mathcal{F}$. Here, G_i is the graph with vertex set $V(G)$ and edge set as $\{e \in E(G) \mid i \in \mathsf{col}(e)\}$. These problems are also called *simultaneous* variant of $\mathcal{F}$-DELETION (EDGE $\mathcal{F}$-DELETION).

Cai and Ye studied the DUALLY CONNECTED INDUCED SUBGRAPH and DUAL SEPARATOR on 2-edge colored graphs [7]. Agrawal et al. [1] studied a simultaneous variant of FEEDBACK VERTEX SET problem, called SIMULTANEOUS FEEDBACK VERTEX SET, in the realm of parameterized complexity. Here, the input is a graph G, an integer k, and a coloring function $\mathsf{col} : E(G) \to 2^{[\alpha]}$ and the objective is to check whether there is a set S of at most k vertices in G such that for all $i \in [\alpha]$, $G_i - S$ is acyclic. Here, $G_i = (V(G), \{e \in E(G) \mid i \in \mathsf{col}(e)\})$. In this paper we consider the edge variant of the problem, namely, SIMULTANEOUS FEEDBACK EDGE SET, in the realm of parameterized complexity.

In the Parameterized Complexity paradigm the main objective is to design an algorithm with running time $f(\mu) \cdot n^{\mathcal{O}(1)}$, where μ is the parameter associated with the input, n is the size of the input and $f(\cdot)$ is some computable function whose value depends only on μ. A problem which admits such an algorithm is said to be *fixed parameter tractable* parameterized by μ. Typically, for edge/vertex deletion problems one of the natural parameter that is associated with the input is the size of the solution we are looking for. Another objective in parameterized complexity is to design polynomial time pre-processing routines that reduces the size of the input as much as possible. The notion of such a pre-processing routine is captured by *kernelization* algorithms. The kernelization algorithm for a parameterized problem Q takes as input an instance (I, k) of Q, runs in polynomial time and returns an equivalent instance (I', k') of Q. Moreover, the size of the instance (I', k') returned by the kernelization algorithm is bounded by $g(k)$, where $g(\cdot)$ is some computable function whose value depends only on k. If $g(\cdot)$ is polynomial in k, then the problem Q is said to admit a polynomial kernel. The instance returned by the kernelization is referred to as a *kernel* or a reduced instance. We refer the readers to the recent book of Cygan et al. [9] for a more detailed overview of parameterized complexity and kernelization.

A *feedback edge set* in a graph G is $S \subseteq E(G)$ such that $G - S$ is a forest. For a graph G with a coloring function $\mathsf{col} : E(G) \to 2^{[\alpha]}$, *simultaneous feedback edge set* is a subset $S \subseteq E(G)$ such that $G_i - S$ is a forest for all $i \in [\alpha]$. Here, $G_i = (V(G), E_i)$, where $E_i = \{e \in E(G) \mid i \in \mathsf{col}(e)\}$. Formally, the problem is stated below.

Simultaneous Feedback Edge Set (Sim-FES) **Parameter:** k, α
Input: An n-vertex graph G, an integer k and a coloring function $\mathsf{col} : E(G) \rightarrow 2^{[\alpha]}$
Question: Is there a simultaneous feedback edge set of cardinality at most k in G

Feedback Vertex Set (FVS) is one of the classic NP-complete [13] problems and has been extensively studied from all the algorithmic paradigms that are meant for coping with NP-hardness, such as approximation algorithms, parameterized complexity and moderately exponential time algorithms. The problem admits a factor 2-approximation algorithm [4], an exact algorithm with running time $\mathcal{O}(1.7217^n n^{\mathcal{O}(1)})$ [12], a deterministic parameterized algorithm running in $\mathcal{O}(3.619^k n^{\mathcal{O}(1)})$ time [16], a randomized algorithm running in $\mathcal{O}(3^k n^{\mathcal{O}(1)})$ time [10], and a kernel with $\mathcal{O}(k^2)$ vertices [24]. Agrawal et al. [1] studied Simultaneous Feedback Vertex Set (Sim-FVS) and gave an FPT algorithm running in time $2^{\mathcal{O}(\alpha k)} n^{\mathcal{O}(1)}$ and a kernel of size $\mathcal{O}(\alpha k^{3(\alpha+1)})$. Finally, unlike the FVS problem, Sim-FES is polynomial time solvable when $\alpha = 1$, because it is equivalent to finding maximal spanning forest.

Our results and approach. In Section 3 we design an FPT algorithm for Sim-FES by reducing to α-Matroid Parity on direct sum of elongated co-graphic matroids of G_i, $i \in [\alpha]$ (see Section 2 for definitions related to matroids). This algorithm runs in time $\mathcal{O}(2^{\omega k \alpha + \alpha \log k} n^{\mathcal{O}(1)})$. Unlike the vertex counterpart, we show that for $\alpha = 2$ (2-edge colored graphs) Sim-FES is polynomial time solvable. This follows from the polynomial time algorithm for the Matroid parity problem. In Section 4 we show that for $\alpha = 3$, Sim-FES is NP-hard. Towards this, we give a reduction from the Vertex Cover in cubic graphs which is known to be NP-hard [22]. Furthermore, the same reduction shows that the problem cannot be solved in $2^{o(k)} n^{\mathcal{O}(1)}$ time unless Exponential Time Hypothesis (ETH) fails [14]. We complement our FPT algorithms by showing that Sim-FES is W[1]-hard when parameterized by the solution size k (Section 5). When $\alpha = \mathcal{O}(|V(G)|)$, we give a parameter preserving reduction from the Hitting Set problem, a well known W[2]-hard problem parameterized by the solution size [9]. However, Sim-FES remains W[1]-hard even when $\alpha = \mathcal{O}(\log(|V(G)|))$. We show this by giving a parameter preserving reduction from Partitioned Hitting Set problem, a variant of the Hitting set problem, defined in [1]. In [1], Partitioned Hitting Set was shown to be W[1]-hard parameterized by the solution size. In Section 6 we give a kernel with $\mathcal{O}((k\alpha)^{\mathcal{O}(\alpha)})$ vertices. Towards this we apply some of the standard preprocessing rules for obtaining kernel for Feedback Vertex Set and use the approach similar to the one developed for designing kernelization algorithm for Sim-FVS [1]. In Section 7 we give an FPT algorithm for the problem, when parameterized by the dual parameter. Formally, this problem is defined as follows.

Maximum Simultaneous Acyclic Subgraph (Max-Sim-Subgraph) **Parameter:** q
Input: An n-vertex graph G, a positive integer q and a function $\mathsf{col} : E(G) \rightarrow 2^{[\alpha]}$.
Question: Is there a subset $F \subseteq E(G)$ such that $|F| \geq q$ and for all $i \in [\alpha]$, $G[F \cap E(G_i)]$ is acyclic?

For solving Max-Sim-Subgraph we reduce it to an equivalent instance of the α-Matroid Parity problem. As an immediate corollary we get an exact algorithm for Sim-FES running in time $\mathcal{O}(2^{\omega n \alpha^2} n^{\mathcal{O}(1)})$.

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N}$. For $n \in \mathbb{N}$, by $[n]$ we denote the set $\{1, \ldots, n\}$. For a set X, by 2^X we denote the set of all subsets of X. We use the term *ground set/ universe* to distinguish a set from its subsets. We will use ω to denote the exponent in the running time of matrix multiplication, the current best known bound for which is $\omega < 2.373$ [25].

Graphs. We use the term *graph* to denote undirected graph. For a graph G, by $V(G)$ and $E(G)$ we denote its vertex set and edge set, respectively. We will be considering finite graphs possibly having loops and multi-edges. In the following, let G be a graph and let H be a subgraph of G. By $d_H(v)$, we denote the degree of the vertex v in H, i.e, the number of edges in H which are incident with v. A self-loop at a vertex v contributes 2 to the degree of v. For any non-empty subset $W \subseteq V(G)$, the subgraphs of G induced by W, $V(G) \setminus W$ are denoted by $G[W]$ and $G - W$ respectively. Similarly, for $F \subseteq E(G)$, the subgraph of G induced by F is denoted by $G[F]$; its vertex set is $V(G)$ and its edge set is F. For $F \subseteq E(G)$, by $G - F$ we denote the graph obtained by deleting the edges in F. We use the convention that a double edge and a self-loop is a cycle. An α-edge colored graph is a graph G with a color function $\mathsf{col} : E(G) \rightarrow 2^{[\alpha]}$. By G_i we will denote the color i (or i-color) graph of G, where $V(G_i) = V(G)$ and $E(G_i) = \{e \in E(G) | i \in \mathsf{col}(e)\}$. For an α-edge colored graph G, the *total degree* of a vertex v is $\sum_{i=1}^{\alpha} d_{G_i}(v)$. We refer the reader to [11] for details on standard graph theoretic notations and terminologies.

Matroids and Representable Matroids. A pair $M = (E, \mathcal{I})$, where E is a ground set and $\mathcal{I}$ is a family of subsets (called independent sets) of E, is a *matroid* if it satisfies the following conditions: (I1) $\phi \in \mathcal{I}$. (I2) If $A' \subseteq A$ and $A \in \mathcal{I}$ then $A' \in \mathcal{I}$. (I3) If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there is $e \in (B \setminus A)$ such that $A \cup \{e\} \in \mathcal{I}$. We refer the reader to [23] for more details.

Let A be a matrix over an arbitrary field $\mathbb{F}$ and let E be the set of columns of A. For A, we define matroid $M = (E, \mathcal{I})$ as follows. A set $X \subseteq E$ is independent (that is $X \in \mathcal{I}$) if the corresponding columns are linearly independent over $\mathbb{F}$. The matroids that can be defined by such a construction are called *linear matroids*, and if a matroid can be defined by a matrix A over a field $\mathbb{F}$, then we say that the matroid is representable over $\mathbb{F}$. A matroid $M = (E, \mathcal{I})$ is called *representable* or *linear* if it is representable over some field $\mathbb{F}$.

Direct Sum of Matroids. Let $M_1 = (E_1, \mathcal{I}_1)$, $M_2 = (E_2, \mathcal{I}_2)$, $\ldots$, $M_t = (E_t, \mathcal{I}_t)$ be t matroids with $E_i \cap E_j = \emptyset$ for all $1 \leq i \neq j \leq t$. The direct sum $M_1 \oplus \cdots \oplus M_t$ is a matroid $M = (E, \mathcal{I})$ with $E := \bigcup_{i=1}^{t} E_i$ and $X \subseteq E$ is independent if and only if $X \cap E_i \in \mathcal{I}_i$ for all $i \in [t]$. Let A_i be the representation matrix of $M_i = (E_i, \mathcal{I}_i)$ over field $\mathbb{F}$. Then, a representation matrix of $M_1 \oplus \cdots \oplus M_t$ over $\mathbb{F}$ can be found in polynomial time [21, 23].

Uniform Matroid. A pair $M = (E, \mathcal{I})$ over an n-element ground set E, is called a uniform matroid if the family of independent sets is given by $\mathcal{I} = \{A \subseteq E \mid |A| \leq k\}$, where k is some constant. This matroid is also denoted as $U_{n,k}$.

▶ **Proposition 2.1** ([9, 23]). *Uniform matroid $U_{n,k}$ is representable over any field of size strictly more than n and such a representation can be found in time polynomial in n.*

Graphic and Cographic Matroid. Given a graph G, the graphic matroid $M = (E, \mathcal{I})$ is defined by taking the edge set $E(G)$ as universe and $F \subseteq E(G)$ is in $\mathcal{I}$ if and only if $G[F]$ is a forest. Let G be a graph and η be the number of components in G. The co-graphic matroid $M = (E, \mathcal{I})$ of G is defined by taking the the edge set $E(G)$ as universe and $F \subseteq E(G)$ is in $\mathcal{I}$ if and only if the number of connected components in $G - F$ is η.

▶ **Proposition 2.2** ([23]). *Graphic and co-graphic matroids are representable over any field of size ≥ 2 and such a representation can be found in time polynomial in the size of the graph.*

Elongation of Matroid. Let $M = (E, \mathcal{I})$ be a matroid and k be an integer such that $\mathsf{rank}(M) \leq k \leq |E|$. A k-elongation matroid M_k of M is a matroid with the universe as E and $S \subseteq E$ is a basis of M_k if and only if, it contains a basis of M and $|S| = k$. Observe that the rank of the matroid M_k is k.

▶ **Proposition 2.3** ([18]). *Let M be a linear matroid of rank r, over a ground set of size n, which is representable over a field $\mathbb{F}$. Given a number $\ell \geq r$, we can compute a representation of the ℓ-elongation of M, over the field $\mathbb{F}(X)$ in $\mathcal{O}(nr\ell)$ field operations over $\mathbb{F}$.*

α-Matroid Parity. In our algorithms we use a known algorithm for α-Matroid Parity. Below we define α-Matroid Parity problem formally and state its algorithmic result.

α-Matroid Parity **Parameter:** α, q
Input: A representation A_M of a linear matroid $M = (E, \mathcal{I})$, a partition $\mathcal{P}$ of E into blocks of size α and a positive integer q.
Question: Does there exist an independent set which is a union of q blocks?

▶ **Proposition 2.4** ([18, 21]). *α-Matroid Parity can be solved in $\mathcal{O}(2^{\omega q \alpha}||A_M||^{\mathcal{O}(1)})$ time.*

3 FPT Algorithm for Simultaneous Feedback Edge Set

In this section we design an algorithm for Sim-FES by giving a reduction to α-Matroid Parity on the direct sum of elongated co-graphic matroids associated with graphs restricted to different color classes.

We describe our algorithm, Algo-SimFES, for Sim-FES. Let $(G, k, \mathsf{col} : E(G) \rightarrow 2^{[\alpha]})$ be an input instance to Sim-FES. Recall that for $i \in [\alpha]$, G_i is the graph with vertex set as $V(G)$ and edge set as $E(G_i) = \{e \in E(G) \mid i \in \mathsf{col}(e)\}$. Let $n = |V(G)|$. Note that $n = |V(G_i)|$ for all $i \in [\alpha]$. Let η_i be the number of connected components in G_i. To make G_i acyclic we need to delete at least $|E(G_i)| - n + \eta_i$ edges from G_i. Therefore, if there is $i \in [\alpha]$ such that $|E(G_i)| - n + \eta_i > k$, then Algo-SimFES returns No. We let $k_i = |E(G_i)| - n + \eta_i$. Observe that for $i \in [\alpha]$, $0 \leq k_i \leq k$. We need to delete at least k_i edges from $E(G_i)$ to make G_i acyclic. Therefore, the algorithm Alg-SimFES for each $i \in [\alpha]$, guesses k'_i, where $k_i \leq k'_i \leq k$ and computes a solution S of Sim-FES such that $|S \cap E(G_i)| = k'_i$. Let $M_i = (E_i, \mathcal{I}_i)$ be the k'_i-elongation of the co-graphic matroid associated with G_i.

▶ **Proposition 3.1** ($*$[1]). *Let G be a graph with η connected components and M be an r-elongation of the co-graphic matroid associated with G, where $r \geq |E(G)| - |V(G)| + \eta$. Then $B \subseteq E(G)$ is a basis of M if and only if the subgraph $G - B$ is acyclic and $|B| = r$.*

By Proposition 3.1, for any basis F_i in M_i, $G_i - F_i$ is acyclic. Therefore, our objective is to compute $F \subseteq E(G)$ such that $|F| = k$ and the elements of F restricted to the elements of M_i form a basis for all $i \in [\alpha]$. For this we will construct an instance of α-Matroid Parity as follows. For each $e \in E(G)$ and $i \in \mathsf{col}(e)$, we use e^i to denote the corresponding element in M_i. For each $e \in E(G)$, by $\mathsf{Original}(e)$ we denote the set of elements $\{e^j \mid j \in \mathsf{col}(e)\}$. For each edge $e \in E(G)$, we define $\mathsf{Fake}(e) = \{e^j \mid j \in [\alpha] - \mathsf{col}(e)\}$. Finally, for each edge $e \in E(G)$, by $\mathsf{Copies}(e)$ we denote the set $\mathsf{Original}(e) \cup \mathsf{Fake}(e)$. Let $\mathsf{Fake}(G) = \bigcup_{e \in E(G)} \mathsf{Fake}(e)$. Furthermore, let $\tau = |\mathsf{Fake}(G)| = \sum_{e \in E(G)} |\mathsf{Fake}(e)|$ and

[1] Proofs of results marked with ($*$) can be found in the full version of the paper [2].

Algorithm 1: Pseudocode of Algo-SimFES

```
Input: A graph $G, k \in \mathbb{N}$ and $\mathsf{col} : E(G) \to 2^{[\alpha]}$.
Output: YES if there is a simultaneous feedback edge set of size $\leq k$ and NO
        otherwise.
1  Let $\eta_i$ be the number of connected components in $G_i$ for all $i \in [\alpha]$
2  $k_i := |E(G_i)| - n + \eta_i$ for all $i \in [\alpha]$
3  if there exists $i \in [\alpha]$ such that $k_i > k$ then
4      return NO
5  for $(k'_1, \dots, k'_\alpha) \in ([k] \cup \{0\})^\alpha$ such that $k_i \leq k'_i$ for all $i \in [\alpha]$ do
6      Let $M_i$ be the $k'_i$-elongation of the co-graphic matroid associated with $G_i$.
7      Let $M_{\alpha+1} = U_{\tau,k'}$ over the gound set Fake(G), where, $k' = \sum_{i\in[\alpha]}(k - k'_i)$.
8      Let $M := \bigoplus_{i\in[\alpha+1]} M_i$.
9      For each $e \in E(G)$, let Copies($e$) be the block of elements of $M$.
10     if there is an independent set of $M$ composed of $k$ blocks then
11         return YES
12 return NO
```

$k' = \sum_{i\in[\alpha]}(k - k'_i)$. Let $M_{\alpha+1} = (E_{\alpha+1}, \mathcal{I}_{\alpha+1})$ be a uniform matroid over the ground set $\mathsf{Fake}(G)$. That is, $M_{\alpha+1} = U_{\tau,k'}$. By Propositions 2.1 to Proposition 2.3 we know that M_is are representable over $\mathbb{F}_p(X)$, where $p > \max(\tau, 2)$ is a prime number and their representation can be computed in polynomial time. Let A_i be the linear representation of M_i for all $i \in [\alpha + 1]$. Notice that $E_i \cap E_j = \emptyset$ for all $1 \leq i \neq j \leq \alpha + 1$. Let M denote the direct sum $M_1 \oplus \cdots \oplus M_{\alpha+1}$ with its representation matrix being A_M. Note that the ground set of M is $\bigcup_{e\in E(G)} \mathsf{Copies}(e)$. Now we define an instance of α-MATROID PARITY, which is the linear representation A_M of M and the partition of ground set into $\mathsf{Copies}(e)$, $e \in E(G)$. Notice that for all $e \in E(G)$, $|\mathsf{Copies}(e)| = \alpha$. Also for each $i \in [\alpha]$, $\mathsf{rank}(M_i) = k'_i$ and $\mathsf{rank}(M_{\alpha+1}) = k' = \sum_{i\in[\alpha]}(k - k'_i)$. This implies that $\mathsf{rank}(M) = \alpha k$.

Now Algo-SimFES outputs YES if there is a basis (an independent set of cardinality αk) of M which is a union of k blocks in M and otherwise outputs NO. Algo-SimFES uses the algorithm mentioned in Proposition 2.4 to check whether there is an independent set of M, composed of blocks. A pseudocode of Algo-SimFES can be found in Algorithm 1.

▶ **Lemma 3.2.** *Algo-SimFES is correct.*

Proof. Let $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ be a YES instance of SIM-FES and let $F \subseteq E(G)$, where $|F| = k$ be a solution of $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$. Let $k_i = |E(G_i)| - n + \eta_i$, where η_i is the number of connected components in G_i, for all $i \in [\alpha]$. For all $i \in [\alpha]$, let $k'_i = |F \cap E(G_i)|$. Since F is a solution, $k_i \leq k'_i$ for all $i \in [\alpha]$. This implies that Algo-SimFES will not execute Step 4. Consider the **for** loop for the choice $(k'_1, \dots, k'_\alpha)$. We claim that the columns corresponding to $S = \bigcup_{e\in F} \mathsf{Copies}(e)$ form a basis in M and it is union of k blocks. Note that $|S| = \alpha k$ by construction. For all $i \in [\alpha]$, let $F^i = \{e^i \mid e \in F, i \in \mathsf{col}(e)\}$, which is subset of ground set of M_i. By Proposition 3.1, for all $i \in [\alpha]$, F^i is a basis for M_i. This takes care of all the edges in $\cup_{e\in F}\mathsf{Original}(e)$. Now let $S^* = S - \cup_{i\in[\alpha]} F^i = \cup_{e\in F}\mathsf{Fake}(e)$. Observe that $|S^*| = \sum_{i\in[\alpha]}(k - k'_i) = k'$. Also, S^* is a subset of ground set of $U_{\tau,k'}$ and thus is a basis since $|S^*| = k'$. Hence S is a basis of M. Note that S is the union of blocks corresponding to $e \in F$ and hence is union of k blocks. Therefore, Algo-SimFES will output YES.

In the reverse direction suppose Algo-SimFES outputs Yes. This implies that there is a basis, say S, that is the union of k blocks. By construction S corresponds to union of the sets $\mathsf{Copies}(e)$ for some k edges in G. Let these edges be $F = \{e_1, \ldots, e_k\}$. We claim that F is a solution of $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$. Clearly $|F| = k$. Since S is a basis of M, for each $i \in [\alpha]$, $B(i) = S \cap \{e^i \mid e \in E(G_i)\}$ is a basis in M_i. Let $F(i) = \{e \mid e^i \in B(i)\} \subseteq F$. Since $B(i)$ is a basis of M_i, by Proposition 3.1, $G_i - F(i)$ is an acyclic graph. ◀

▶ **Lemma 3.3.** *Algo-SimFES runs in deterministic time $\mathcal{O}(2^{\omega k\alpha+\alpha\log k}|V(G)|^{\mathcal{O}(1)})$.*

Proof. The for loop runs $(k+1)^\alpha$ times. The step 10 uses the algorithm mentioned in Proposition 2.4, which takes time $\mathcal{O}(2^{\omega k\alpha}||A_M||^{\mathcal{O}(1)}) = \mathcal{O}(2^{\omega k\alpha}|V(G)|^{\mathcal{O}(1)})$. All other steps in the algorithm takes polynomial time. Thus, the total running time is $\mathcal{O}(2^{\omega k\alpha+\alpha\log k}|V(G)|^{\mathcal{O}(1)})$. ◀

Since α-Matroid Parity for $\alpha = 2$ can be solved in polynomial time [19] algorithm Algo-SimFES runs in polynomial time for $\alpha = 2$. This gives us the following theorem.

▶ **Theorem 3.4.** Sim-FES *is in* FPT *and when $\alpha = 2$* Sim-FES *is in P.*

4 Hardness results for Sim-FES

In this section we show that when $\alpha = 3$, Sim-FES is NP-Hard. Furthermore, from our reduction we conclude that it is unlikely that Sim-FES admits a subexponential-time algorithm. We give a reduction from Vertex Cover (VC) in cubic graphs to the special case of Sim-FES where $\alpha = 3$. Let (G, k) be an instance of VC in cubic graphs, which asks whether the graph G has a vertex cover of size at most k. We assume without loss of generality that $k \leq |V(G)|$. It is known that VC in cubic graphs is NP-hard [22] and unless the ETH fails, it cannot be solved in time $\mathcal{O}^\star(2^{o(|V(G)|+|E(G)|)})$[2] [17]. Thus, to prove that when $\alpha = 3$, it is unlikely that Sim-FES admits a parameterized subexponential time algorithm (an algorithm of running time $\mathcal{O}^\star(2^{o(k)})$), it is sufficient to construct (in polynomial time) an instance of the form $(G', k' = \mathcal{O}(|V(G)| + |E(G)|), \mathsf{col}' : E' \to 2^{[3]})$ of Sim-FES that is equivalent to (G, k). Refer Figure 1 for an illustration of the construction.

To construct $(G', k', \mathsf{col}' : E(G') \to 2^{[3]})$, we first construct an instance $(\widehat{G}, \widehat{k})$ of VC in subcubic graphs which is equivalent to (G, k). We set $V(\widehat{G}) = V(G) \cup (\bigcup_{\{v,u\}\in E(G)} \{x_{v,u}, x_{u,v}\})$, and $E(\widehat{G}) = \{\{x_{v,u}, x_{u,v}\} : \{v, u\} \in E(G)\} \cup (\bigcup_{\{v,u\}\in E(G)} \{\{v, x_{v,u}\}, \{u, x_{u,v}\}\})$. That is, the graph $\widehat{G}$ is obtained from the graph G by subdividing each edge in $E(G)$ twice.

▶ **Lemma 4.1** ($*$)**.** *G has a vertex cover of size k if and only if $\widehat{G}$ has a vertex cover of size $\widehat{k} = k + |E(G)|$*

Observe that in $\widehat{G}$ every path between two degree-3 vertices contains an edge of the form $\{x_{v,u}, x_{u,v}\}$. Thus, the following procedure results in a partition (M_1, M_2, M_3) of $E(\widehat{G})$ such that for all $i \in [3]$, $\{v, u\} \in M_i$ and $\{v', u'\} \in M_i \setminus \{\{v, u\}\}$, it holds that $\{v, u\} \cap \{v', u'\} = \emptyset$. Initially, $M_1 = M_2 = M_3 = \emptyset$. For each degree-3 vertex v, let $\{v, x\}$, $\{v, y\}$ and $\{v, z\}$ be the edges containing v. We insert $\{v, x\}$ into M_1, $\{v, y\}$ into M_2, and $\{v, z\}$ into M_3 (the choice of which edge is inserted into which set is arbitrary). Finally, we insert each edge of the form $\{x_{v,u}, x_{u,v}\}$ into a set M_i that contains neither $\{v, x_{v,u}\}$ nor $\{u, x_{u,v}\}$.

We are now ready to construct the instance $(G', k', \mathsf{col}' : E(G') \to 2^{[3]})$. Let $V(G') = V(\widehat{G}) \cup V^\star$, where $V^\star = \{v^\star : v \in V(\widehat{G})\}$ contains a copy $v^\star$ of each vertex v in $V(\widehat{G})$. The

[2] $\mathcal{O}^\star$ notation suppresses polynomial factors in the running-time expression.

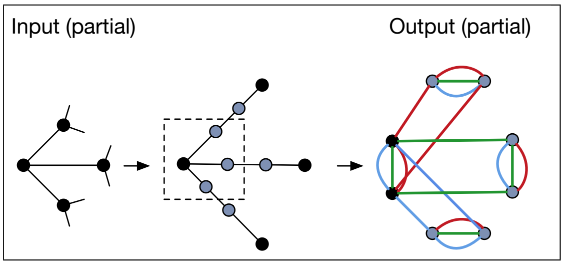


Figure 1 The construction given in the proof of Theorem 4.3.

set $E(G')$ and coloring col' are constructed as follows. For each vertex $v \in V(\widehat{G})$, add an edge $\{v, v^\star\}$ into $E(G')$ and its color-set is $\{1,2,3\}$. For each $i \in [3]$ and for each $\{v,u\} \in M_i$, add the edges $\{v,u\}$ and $\{v^\star, u^\star\}$ into $E(G')$ and its color-set is $\{i\}$. We set $k' = \widehat{k}$. Clearly, the instance $(G', k', \mathsf{col}' : E(G') \to 2^{[3]})$ can be constructed in polynomial time, and it holds that $k' = \mathcal{O}(|V(G)| + |E(G)|)$.

Lemma 4.2 proves that $(\widehat{G}, \widehat{k})$ is a YES instance of VC if and only if $(G', k', \mathsf{col}' : E(G') \to 2^{[3]})$ is a YES instance of SIM-FES. Observe that because of the above mentioned property of the partition (M_1, M_2, M_3) of $E(\widehat{G})$, we ensure that in G', no vertex participates in two (or more) monochromatic cycles that have the same color. By construction, each monochromatic cycle in G' is of the form $v - v^\star - u^\star - u - v$, where $\{v,u\} \in E(\widehat{G})$, and for each edge $\{v,u\} \in E(G')$, where either $v, u \in V(\widehat{G})$ or $v, u \in V^\star$, G' contains exactly one monochromatic cycle of this form.

▶ **Lemma 4.2** (∗). *$(\widehat{G}, \widehat{k})$ is a* YES *instance of VC if and only if $(G', k', \mathsf{col}' : E(G') \to 2^{[3]})$ is a* YES *instance of* SIM-FES.

We get the following theorem and its proof follows from Lemma 4.1 and Lemma 4.2.

▶ **Theorem 4.3.** SIM-FES *where $\alpha = 3$ is* NP-hard*. Furthermore, unless the Exponential Time Hypothesis* (ETH) *fails,* SIM-FES *when $\alpha = 3$ cannot be solved in time $\mathcal{O}^*(2^{o(k)})$.*

5 Tight Lower Bounds for Simultaneous Feedback Edge Set

We show that SIM-FES parameterized by k is $W[2]$ hard when $\alpha = \mathcal{O}(|V(G)|)$ and $W[1]$ hard when $\alpha = \mathcal{O}(\log(|V(G)|))$. Our reductions follow the approach of Agrawal et al. [1].

W[2] Hardness of Sim-FES when $\alpha = \mathcal{O}(|V(G)|)$. We give a reduction from HITTING SET (HS) to SIM-FES where $\alpha = \mathcal{O}(|V(G)|)$. Let $(U = \{u_1, u_2, \ldots, u_{|U|}\}, \mathcal{F} = \{F_1, F_2, \ldots, F_{|\mathcal{F}|}\}, k)$ be an instance of HS, where $\mathcal{F} \subseteq 2^U$, which asks whether there exists a subset $S \subseteq U$ of size at most k such that for all $F \in \mathcal{F}$, $S \cap F \neq \emptyset$. It is known that HS parameterized by k is W[2]-hard (see, e.g., [9]). Thus, to prove the result, it is sufficient to construct (in polynomial time) an instance of the form $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ of SIM-FES that is equivalent to $(U, \mathcal{F}, k)$, where $\alpha = \mathcal{O}(|V(G)|)$. We construct a graph G such that $V(G) = \mathcal{O}(|U||\mathcal{F}|)$ and the number of colors used will be $\alpha = |\mathcal{F}|$. The intuitive idea is to have one edge per element in the universe which is colored with all the indices of sets in the family $\mathcal{F}$ that contains the element and for each $F_i \in \mathcal{F}$ creating a unique monochromatic cycle with color i which passes through all the edges corresponding to the elements it contain. We explain the reduction formally in the next paragraph.

Without loss of generality we assume that each set in $\mathcal{F}$ contains at least two elements from U. The instance $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ is constructed as follows. Initially, $V(G) = E(G) = \emptyset$. For each element $u_i \in U$, insert two new vertices into $V(G)$, v_i and w_i, add the edge $\{v_i, w_i\}$ into $E(G)$ and let $\{j \mid F_j \in \mathcal{F}, u_i \in F_j\}$ be its color-set. Now, for all $1 \leq i < j \leq |U|$ and for all $1 \leq t \leq |\mathcal{F}|$ such that $u_i, u_j \in F_t$ and $\{u_{i+1}, \ldots, u_{j-1}\} \cap F_t = \emptyset$, perform the following operation: add a new vertex into $V(G)$, $s_{i,j,t}$, add the edges $\{w_i, s_{i,j,t}\}$ and $\{s_{i,j,t}, v_j\}$ into $E(G)$ and let their color-set be $\{t\}$. Moreover, for each $1 \leq t \leq |\mathcal{F}|$, let u_i and u_j be the elements with the largest and smallest index contained in F_t, respectively, and perform the following operation: add a new vertex into $V(G)$, $s_{i,j,t}$, add the edges $\{w_i, s_{i,j,t}\}$ and $\{s_{i,j,t}, v_j\}$ into $E(G)$, and let their color-set be $\{t\}$. Observe that $|V(G)| = \mathcal{O}(|U||\mathcal{F}|)$ and that $\alpha = |\mathcal{F}|$. Therefore, $\alpha = \mathcal{O}(|V(G)|)$. It remains to show that the instances (G, k, col) and $(U, \mathcal{F}, k)$ are equivalent. By construction, each monochromatic cycle in G is of the form $v_{i_1} - w_{i_1} - s_{i_1,i_2,t} - v_{i_2} - w_{i_2} - s_{i_2,i_3,t} - \cdots - v_{i_{|F_t|}} - w_{i_{|F_t|}} - s_{i_{|F_t|},i_1,t} - v_{i_1}$, where $\{u_{i_1}, u_{i_2}, \ldots, u_{i_{|F_t|}}\} = F_t \in \mathcal{F}$, and for each set $F_t \in \mathcal{F}$, G contains exactly one such monochromatic cycle.

▶ **Lemma 5.1** (∗). *$(U, \mathcal{F}, k)$ is a* Yes *instance of* HS *if and only if $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ is a* Yes *instance of* Sim-FES*.*

▶ **Theorem 5.2.** Sim-FES *parameterized by k, when $\alpha = \mathcal{O}(|V(G)|)$, is $W[2]$-hard.*

W[1] Hardness of Sim-FES when $\alpha = \mathcal{O}(\log |V(G)|)$. We modify the reduction given in the proof of Theorem 5.2 to show that when $\alpha = \mathcal{O}(\log |V(G)|)$, Sim-FES is W[1]-hard with respect to the parameter k. This result implies that the dependency on α of our $\mathcal{O}((2^{\mathcal{O}(\alpha)})^k n^{\mathcal{O}(1)})$-time algorithm for Sim-FES is optimal in the sense that it is unlikely that there exists an $\mathcal{O}((2^{o(\alpha)})^k n^{\mathcal{O}(1)})$-time algorithm for this problem.

We give a reduction from a variant of HS, called Partitioned Hitting Set (PHS), to Sim-FES where $\alpha = \mathcal{O}(\log |V(G)|)$. The input of PHS consists of a universe U, a collection $\mathcal{F} = \{F_1, F_2, \ldots, F_{|\mathcal{F}|}\}$, where each F_i is a family of *disjoint* subsets of U, and a parameter k. The goal is to decide the existence of a subset $S \subseteq U$ of size at most k such that for all $f \in (\bigcup_{F \in \mathcal{F}} F)$, $S \cap f \neq \emptyset$. It is known that the special case of PHS where $|\mathcal{F}| = \mathcal{O}(\log(|U||(\bigcup \mathcal{F})|))$ is W[1]-hard when parameterized by k (see, e.g., [1]). Thus, to prove the theorem, it is sufficient to construct (in polynomial time) an instance of the form $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ of Sim-FES that is equivalent to $(U, \mathcal{F}, k)$, where $\alpha = \mathcal{O}(\log |V(G)|)$. The construction of the graph G is exactly similar to the one in Theorem 5.2. But instead of creating a unique monochromatic cycle with a color i for each $f_i \in \bigcup \mathcal{F}$, for each $F_i \in \mathcal{F}$ we create $|F_i|$ vertex disjoint cycles of same color i. Since for each $F \in \mathcal{F}$ the sets in F are pairwise disjoint, guarantees the correctness. Formal description of the reduction is given below.

Without loss of generality we assume that each set in $\bigcup_{F \in \mathcal{F}} F$ contains at least two elements from U. The instance $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ is constructed as follows. Initially, $V(G) = E(G) = \emptyset$. For each element $u_i \in U$, insert two new vertices v_i and w_i into $V(G)$, and add the edge $\{v_i, w_i\}$ into $E(G)$ with its color-set being $\{j : F_j \in \mathcal{F}, u_i \in (\bigcup F_j)\}$. Now, for all $1 \leq i < j \leq |U|$ and for all $1 \leq t \leq |\mathcal{F}|$ such that there exists $f \in F_t$ satisfying $u_i, u_j \in f$ and $\{u_{i+1}, \ldots, u_{j-1}\} \cap f = \emptyset$, perform the following operation: add a new vertex $s_{i,j,t}$ into $V(G)$, add the edges $\{w_i, s_{i,j,t}\}$ and $\{s_{i,j,t}, v_j\}$ into $E(G)$ with both of its color-set being $\{t\}$. Moreover, for each $1 \leq t \leq |\mathcal{F}|$ and $f \in F_t$, let u_i and u_j be the elements with the largest and smallest index contained in f, respectively, we perform the following operation: add a new vertex into $V(G)$, $s_{i,j,t}$, add the edges $\{w_i, s_{i,j,t}\}$ and $\{s_{i,j,t}, v_j\}$ into $E(G)$, and let their color-set be $\{t\}$. Observe that $|V(G)| = \mathcal{O}(|U||(\bigcup \mathcal{F})|)$ and that $\alpha = |\mathcal{F}|$.

Since $|\mathcal{F}| = \mathcal{O}(\log(|U||(\bigcup \mathcal{F})|))$, we have that $\alpha = \mathcal{O}(\log |V(G)|)$. Since the sets in each family F_i are disjoint, the construction implies that each monochromatic cycle in G is of the form $v_{i_1} - w_{i_1} - s_{i_1,i_2,t} - v_{i_2} - w_{i_2} - s_{i_2,i_3,t} - \cdots - v_{i_{|f|}} - w_{i_{|f|}} - s_{i_{|f|},i_1,t} - v_{i_1}$, where $\{u_{i_1}, u_{i_2}, \ldots, u_{i_{|F_t|}}\} = f$ for a set $f \in F_t \in \mathcal{F}$, and for each set $f \in F_t \in \mathcal{F}$, G contains a monochromatic cycle of this form. By using the arguments similar to one in the proof of Lemma 5.1, we get that the instances $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ and $(U, \mathcal{F}, k)$ are equivalent. Hence we get the following theorem.

▶ **Theorem 5.3.** Sim-FES *parameterized by k, when $\alpha = \mathcal{O}(\log |V(G)|)$ is $W[1]$-hard.*

6 Kernel for Simultaneous Feedback Edge Set

In this section we give a kernel for Sim-FES with $\mathcal{O}((k\alpha)^{\mathcal{O}(\alpha)})$ vertices. We start by applying preprocessing rules similar in spirit to the ones used to obtain a kernel for Feedback Vertex Set, but it requires subtle differences due to the fact that we handle a problem where edges rather than vertices are deleted, as well as the fact that the edges are colored (in particular, each edge in Sim-FES has a color-set, while each vertex in Sim-FVS is uncolored). We obtain an approximate solution by computing a spanning tree per color. We rely on the approximate solution to bound the number of vertices whose degree in certain subgraphs of G is not equal to 2. Then, the number of the remaining vertices is bounded by adapting the "interception"-based approach of Agrawal et al. [1] to a form relevant to Sim-FES.

Let $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ be an instance of Sim-FES. For each color $i \in [\alpha]$ recall G_i is the graph consisting of the vertex-set $V(G)$ and the edge-set $E(G_i)$ includes every edge in $E(G)$ whose color-set contains the color i. It is easy to verify that the following rules are correct when applied exhaustively in the order in which they are listed. We note that the resulting instance can contain multiple edges.

- **Reduction Rule 1:** If $k < 0$, return that $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ is a No instance.
- **Reduction Rule 2:** If for all $i \in [\alpha]$, G_i is acyclic, return that $(G, k, \mathsf{col} : E(G) \to 2^{[\alpha]})$ is a Yes instance.
- **Reduction Rule 3:** If there is a self-loop at a vertex $v \in V(G)$, then remove v from G and decrement k by 1.
- **Reduction Rule 4:** If there exists an isolated vertex in G, then remove it.
- **Reduction Rule 5:** If there exists $i \in [\alpha]$ and an edge whose color-set contains i but it does not participate in any cycle in G_i, remove i from its color-set. If the color-set becomes empty, remove the edge.
- **Reduction Rule 6:** If there exists $i \in [\alpha]$ and a vertex v of degree exactly two in G, remove v and connect its two neighbors by an edge whose color-set is the same as the color-set of the two edges incident to v (we prove in Lemma 6.1 that the color set of two edges are same).

▶ **Lemma 6.1** $(*)$. *Reduction rule 6 is safe.*

We apply Reduction Rule 1 to 6 exhaustively (in that order). The safeness of Reduction Rules 1 to 5 are easy to see. Lemma 6.1 proves the safeness Reduction Rule 6. After this, we follow the approach similar to that in [1] to bound the size of the instance. This gives the following theorem.

▶ **Theorem 6.2** $(*)$. Sim-FES *admits a kernel with $(k\alpha)^{\mathcal{O}(\alpha)}$ vertices.*

7 Maximum Simultaneous Acyclic Subgraph

In this section we design an algorithm for Maximum Simultaneous Acyclic Subgraph. Let $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$ be an input to Max-Sim-Subgraph. A set $F \subseteq E(G)$ such that for all $i \in [\alpha]$, $G[F_i]$ is acyclic is called *simultaneous forest*. Here, $F_i = \{e \in F \mid i \in \mathsf{col}(e)\}$, denotes the subset of edges of F which has the integer i in its image when the function col is applied to it. We will solve Max-Sim-Subgraph by reducing to an equivalent instance of the α-Matroid Parity problem and then using the algorithm for the same.

We start by giving a construction that reduces the Max-Sim-Subgraph to α-Matroid Parity. Let $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$ be an input to Max-Sim-Subgraph. Given, $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$, for $i \in [\alpha]$, recall that by G_i we denote the graph with the vertex set $V(G_i) = V(G)$ and the edge set $E(G_i) = \{e^i \mid e \in E(G) \text{ and } i \in \mathsf{col}(e)\}$. For each edge $e \in E(G)$, we will have its distinct copy in G_i if $i \in \mathsf{col}(e)$. Thus, for each edge $e \in E(G)$, by $\mathsf{Original}(e)$ we denote the set of edges $\{e^j \mid j \in \mathsf{col}(e)\}$. On the other hand for each edge $e \in E(G)$, by $\mathsf{Fake}(e)$ we denote the set of edges $\{e^j \mid j \in [\alpha] - \mathsf{col}(e)\}$. Finally, for each edge $e \in E(G)$, by $\mathsf{Copies}(e)$ we denote the set $\mathsf{Original}(e) \cup \mathsf{Fake}(e)$. Let $M_i = (E_i, \mathcal{I}_i)$ denote the graphic matroid on G_i. That is, edges of G_i forms the universe E_i and $\mathcal{I}_i$ contains, $S \subseteq E(G_i)$ such that $G_i[S]$ forms a forest. By Proposition 2.2 we know that graphic matroids are representable over any field and given a graph G one can find the corresponding representation matrix in time polynomial in $|V(G)|$. Let A_i denote the linear representation of M_i. That is, A_i is a matrix over $\mathbb{F}_2$, where the set of columns of A_i are denoted by $E(G_i)$. In particular, A_i has dimension $d \times |E(G_i)|$, where $d = \mathsf{rank}(M_i)$. A set $X \subseteq E(G_i)$ is independent (that is $X \in \mathcal{I}_i$) if and only if the corresponding columns are linearly independent over $\mathbb{F}_2$. Let $\mathsf{Fake}(G)$ denote the set of edges in $\bigcup_{e \in E(G)} \mathsf{Fake}(e)$. Furthermore, let $\tau = |\mathsf{Fake}(G)| = \sum_{e \in E(G)} |\mathsf{Fake}(e)|$. Let $M_{\alpha+1}$ be the uniform matroid over $\mathsf{Fake}(G)$ of rank τ. That is, $E_{\alpha+1} = \mathsf{Fake}(G)$ and $M_{\alpha+1} = U_{\tau,\tau}$. Let I_τ denote the identity matrix of dimension $\tau \times \tau$. Observe that, $A_{\alpha+1} = I_\tau$ denotes the linear representation of $M_{\alpha+1}$ over $\mathbb{F}_2$. Notice that $E_i \cap E_j = \emptyset$ for all $1 \leq i \neq j \leq \alpha + 1$. Let M denote the direct sum of $M_1 \oplus \cdots \oplus M_{\alpha+1}$ with its representation matrix being A_M.

Now we are ready to define an instance of α-Matroid Parity. The ground set is the columns of A_M, which is indexed by edges in $\bigcup_{e \in E(G)} \mathsf{Copies}(e)$. Furthermore, the ground set is partitioned into $\mathsf{Copies}(e)$, $e \in E(G)$, which are called blocks. The main technical lemma of this section on which the whole algorithm is based is the following.

▶ **Lemma 7.1** (∗). *Let $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$ be an instance of* Max-Sim-Subgraph. *Then G has a simultaneous forest of size q if and only if $(A_M, \biguplus_{e \in E(G)} \mathsf{Copies}(e), q)$ is a* Yes *instance of α-*Matroid Parity. *Furthermore, given $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$ we can obtain an instance $(A_M, \biguplus_{e \in E(G)} \mathsf{Copies}(e), q)$ in polynomial time.*

We will use the polynomial time reduction provided in Lemma 7.1 to get the desired FPT algorithm for Max-Sim-Subgraph. Towards this will use the following FPT result regarding α-Matroid Parity for our FPT as well as for an exact exponential time algorithm.

Given an instance $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$ of Max-Sim-Subgraph we first apply Lemma 7.1 and obtain an instance $(A_M, \biguplus_{e \in E(G)} \mathsf{Copies}(e), q)$ of α-Matroid Parity and then apply Proposition 2.4 to obtain the following result.

▶ **Theorem 7.2.** Max-Sim-Subgraph *can be solved in time $\mathcal{O}(2^{\omega q \alpha}|V(G)|^{\mathcal{O}(1)})$.*

Let $(G, q, \mathsf{col} : E(G) \to 2^{[\alpha]})$ be an instance of Max-Sim-Subgraph. Observe that q is upper bounded by $\alpha(|V(G)| - 1)$. Thus, as a corollary to Theorem 7.2 we get an exact algorithm for finding the largest sized simultaneous acyclic subgraph, running in time $\mathcal{O}(2^{\omega n\alpha^2}|V(G)|^{\mathcal{O}(1)})$.

References

1 A. Agrawal, D. Lokshtanov, A. E. Mouawad, and S. Saurabh. Simultaneous feedback vertex set: A parameterized perspective. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS*, pages 7:1–7:15, 2016.

2 Akanksha Agrawal, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Simultaneous feedback edge set: A parameterized perspective. *CoRR*, abs/1611.07701, 2016. URL: `https://arxiv.org/abs/1611.07701`.

3 Noga Alon and Timothy H. Marshall. Homomorphisms of edge-colored graphs and coxeter groups. *Journal of Algebraic Combinatorics*, 8(1):5–13, 1998.

4 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, September 1999.

5 József Balogh, János Barát, Dániel Gerbner, András Gyárfás, and Gábor N. Sárközy. Partitioning 2-edge-colored graphs by monochromatic paths and cycles. *Combinatorica*, 34(5):507–526, 2014.

6 Jørgen Bang-Jensen and Gregory Gutin. Alternating cycles and paths in edge-coloured multigraphs: a survey. *Discrete Mathematics*, 165:39–60, 1997.

7 Leizhen Cai and Junjie Ye. Dual connectedness of edge-bicolored graphs and beyond. In *Mathematical Foundations of Computer Science, MFCS*, volume 8635, pages 141–152, 2014.

8 W. S. Chou, Yannis Manoussakis, Olga Megalakaki, M. Spyratos, and Zs. Tuza. Paths through fixed vertices in edge-colored graphs. *Mathématiques et sciences humaines*, 127:49–58, 1994.

9 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Joham M.M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Foundations of Computer Science (FOCS), IEEE 52nd Annual Symposium*, pages 150–159, 2011.

11 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

12 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 764–775, 2016.

13 M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman & Co., 1979.

14 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

15 Mikio Kano and Xueliang Li. Monochromatic and heterochromatic subgraphs in edge-colored graphs-a survey. *Graphs and Combinatorics*, 24(4):237–263, 2008.

16 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.

17 Christian Komusiewicz. Tight running time lower bounds for vertex deletion problems. *arXiv preprint arXiv:1511.05449*, 2015.

18 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. In *Automata, Languages, and Programming*, pages 922–934. Springer, 2015.

19 László Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Series B*, 28(2):208–236, 1980.

20 Yannis Manoussakis. Alternating paths in edge-colored complete graphs. *Discrete Applied Mathematics*, 56(2):297–309, 1995.

21 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, October 2009.

22 Bojan Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82(1):102–117, 2001.

23 James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

24 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms, (TALG)*, 6(2):32:1–32:8, 2010.

25 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th annual ACM symposium on Theory of computing*, pages 887–898, 2012.

Kernels for Deletion to Classes of Acyclic Digraphs*

Akanksha Agrawal[1], **Saket Saurabh**[2], **Roohani Sharma**[3], and **Meirav Zehavi**[4]

1 **University of Bergen, Norway**
`akanksha.agrawal@uib.no`

2 **University of Bergen, Norway; and**
Institute of Mathematical Science, HBNI, India
`saket@imsc.res.in`

3 **University of Bergen, Norway**
`roohani@imsc.res.in`

4 **University of Bergen, Norway**
`meirav.zehavi@uib.no`

Abstract

In the Directed Feedback Vertex Set (DFVS) problem, we are given a digraph D on n vertices and a positive integer k and the objective is to check whether there exists a set of vertices S of size at most k such that $F = D - S$ is a directed acyclic digraph. In a recent paper, Mnich and van Leeuwen [*STACS 2016*] considered the kernelization complexity of DFVS with an additional restriction on F, namely that F must be an out-forest (Out-Forest Vertex Deletion Set), an out-tree (Out-Tree Vertex Deletion Set), or a (directed) pumpkin (Pumpkin Vertex Deletion Set). Their objective was to shed some light on the kernelization complexity of the DFVS problem, a well known open problem in the area of Parameterized Complexity. In this article, we improve the kernel sizes of Out-Forest Vertex Deletion Set from $\mathcal{O}(k^3)$ to $\mathcal{O}(k^2)$ and of Pumpkin Vertex Deletion Set from $\mathcal{O}(k^{18})$ to $\mathcal{O}(k^3)$. We also prove that the former kernel size is tight under certain complexity theoretic assumptions.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases out-forest, pumpkin, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.6

1 Introduction

Feedback Set problems form a family of fundamental combinatorial optimization problems. The input for Directed Feedback Vertex Set (DFVS) (Directed Feedback Edge Set (DFES)) consists of a directed graph (digraph) D and a positive integer k, and the question is whether there exists a subset $S \subseteq V(D)$ ($S \subseteq E(D)$) such that the graph obtained after deleting the vertices (edges) in S is a directed acyclic graph (DAG). Similarly, the input for Undirected Feedback Vertex Set (UFVS) (Undirected Feedback Edge

* The research leading to these results received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreements no. 306992 (S. Saurabh). Part of this work was done while M. Zehavi was visiting the Simons Institute for the Theory of Computing.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 6; pp. 6:1–6:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Set (UFES)) consists of an undirected graph G and a positive integer k, and the question is whether there exists a subset $S \subseteq V(G)$ ($S \subseteq E(G)$) such that the graph obtained after deleting the vertices (edges) in S is a forest.

All of these problems, excluding Undirected Feedback Edge Set, are NP-complete. Furthermore, Feedback Set problems are among Karp's 21 NP-complete problems and have been topic of active research from algorithmic [2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 18, 19, 20, 22, 24, 27, 32] as well as structural points of view [17, 21, 23, 26, 28, 29, 30]. In particular, such problems constitute one of the most important topics of research in Parameterized Complexity [6, 8, 9, 10, 12, 13, 22, 20, 24, 27, 32], spearheading development of new techniques. In this paper we study the parameterized complexity of restrictions of DFVS.

In Parameterized Complexity each problem instance is accompanied by a parameter k. A central notion in this field is the one of *fixed-parameter tractability (FPT).* This means, for a given instance (I, k), solvability in time $f(k)|I|^{\mathcal{O}(1)}$ where f is some function of k. Another central notion is the one of *kernelization.* A parameterized problem is said to admit a *kernel* of size $f(k)$ for some function f of k if there is a polynomial-time algorithm, called a *kernelization algorithm*, that translates any input instance to an equivalent instance of the same problem whose size is bounded by $f(k)$. In case the function f is polynomial in k, the problem is said to admit a *polynomial kernel.* For more information on these concepts we refer the reader to monographs such as [16, 11].

In contrast to UFVS which admits a polynomial kernel, the existence of a polynomial kernel for DFVS is still an open problem. The lack of progress on this question led to the consideration of various restrictions on input instances. In particular, we know of polynomial kernels for DFVS in tournaments as well as various generalizations [1, 3, 15]. However, the existence of a polynomial kernel for DFVS is open even for planar digraphs. Recently, in a very interesting article, to make progress on this question Mnich and van Leeuwen [25] considered DFVS with an additional restriction on *the output rather than the input.* Essentially, the basic philosophy of their program is the following: What happens to the kernelization complexity of DFVS when we consider subclasses of DAGs?

Mnich and van Leeuwen [25] inspected this question by considering the classes of out-forests, out-trees and (directed) pumpkins. An *out-tree* is a digraph where each vertex has in-degree at most 1 and the underlying (undirected) graph is a tree. An *out-forest* is a disjoint union of out-trees. On the other hand, a digraph is a *pumpkin* if it consists of a source vertex s and a sink vertex t, $s \neq t$, together with a collection of internally vertex-disjoint induced directed paths from s to t. Here, all vertices except s and t have in-degree 1 and out-degree 1. The examination of the classes of out-forests and out-trees was also motivated by the corresponding questions of UFVS and Tree Deletion Set in the undirected settings. Formally, Mnich and van Leeuwen [25] studied the following problems.

Out-Forest Vertex Deletion Set (OFVDS) **Parameter:** k
Input: A digraph D and a positive integer k.
Question: Is there a set $S \subseteq V(D)$ of size at most k such that $F = D \setminus S$ is an out-forest?

Out-Tree Vertex Deletion Set (OTVDS) and Pumpkin Vertex Deletion Set (PVDS) are defined in a similar manner, where instead of an out-forest, F should be an out-tree or a pumpkin, respectively. Mnich and van Leeuwen [25] showed that OFVDS and OTVDS admit kernels of size $\mathcal{O}(k^3)$ and PVDS admits a kernel of size $\mathcal{O}(k^{18})$.

Our Results and Methods. The objective of this article is to give improved kernels for OFVDS and PVDS. In this context, we obtain the following results.

- OFVDS admits an $\mathcal{O}(k^2)$ kernel and PVDS admits an $\mathcal{O}(k^3)$ kernel. These results improve upon the best known upper bounds $\mathcal{O}(k^3)$ and $\mathcal{O}(k^{18})$, respectively.
- For any $\epsilon > 0$, OFVDS does not admit a kernel for of size $\mathcal{O}(k^{2-\epsilon})$ unless coNP $\subseteq$ NP /poly.

To get the improved kernel for OFVDS we incorporate the Expansion Lemma as well as a factor 3-approximation algorithm for OFVDS in the kernelization routine given in [25]. The significance of this improvement also lies in the fact that we show that it is essentially tight. Due to space constraints, the lower bound is omitted from this version of the paper.

The kernelization algorithm for PVDS given in [25] works roughly as follows. It has two phases: (a) first it gives an $\mathcal{O}(k^5)$ kernel for a variant of the problem where we know the source and the sink of the pumpkin obtained after deleting the solution vertices; and (b) in the second phase, it reduces PVDS to polynomially many instances of a variant of the problem mentioned in the item (a) and then composes these instances to get a kernel of size $\mathcal{O}(k^{18})$. In fact given an instance (D, k) of PVDS, the kernelization algorithm of [25] outputs an equivalent instance (D', k') such that $k' = \mathcal{O}(k^{18})$. We take a completely different route and use “sun-flower style” reduction rules together with a marking strategy to obtain an equivalent instance (D', k') such that $|V(D)| + |E(D)| = \mathcal{O}(k^3)$ and $k' \leq k$. We believe the method applied in this algorithm could be useful also in other kernelization algorithms.

2 Preliminaries

We denote the set of natural numbers from 1 to n by $[n]$, and we use standard terminology from the book of Diestel [14] for graph-related terms which are not explicitly defined here. A digraph D is a pair $(V(D), E(D))$ such that $V(D)$ is a set of vertices and $E(D)$ is a set of ordered pairs of vertices. The underlying undirected graph G of D is a pair $(V(G), E(G))$ such that $V(G) = V(D)$ and $E(G)$ is a set of unordered pairs of vertices such that $\{u, v\} \in E(G)$ if and only if either $(u, v) \in E(D)$ or $(v, u) \in E(D)$. Let D be a digraph. For any $v \in V(D)$, we denote by $N^-(v)$ the set of in-neighbors of v, that is, $N^-(v) = \{(u, v) \mid (u, v) \in E(D)\}$. Similarly, we denote by $N^+(v)$ the set of out-neighbors of v, that is, $N^+(v) = \{(v, u) \mid (v, u) \in E(D)\}$. We denote the in-degree of a vertex v by $d^-(v) = |N^-(v)|$ and its out-degree by $d^+(v) = N^+(v)$. We say that $P = (u_1, \ldots, u_l)$ is a directed path in the digraph D is $u_1, \ldots, u_l \in V(D)$ and for all $i \in [l-1]$, $(u_i, u_{i+1}) \in E(D)$. A *collision* is a triplet (u, w, v) of distinct vertices such that $(u, w), (v, w) \in E(D)$.

3 Improved Kernel for Out-Forest Vertex Deletion Set

The aim of this section is to present an $\mathcal{O}(k^2)$ kernel for OFVDS. In Section 3.1 we state definitions and results relevant to our kernelization algorithm. Next, in Section 3.2, we design an algorithm for OFVDS that outputs a 3-approximate solution, which will also be used by our kernelization algorithm. Finally, in Section 3.3, we present our kernelization algorithm.

3.1 Prerequisites

We start by giving the definition of a q-expansion and the statement of the Expansion Lemma.

▶ **Definition 1** (q-Expansion). For a positive integer q, a set of edges $M \subseteq E(G)$ is a *q-expansion of A into B* if (i) every vertex in A is incident to exactly q vertices in M, and (ii) M saturates exactly $q|A|$ vertices in B (i.e., there is a set of $q|A|$ vertices in B which are incident to edges in M).

▶ **Lemma 2** (Expansion Lemma [11, 31]). *Let q be a positive integer and G be an undirected bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$, and there are no isolated vertices in B. Then, there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there exists a q-expansion of X into Y, and no vertex in Y has a neighbor outside X (i.e., $N(Y) \subseteq X$). Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

We will also need to rely on the well-known notion of l-flowers.

▶ **Definition 3** (l-Flower). An undirected graph G contains an *l-flower through v* if there is a family of cycles $\{C_1, \ldots, C_l\}$ in G such that for all distinct $i, j \in [l]$, $V(C_i) \cap V(C_j) = \{v\}$.

▶ **Lemma 4** ([11, 31]). *Given an undirected graph G and a vertex $v \in V(G)$, there is a polynomial-time algorithm that either outputs a $(k+1)$-flower through v or, if no such flower exists, outputs a set $Z_v \subseteq V(G) \setminus \{v\}$ of size at most $2k$ that intersects every cycle that passes through v in G.*

3.2 Approximation Algorithm for Out-Forest Vertex Deletion Set

This section presents a 3-factor approximation algorithm for OFVDS. Given an instance of OFVDS, let OPT be the minimum size of a solution. Formally, we solve the following.

3−Approximate Out-Forest Vertex Deletion Set (Approx-OFVDS)
Input: A DAG D.
Output: A subset $X \subseteq V(D)$ such that $D \setminus X$ is an out-forest and $|X| \leq 3 \cdot OPT$.

Given three distinct vertices $u_1, u_2, u_3 \in V(D)$, we say (u_1, u_2, u_3) is an *obstruction* if u_1 and u_2 are in-neighbors of u_3. Observe that any solution to OFVDS (and hence, Approx-OFVDS) must intersect any obstruction in at least 1 vertex. Moreover, it must intersect any cycle in at least 1 vertex. These observations form the basis of this algorithm.

▶ **Lemma 5.** Approx-OFVDS *can be solved in polynomial time.*

Proof. Given a digraph D, the algorithm first constructs (in polynomial time) a family $\mathcal{F}$ of obstructions and induced cycles in D such that the vertex sets of the entities in this family are pairwise disjoint. To this it, it initializes $\mathcal{F} = \emptyset$. Then, as long as there exists a vertex $v \in V(D)$ with at least two in-neighbors, u_1 and u_2, it inserts (v, u_1, u_2) into $\mathcal{F}$ and removes v, u_1 and u_2 from $\mathcal{F}$ (only for the purpose of the construction of $\mathcal{F}$). Once there is no vertex $v \in V(D)$ such that $d^-(v) \geq 2$, the digraph is a collection of directed vertex-disjoint cycles and paths. Each of these cycles is inserted into the family $\mathcal{F}$.

Let us now construct a solution, S_{app}, for Approx-OFVDS. For every obstruction in $\mathcal{F}$, we let S_{app} contain each of the three vertices of this obstruction. From every cycle C in $\mathcal{F}$ we pick an arbitrary vertex and insert it into S_{app}. Clearly, $|S_{app}| \leq 3|\mathcal{F}|$. It is now sufficient to prove is that $D \setminus S_{app}$ is an out-forest. Observe that no vertex v in $D \setminus S_{app}$ has in-degree at least 2, otherwise the obstruction consisting of v and two of its in-neighbors would have been inserted into $\mathcal{F}$ and hence also into S_{app}. Moreover, there is no directed cycle C in $D \setminus S_{app}$. Indeed, if the cycle C intersects an obstruction in $\mathcal{F}$, it is clear that it cannot exist in $D \setminus S_{app}$, and otherwise it would have been inserted into $\mathcal{F}$ and hence one of its vertices would have been inserted into S_{app}. We thus conclude that the theorem is correct. ◀

3.3 Kernelization algorithm for Out-Forest Vertex Deletion Set

We are are now ready to present our kernelization algorithm. Let (D, k) be an instance of OFVDS. We note that during the execution of our algorithm, D may become a multigraph.

Preprocessing. We start by applying the following reduction rules exhaustively, where a rule is applied only if its condition is true and the conditions of all of the preceding rules are false. Rule 4 is given in [25], and its correctness is proven in that paper. It will be clear that the other first five rules can be applied in polynomial time, while for the last rule, we call the algorithm given by Lemma 4. Moreover, it is straightforward to verify that each of these rules, except Rule 4, is safe (i.e., the instance it returns is equivalent to the input instance).

▶ **Reduction Rule 1.** *If there exists a vertex $v \in V(D)$ such that $d^+(v) = 0$ and $d^-(v) \leq 1$, remove v from D.*

▶ **Reduction Rule 2.** *If there exists a directed path $P = (w_0, w_1, \ldots, w_l, w_{l+1})$ in D such that $l \geq 2$ and for all $i \in [l]$, $d^-(w_i) = d^+(w_i) = 1$, remove each vertex in $\{w_1, \ldots, w_{l-1}\}$ from D and add the edge (w_0, w_l) to D.*

▶ **Reduction Rule 3.** *If there exists an edge $(u, v) \in E(D)$ with multiplicity at least 3, remove all but two copies of it.*

▶ **Reduction Rule 4.** *If there exist collisions $(u_1, w_1, v), \ldots, (u_{k+1}, w_{k+1}, v)$ that pairwise intersect only at v, remove v from D and decrease k by 1.*

▶ **Reduction Rule 5.** *If there exists a vertex $v \in V(D)$ such that $d^-(v) \geq k + 2$, remove v from D and decrease k by 1.*

▶ **Reduction Rule 6.** *Let G be the underlying graph of D. If there exists a vertex $v \in V(G)$ such that there is a $(k + 1)$-flower through v in G, remove v from D and decrease k by 1.*

Bounding Out-Degrees. Next, we aim to bound the maximum out-degree of a vertex in D. To this end, suppose that there exists a vertex $v \in V(D)$ with $d^+(v) \geq 16k + 1$. Let G be the underlying graph of D. Since Reduction Rule 6 is not applicable, we let Z_v be the set obtained by calling the algorithm given by Lemma 4. Moreover, we let S_{app} be a 3-factor approximate solution obtained be calling the algorithm given by Theorem 5. We can assume that $|S_{app}| \leq 3k$, since otherwise the input instance is a NO-instance. Denote $X_v = (S_{app} \cup Z_v) \setminus \{v\}$. Since $|Z_v| \leq 2k$, we have that $|X_v| \leq 5k$.

We proceed by examining the set $\mathcal{C}_v = \{C_1, C_2, \ldots, C_{|\mathcal{C}_v|}\}$ of the connected components in $G \setminus (X_v \cup \{v\})$. Since S_{app} is an approximate solution, each component $C_i \in \mathcal{C}_v$ is an out-tree. Moreover, for any component $C_i \in \mathcal{C}_v$, v has at most one neighbor in C_i, since otherwise there would have been cycle passing through v in $G \setminus Z_v$, contradicting the definition of Z_v. For each component $C_i \in \mathcal{C}_v$, let u_i be the root of C_i. Let $\mathcal{D}_v = \{C_i \mid C_i \in \mathcal{C}_v, (v, u_i) \in E(D)\}$ and $\tilde{\mathcal{D}}_v = \{C_i \mid C_i \in \mathcal{C}, (v, u) \in E(D), u \in C_i, u \neq u_i\}$. Observe that $d^+(v) \leq |\mathcal{D}_v| + |\tilde{\mathcal{D}}_v| + |X_v|$. Moreover, since Reduction Rule 4 is not applicable, $|\tilde{\mathcal{D}}_v| \leq k + 1$. Since $d^+(v) \geq 16k + 1$, we have that $|\mathcal{D}_v| \geq 10k$. Without loss of generality, let $\mathcal{D}_v = \{C_1, \ldots, C_p\}$ where $p = |\mathcal{D}_v|$. Since Reduction Rule 1 is not applicable, for any component $C_i \in \mathcal{D}_v$ there exists an edge in $E(G)$ with one endpoint in C_i and the other in X_v.

We now construct an auxiliary (undirected) bipartite graph H with bipartition (A, B), where $A = X_v$ and B is a set of new vertices denoted by $b_1, \ldots, b_p$. For any $u \in A$ and $b_i \in B$, $(u, b_i) \in E(H)$ if and only if there exists an edge in G between u and some vertex in C_i. Since $|B| \geq 2|A|$ and there are no isolated vertices in B, we can use the algorithm given by Lemma 2 to obtain nonempty vertex sets $X'_v \subseteq A$ and $Y'_v \subseteq B$ such that there is a 2-expansion of X'_v into Y'_v and $N(Y'_v) \subseteq X'_v$. Let $\mathcal{D}'_v = \{C_i \mid b_i \in Y'_v\}$.

▶ **Reduction Rule 7.** *Remove each of the edges in D between v and any vertex in a component in $\mathcal{D}'_v$. For every vertex $x_i \in X'_v$, insert two copies of the edge (v, x_i) into $E(D)$.*

▶ **Lemma 6.** *Reduction Rule 7 is safe.*

Proof. Let D' be the graph resulting from the application of the rule. We need to prove that (D, k) is a YES-instance if and only if (D', k) is a YES-instance.

Forward Direction. For the forward direction, we first claim that if (D, k) has a solution S such that $v \notin S$, then it has a solution S' such that $X'_v \subseteq S'$. To this end, suppose that (D, k) has a solution S such that $v \notin S$. Let $S' = (S \setminus \bigcup_{C_i \in \mathcal{D}'_v} V(C_i)) \cup X'_v$. It holds that $|S'| \leq |S|$ since for each $x \in X'_v \setminus S$, at least one vertex from at least one of the two components in its expansion set must belong to the solution. Suppose for the sake of contradiction that $F = D \setminus S'$ is not an out-forest. First, assume that there exists a vertex in F with in-degree at least 2. Note that $V(D) = \bigcup_{C_i \in \mathcal{C}_v} V(C_i) \cup X_v \cup \{v\}$. Recall that the neighborhood of each of the vertices in the connected components that belong to $\mathcal{D}'_v$ is contained in $\{v\} \cup X'_v$. Moreover, v only has out-neighbors in the components that belong to $\mathcal{D}'_v$ and each $C_i \in \mathcal{C}_v$ is an out-tree. Therefore, since $D \setminus S$ has no vertex of in-degree at least 2, so does $D \setminus S'$. Now, assume that there is a cycle C in F. Then, if $V(C) \cap (S \cap \bigcup_{C_i \in \mathcal{D}'} V(C_i)) = \emptyset$, then C is also a cycle in $D \setminus S$, which is a contradiction. Thus, $V(C) \cap (S \cap \bigcup_{C_i \in \mathcal{D}'} V(C)_i) \neq \emptyset$. However, any cycle that passes through a component in $\mathcal{D}'_v$ also passes through v and a vertex in X'_v. Since $X'_v \subseteq S'$, no such cycle exists. This finishes the proof of the claim.

Let S be a solution to (D, k). If $v \in S$, then it is clear that $D' \setminus S$ is an out-forest. Otherwise, if $v \notin S$, our claim implies that (D, k) has a solution S' such that $X'_v \subseteq S'$. Then, $D' \setminus S'$ is an out-forest.

Backward Direction. For the backward direction, let us prove the following claim. If (D', k) has a solution S such that $v \notin S$, then $X'_v \subseteq S$. Suppose, by way of contradiction, that the claim is incorrect. Then, there exists $x \in X'_v$ such that $x \notin S$. However, this implies that $D' \setminus S$ is not an out-forest as it contains the double edges (v, x_i).

Now, let S be a solution to (D', k), and denote $F = D' \setminus S$. Suppose $v \in S$. Then, $F = D \setminus S$ is an out forest and thus S is solution to (D, k). If $v \notin S$, then by our previous claim, $X'_v \subseteq S$. Observe that each vertex $u_i \notin S$ is a root in F. Moreover, each such vertex u_i and v belong to different out-trees of F. This implies that if we add (to D') the edges between v and each vertex u_i that have been removed by the application of the rule, F will remain an out-forest. Thus, S is a solution to (D, k). ◀

After an exhaustive application of Reduction Rule 7, the out-degree of each vertex in D is at most $16k$. However, since this rule inserts edges into $E(G)$, we need the following lemma.

▶ **Lemma 7** (*[1]). *The total number of applications of the reduction rules is bounded by a polynomial in the input size.*

Correctness. By relying on counting arguments as well as Lemmas 6 and 7, we obtain the main result of this section.

▶ **Theorem 8** (*). OFVDS *admits an $\mathcal{O}(k^2)$-kernel.*

We also prove that the size of the kernel given in Theorem 8 is tight, that is OFVDS does not admit an $\mathcal{O}(k^{2-\epsilon})$ size kernel unless coNP $\subseteq$ NP/poly. This result follows from an easy polynomial time parameter preserving transformation from the Vertex Cover problem parameterized by the solution size to OFVDS.

[1] Due to space constraints, proofs of results marked with (*) were omitted.

4 Improved Kernel for Pumpkin Vertex Deletion Set

In this section we prove the following theorem.

▶ **Theorem 9.** PVDS *admits an* $\mathcal{O}(k^3)$*-vertex kernel.*

Let (D, k) be an instance of PVDS. We assume that $|V(D)| \geq k^3$, else we are done. Let $\mathsf{HO} = \{v \in V \mid d^+(v) \geq k+2\}$ and $\mathsf{HI} = \{v \in V \mid d^-(v) \geq k+2\}$. That is, HO and HI are vertices of high out-degrees and high in-degrees, respectively. Mnich and Leeuwen [25] proved that the following reduction rule is safe.

▶ **Reduction Rule 4.1.** *If* $|\mathsf{HO}| > k+1$ *or* $|\mathsf{HI}| > k+1$*, return that* (D, k) *is a* NO*-instance.*

For the sake of clarity, we divide the presentation of the kernelization algorithm into two subsections. At the end of Section 4.1, we will simplify the instance in a way that will allow us to assume that if there is a solution S, then *both the source and sink of the pumpkin* $D \setminus S$ *belong to* $\mathsf{HO} \cup \mathsf{HI}$ (Assumption 17). This assumption will be at the heart of the "marking approach" of Section 4.2, which will handle instances which have been reduced with respect to the reduction rules in Section 4.1. An intuitive explanation of the necessity of our marking process is given at the beginning of Section 4.2. Throughout this section, if k becomes negative, we return that (D, k) is a NO-instance, and if D becomes a pumpkin and k is positive or zero, we return that (D, k) is a YES-instance.

4.1 Simplification Phase

For any $v \in V(D)$, denote by X_v the set of in-neighbors of v, that is, $X_v = N^-(v)$ and by Y_v the set of every vertex $y \in V(D)$ for which there exists a vertex $x \in X_v$ such that $(x, y) \in E(D)$. Note that X_v and Y_v may or may not be disjoint sets. We now give a construction of an auxiliary graph that will be used to prove the safeness of the upcoming reduction rule. For this, consider a set Y'_v of new vertices such that there is exactly one vertex $y' \in Y'_v$ for any $y \in Y_v$. That is, Y'_v is a set containing a copy for each of the vertex in Y_v. By construction, X_v and Y'_v are disjoint sets. Let H^-_v be the (undirected) bipartite graph on the vertex set $X_v \cup Y'_v$ where for all $x \in X_v$ and $y' \in Y'_v$, $\{x, y'\} \in E(H^-_v)$ if and only if $(x, y) \in E(D)$. Let $\mathsf{match}^-(v)$ be the size of a maximum matching in H^-_v.

▶ **Reduction Rule 4.2.** *If there exists a vertex* $v \in V(D)$ *such that* $\mathsf{match}^-(v) > 2(k+1)$*, remove* v *from* D *and decrease* k *by* 1*.*

▶ **Lemma 10.** *Reduction Rule 4.2 is safe.*

Proof. For the backward direction, trivially if S is a pumpkin deletion set in $D \setminus \{v\}$ of size at most $k-1$, then $S \cup \{v\}$ is a pumpkin deletion set in D of size at most k. For the forward direction, it is *sufficient* to show that if (D, k) is a YES-instance then *every solution* S *contains* v. For a contradiction, assume that there exists a solution S that does not contain v. Let M be a maximum matching in the graph H^-_v. Observe that for every edge $\{x, y'\} \in M$ where $x \in X_v$, if x is not the source of the pumpkin $D \setminus S$, it holds that $|S \cap \{x, y\}| \geq 1$ (otherwise the pumpkin $D \setminus S$ contains a vertex, which is not its source, and has at least two out-neighbors). Moreover, for every edge $\{x, y'\} \in M$ where $x \in X_v$, if y is the source of the pumpkin $D \setminus S$, it holds that $x \in S$. We thus deduce that *for all but one of the edges* $\{x, y'\} \in M$*, we have that* $|S \cap \{x, y\}| \geq 1$. Since M is a matching, for every vertex $u \in S$, the vertex u can belong to at most one edge in M, and the vertex u' (if it belongs to Y'_v) can also belong to at most one edge in M. However, $|S| \leq k$, and therefore

$S \cup \{y' \in Y'_v : y \in S\}$ can intersect at most $2k$ edges in M. Since $S \cup \{y' \in Y'_v : y \in S\}$ must intersect all but one edge of M and $|M| > 2(k+1)$, we obtain a contradiction. ◀

Now, to present the symmetric rule, for any vertex $v \in V(D)$, denote by X_v the set of out-neighbors of v, that is, $X_v = N^+(v)$. Let Y_v be the set of vertices $y \in V(D)$ for which there exists a vertex $x \in X_v$ such that $(y,x) \in E(D)$. Let Y'_v be a set containing a copy y' of each vertex $y \in Y$. Let H^+_v be the bipartite graph on the vertex-set $X_v \cup Y'_v$ which for all $x \in X_v$ and $y' \in Y'_v$ contains the edge $\{x, y'\}$ if and only if $(y,x) \in E(D)$. Let $\mathsf{match}^+(v)$ be the size of a maximum matching in H^+_v. Then, the following reduction rule is safe.

▶ **Reduction Rule 4.3.** *If there exists a vertex $v \in V(D)$ such that $\mathsf{match}^+(v) > 2(k+1)$, remove v from D and decrement k by 1.*

We also need the following rule, proved by Mnich and Leeuwen [25].

▶ **Reduction Rule 4.4.** *Let $P = (w_0, \cdots, w_\ell)$ be an induced directed path, that is for all $i \in [l-1]$ $d^-(w_i) = d^+(w_i) = 1$, with $\ell > k+2$ in D. Then, delete w_1 from D and add the edge (w_0, w_2).*

Consider some hypothetical solution S (if such a solution exists). Let s and t denote the source and sink, respectively, of the pumpkin $D \setminus S$. Let A (or B) denote the set of out-neighbors (resp. in-neighbors) of s (resp. t) in the pumpkin. Clearly, $|A| = |B|$. Let $C = V(D) \setminus (S \cup A \cup B \cup \{s,t\})$. Next, we prove a series of useful claims relating to S.

▶ **Lemma 11 (*).** (i) *Every vertex in $\{s\} \cup A \cup B \cup C$ has in-degree (in D) at most $k+1$, and* (ii) *every vertex in $\{t\} \cup A \cup B \cup C$ has out-degree (in D) at most $k+1$.*

▶ **Lemma 12 (*).** *For any vertex $v \in V(D)$, $|N^-(v) \cap C|, |N^+(v) \cap C| \leq 2(k+1)$.*

The set of in-neighbors (or out-neighbors) of any vertex $v \in V(D)$ is contained in $A \cup B \cup C \cup S \cup \{s,t\}$. Since $|A| \leq d^+(s)$, $|B| \leq d^-(t)$ and $|S| \leq k$, Lemma 12 gives us the following corollary.

▶ **Corollary 13.** *For any vertex $v \in V(D)$, $d^-(v), d^+(v) \leq 3k + d^+(s) + d^-(t) + 4$.*

We further strengthen this corollary to obtain the following result.

▶ **Lemma 14 (*).** *For any vertex $v \in V(D)$, $d^-(v), d^+(v) \leq \min\{4k + 2d^+(s) + 4, 4k + 2d^-(t) + 4\}$.*

Let $M = \max_{v \in V(D)}\{d^+(v), d^-(v)\}$. The next corollary (derived from Lemma 14) and rule will bring us to the main goal of this subsection, summarized in Assumption 17 below.

▶ **Corollary 15 (*).** *If $M > 6k+6$, then $s \in \mathsf{HO}$ and $t \in \mathsf{HI}$.*

▶ **Reduction Rule 4.5.** *If $|V(D)| > 2k^2M + 4kM + k + 2$, return (D,k) is a NO-instance.*

▶ **Lemma 16 (*).** *Reduction Rule 4.5 is safe.*

By Rule 4.5, if $M \leq 6k+6$, we obtain the desired kernel. Thus, by Corollary 15, we have the following observation.

▶ **Assumption 17.** *From now on, we can assume that if a solution exists, in the resulting pumpkin the source belongs to HO and the target belongs to HI.*

Next, it will be convenient to assume that HI and HO are disjoint sets. To this end, we apply the following rule exhaustively, where safeness follows directly from Lemma 11.

▶ **Reduction Rule 4.6.** *Remove all vertices in $HI \cap HO$ and decrease k by $|\mathsf{HI} \cap \mathsf{HO}|$.*

We will also assume that the following rule has been applied exhaustively. This assumption will be used at the end of the following subsection (in the proof of Lemma 25).

▶ **Reduction Rule 4.7.** *If there exists a vertex $v \notin \mathsf{HI} \cup \mathsf{HO}$ such that $N^-(v) \cap (V(D) \setminus \mathsf{HI}) = \emptyset$ or $N^+(v) \cap (V(D) \setminus \mathsf{HO}) = \emptyset$, delete v from D and decrease k by 1.*

▶ **Lemma 18 (*).** *Reduction Rule 4.7 is safe.*

4.2 Marking Approach

We are now ready to present our marking approach, handling instances to which Assumption 17 applies and none of the rules in Section 4.1 is applicable. Let $\mathcal{P}^*$ is the set of connected components in $D \setminus (\mathsf{HO} \cup \mathsf{HI})$ that are directed paths whose internal vertices have in-degree 1 and out-degree 1 in D, and let V^* be the union of the vertex-sets of the paths in $\mathcal{P}^*$. It turns out that by relying on Lemma 12 and Rule 4.4, one can directly bound the number of vertices in $V(D) \setminus V^*$ by $\mathcal{O}(k^3)$, assuming that the input instance is a YES-instance (see the proof of Lemma 23). However, bounding the size of V^* is more tricky, and our marking process is devoted to this cause. In this process, we will mark $\mathcal{O}(k^3)$ vertices from V^*, and prove that because we are handling instances to which Assumption 17 applies, all of the vertices that are not marked are essentially irrelevant. We will perform two "rounds" of marking. Roughly speaking, for each pair of vertices in HO (or HI) the first round aims to capture enough vertices of paths that describe the relation between the vertices in this pair, or, more precisely, why one of the vertices of the pair is a "better choice" than the other when one should decide which vertex (from HO) is the source of the pumpkin. However, this round is not sufficient, since some vertices in HO (or HI) have conflicts (independent of the other vertices in $\mathsf{HO} \cup \mathsf{HI}$) relating to the endpoints of the paths in $\mathcal{P}^*$. In the second round of marking, for each vertex in $\mathsf{HO} \cup \mathsf{HI}$, we mark enough vertices from these problematic paths.

First Round of Marking. Towards the performance of the first round, we need the following notations. For each vertex $v \in V(D) \setminus (\mathsf{HI} \cup \mathsf{HO})$, let $\widehat{P}(v)$ denote the connected component in $D \setminus (\mathsf{HI} \cup \mathsf{HO})$ containing v. For each $s \in \mathsf{HO}$, let $\widehat{N}(s)$ denote the set of each out-neighbor $v \in V(D) \setminus (\mathsf{HI} \cup \mathsf{HO})$ of s such that $\widehat{P}(v) \in \mathcal{P}^*$ and the first vertex of (the directed path) $\widehat{P}(v)$ is v. Symmetrically, for each $t \in \mathsf{HI}$, let $\widehat{N}(t)$ denote the set of each in-neighbor $v \in V(D) \setminus (\mathsf{HI} \cup \mathsf{HO})$ of t such that $\widehat{P}(v) \in \mathcal{P}^*$ and the last vertex of $\widehat{P}(v)$ is v. By Rule 4.6, $\mathsf{HI} \cap \mathsf{HO} = \emptyset$, and therefore these notations are well defined (i.e., we have not defined $\widehat{N}$ twice for the same vertex). Given $u \in (\mathsf{HI} \cup \mathsf{HO})$, we also denote $\widehat{\mathcal{P}}(u) = \{\widehat{P}(v) \mid v \in \widehat{N}(u)\}$. Observe that the paths in $\widehat{\mathcal{P}}(u)$ are pairwise vertex-disjoint.

Next, we identify enough vertices from paths that capture the relation between each pair of vertices in HO (or HI). For each pair $(s, s') \in \mathsf{HO} \times \mathsf{HO}$, let $\widehat{MK}_P(s, s')$ be an arbitrarily chosen set of minimal size of paths from $\widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$ that together contain at least $k + 1$ vertices not having s' as an in-neighbor. In this context, observe that only the last vertex on a path in $\widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$ can have s' as an in-neighbor. In this case, the path must contain at least two vertices (since its first vertex cannot have s' as an in-neighbor), and while we insert the entire path into $\widehat{MK}_P(s, s')$, its last vertex is not "counted" when we aim to obtain at least $k + 1$ vertices not having s' as an in-neighbor. If there are not enough paths to obtain at least $k + 1$ such vertices, let $\widehat{MK}_P(s, s') = \widehat{\mathcal{P}}(s) \setminus \widehat{\mathcal{P}}(s')$. Symmetrically, for each pair $(t, t') \in \mathsf{HI} \times \mathsf{HI}$, let $\widehat{MK}_P(t, t')$ be an arbitrarily chosen set of minimal size of paths from

$\widehat{\mathcal{P}}(t) \setminus \widehat{\mathcal{P}}(t')$ that together contain at least $k+1$ vertices not having t' as an out-neighbor. If there are not enough paths, let $\widehat{MK}_P(t,t') = \widehat{\mathcal{P}}(t) \setminus \widehat{\mathcal{P}}(t')$.

Finally, given a pair $(v,v') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \times \mathsf{HI})$, let $\widehat{MK}(v,v')$ denote the union of the vertex-sets of the paths in $\widehat{MK}_P(v,v')$. We have the following claim.

▶ **Lemma 19 (*).** *For each pair $(v,v') \in (HO \times HO) \cup (HI \times HI)$, $|\widehat{MK}(v,v')| \leq 3(k+1)$.*

Second Round of Marking. We proceed to the second round of marking. For this purpose, we need the following notation. For each $u \in \mathsf{HI} \cup \mathsf{HO}$, let $\widetilde{MK}_P(u)$ denote the set of each directed path in $\mathcal{P}^*$ whose first and last vertices are both neighbors of u.

▶ **Reduction Rule 4.8.** *If there exists $u \in HI \cup HO$ such that $|\widetilde{MK}_P(u)| > k+1$, delete u from D and decrease k by 1.*

▶ **Lemma 20 (*).** *Reduction Rule 4.8 is safe.*

For each $u \in \mathsf{HI} \cup \mathsf{HO}$, let $\widetilde{MK}(u)$ be the union of the vertex-sets of the paths in $\widetilde{MK}_P(u)$. Since at this point, Rules 4.4 and 4.8 are not applicable, we have the following lemma.

▶ **Lemma 21.** *For each $u \in HI \cup HO$, $|\widetilde{MK}(u)| \leq (k+1)(k+2)$.*

The Size of the Kernel. For the sake of abbreviation, we define the following sets.

- $MK_{\mathcal{P}} = (\bigcup_{(u,u') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \cup \mathsf{HI})} \widehat{MK}_P(u,u')) \cup (\bigcup_{u \in \mathsf{HO} \cup \mathsf{HI}} \widetilde{MK}_P(u))$, and
- $MK = (\bigcup_{(u,u') \in (\mathsf{HO} \times \mathsf{HO}) \cup (\mathsf{HI} \cup \mathsf{HI})} \widehat{MK}(u,u')) \cup (\bigcup_{u \in \mathsf{HO} \cup \mathsf{HI}} \widetilde{MK}(u))$.

By Lemmas 19 and 21, and since Rule 4.1 is not applicable, we bound $|MK|$ as follows.

▶ **Lemma 22.** $|MK| \leq 2 \cdot (3(k+1)^3 + (k+1)^2(k+2)) \leq 8(k+2)^3$.

Let V^R denote the set of unmarked vertices in V^*, i.e., $V^* \setminus MK$. We construct the graph D' by removing from D all of the vertices in V^R, adding a set N_{k+2} of $k+2$ new vertices, and for each of the new vertices, adding an edge from each vertex in HO as well as an edge to each vertex in HI. If $V(D')$ contains at most $2k+4$ vertices, add to it one-by-one a vertex-set of a path in $\mathcal{P}^*$ until its size becomes at least $2k+5$ (by Lemma 4.4, the size will not exceed $3k+6$, and because $|V(D)| \geq k^3$, we will reach the desired size).

▶ **Lemma 23 (*).** *If $|V(D')| > 30(k+2)^3$, (D',k) is a NO-instance of* PVDS.

Correctness. Finally, Theorem 9 follows from Lemma 23 and the two lemmas below.

▶ **Lemma 24 (*).** *If (D,k) is a YES-instance then (D',k) is a YES-instance.*

▶ **Lemma 25.** *If (D',k) is a YES-instance then (D,k) is a YES-instance.*

Proof. Let S be a solution to (D',k). Let s and t be the source and target, respectively, of the pumpkin $D' \setminus S$. Because of the set N_{k+2} of $k+2$ vertices added to D' at its construction, and since $|S| \leq k$, $s \in \mathsf{HO}$ and $t \in \mathsf{HI}$. Moreover, by the definition of HO and HI, $(\mathsf{HO} \cup \mathsf{HI}) \setminus \{s,t\} \subseteq S$. We can also assume that S does not contain any vertex added to D' at its construction since by removing such a vertex from S, we still have a pumpkin. Our goal will be to show that S is also a solution to (D,k), which will imply that the lemma is correct. To this end, we will show that $D \setminus S$ is a pumpkin with source s and sink t.

First, note that we can assume that in $D \setminus S$ there exists a path from s to t. Indeed, if this is not true, then $D' \setminus S$ consists only of s, t and newly added vertices. That is, $V(D')$

contains at most $2k+4$ vertices, which contradicts its construction. By the definition of $\mathcal{P}^*$, each path in $\mathcal{P}^*$ has only internal vertices that have in-degree 1 and out-degree 1 in D , and its endpoints can only be adjacent to vertices in $\mathsf{HI} \cup \mathsf{HO}$ and in the path itself. Thus, to prove the lemma, it is sufficient to show that for each path in $\mathcal{P}^* \setminus MK_{\mathcal{P}}$, its first vertex has s as an ingoing neighbor, its last vertex has t as an out-neighbor, and if it contains at least two vertices, its first vertex is not a neighbor of t and its last vertex is not a neighbor of s.

Consider some path $P \in \mathcal{P}^* \setminus MK_{\mathcal{P}}$. First suppose, by way of contradiction, that the first vertex v of P does not have s as an in-neighbor. Because Rule 4.7 is not applicable, v has at least one in-neighbor $s' \in \mathsf{HO}$. Thus, since $v \notin MK$, $MK(s', s)$ contains at least $k+1$ vertices that are not out-neighbors of s and such that each of them belongs to a path in $\mathcal{P}^*$ whose first vertex is not an out-neighbor of s. The vertices in $MK(s', s)$ belong to D'. Since $|S| \leq k$, at least one of these vertices, say some u, should belong to the pumpkin $D' \setminus S$. However, in $D' \setminus ((\mathsf{HI} \cup \mathsf{HO}) \setminus \{s\})$, which is a supergraph of $D' \setminus S$, u cannot be reached from s, which contradicts the fact that $D' \setminus S$ is a pumpkin. Symmetrically, it is shown that the last vertex of P has t as an out-neighbor.

Now assume that P contains at least two vertices. Suppose, by way of contradiction, that the first vertex of P has t as a neighbor. We have already shown that the last vertex of P is also a neighbor of t, and therefore $P \in \widetilde{MK}_P(t)$. However, $\widetilde{MK}_P(t) \subseteq MK_{\mathcal{P}}$, which contradicts the fact that $P \in \mathcal{P}^* \setminus MK_{\mathcal{P}}$. Symmetrically, it is shown that the last vertex of P does not have s as a neighbor, concluding the proof of the lemma. ◀

References

1 F. N. Abu-Khzam. A kernelization algorithm for d-Hitting Set. *JCSS*, 76(7):524–531, 2010.

2 V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIDMA*, 12(3):289–297, 1999.

3 J. Bang-Jensen, A. Maddaloni, and S. Saurabh. Algorithms and kernels for feedback set problems in generalizations of tournaments. *Algorithmica*, pages 1–24, 2015.

4 R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SICOMP*, 27(4):942–959, 1998.

5 A. Becker and D. Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artif. Intell.*, 83(1):167–188, 1996.

6 Y. Cao, J. Chen, and Y. Liu. On feedback vertex set new measure and new structures. In *SWAT*, pages 93–104, 2010.

7 C. Chekuri and V. Madan. Constant factor approximation for subset feedback problems via a new LP relaxation. In *SODA*, pages 808–820, 2016.

8 J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *JCSS*, 74(7):1188–1198, 2008.

9 J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008.

10 R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx. Directed subset feedback vertex set is fixed-parameter tractable. *TALG*, 11(4):28, 2015.

11 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

12 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.

13 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIDMA*, 27(1):290–309, 2013. doi:10.1137/110843071.

14 R. Diestel. *Graph Theory, 4th Edition.* Springer, 2012.

15 M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *JDA*, 8(1):76–86, 2010. doi:10.1016/j.jda.2009.08.001.

16 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity.* Springer, 2013.

17 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canad. J. Math*, 17:347–352, 1965.

18 G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

19 V. Guruswami and E. Lee. Inapproximability of H-transversal/packing. In *APPROX/RANDOM*, pages 284–304, 2015.

20 N. Kakimura, K. Kawarabayashi, and Y. Kobayashi. Erdös-Pósa property and its algorithmic applications: parity constraints, subset feedback set, and subset packing. In *SODA*, pages 1726–1736, 2012.

21 N. Kakimura, K. Kawarabayashi, and D. Marx. Packing cycles through prescribed vertices. *J. Comb. Theory, Ser. B*, 101(5):378–381, 2011.

22 K. Kawarabayashi and Y. Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the S-cycle packing problem. *J. Comb. Theory, Ser. B*, 102(4):1020–1034, 2012. doi:10.1016/j.jctb.2011.12.001.

23 K. Kawarabayashi, D. Král, M. Krcál, and S. Kreutzer. Packing directed cycles through a specified vertex set. In *SODA*, pages 365–377, 2013.

24 T. Kociumaka and M. Pilipczuk. Faster deterministic feedback vertex set. *IPL*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

25 M. Mnich and E. J. van Leeuwen. Polynomial kernels for deletion to classes of acyclic digraphs. In *STACS*, pages 1–13, 2016.

26 M. Pontecorvi and P. Wollan. Disjoint cycles intersecting a set of vertices. *J. Comb. Theory, Ser. B*, 102(5):1134–1141, 2012.

27 V. Raman, S. Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *TALG*, 2(3):403–415, 2006. doi:10.1145/1159892.1159898.

28 B. A. Reed, N. Robertson, P. D. Seymour, and R. Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

29 P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

30 P. D. Seymour. Packing circuits in eulerian digraphs. *Combinatorica*, 16(2):223–231, 1996.

31 S. Thomassé. A $4k^2$ kernel for feedback vertex set. *TALG*, 6(2), 2010.

32 M. Wahlström. Half-integrality, LP-branching and FPT algorithms. In *SODA*, pages 1762–1781, 2014.

An Efficient Algorithm for Placing Electric Vehicle Charging Stations[*]

Pankaj K. Agarwal[1], Jiangwei Pan[2], and Will Victor[3]

1 **Duke University, Durham, USA**
pankaj@cs.duke.edu

2 **Duke University, Durham, USA**
jwpan@cs.duke.edu

3 **Duke University, Durham, USA**
william.victor@duke.edu

Abstract

Motivated by the increasing popularity of electric vehicles (EV) and a lack of charging stations in the road network, we study the shortest path hitting set (SPHS) problem. Roughly speaking, given an input graph G, the goal is to compute a small-size subset H of vertices of G such that by placing charging stations at vertices in H, every shortest path in G becomes *EV-feasible*, i.e., an EV can travel between any two vertices of G through the shortest path with a full charge. In this paper, we propose a bi-criteria approximation algorithm with running time *near-linear* in the size of G that has a logarithmic approximation on $|H|$ and may require the EV to slightly deviate from the shortest path. We also present a data structure for computing an EV-feasible path between two query vertices of G.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Computations on Discrete Structures

Keywords and phrases Shortest path hitting set, Charging station placement, Electric vehicle

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.7

1 Introduction

Motivation. Electric vehicles (EVs) are becoming increasingly popular as we transition from fossil fuels to cleaner energy. One of the main challenges in the popularization of EVs is the lack of charging facilities in the road network. Ideally, one should be able to reach a charging station quickly anywhere on the road network, as in the case of gas stations. However, due to resource constraints and the relatively small fraction of EVs currently on the road, it is desirable to first build a small number of charging stations to satisfy the most basic transportation needs of EV owners. One natural such need is that an EV, with an initial full charge, should be able to travel between any two locations via the shortest path without draining the battery. In other words, any shortest path in the road network contains sufficient number of charging stations. We study the problem of placing the minimum number of charging stations to satisfy the above condition.

[*] Work on this paper is supported by NSF under grants CCF-11-61359, IIS-14-08846, CCF-15-13816, and ISS-14-47554 by an ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 7; pp. 7:1–7:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

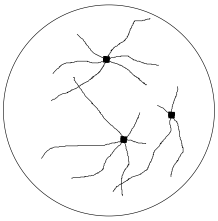


Figure 1 Illustration of the definition of highway dimension: a ball of radius r, and three vertices that intersect all shortest paths within the ball whose lengths are $\Omega(r)$.

Problem statement. The input consists of a graph $G = (V, E)$ of $n = |V|$ vertices and $m = |E|$ edges, which represents a road network, and a positive length function $\ell(\cdot)$ on the edges in E. We assume that an EV can travel a fixed distance r (e.g., 200km) in G with a full charge. More sophisticated models have been proposed for the battery capacity, which not only consider the distance but also the topography of the underlying terrain. But we use this simpler model because the problem is challenging even in this model and furthermore, on realistic terrains the EV will travel distance in the range $[\frac{r}{c}, cr]$, for some small constant $c \geq 1$, with a full charge. For any two vertices $u, v \in V$, let $\pi_G(u, v)$ denote the shortest path from u to v in graph G; it is abbreviated $\pi(u, v)$ when there is no ambiguity. For convenience, we set $\mu(u, v) := \ell(\pi(u, v))$. For a subset $X \subseteq V$ and a vertex $v \in V$, let $\mu(v, X) := \min_{x \in X} \mu(v, x)$.

Given a set X of vertices, a path P is said to be *hit* by X if X contains an *interior* vertex of P — a vertex of P other than its starting and ending vertices. We say a path P is *r-EV-feasible* with respect to X (charging stations) if every contiguous subpath of P of length more than r is hit by X. An *r-shortest-path hitting set* (r-SPHS) of G is a subset $H \subseteq V$ such that for all $u, v \in V$, $\pi(u, v)$ is r-EV-feasible with respect to H. Similarly, given $\delta \in (0, 1)$, a *δ-approximate* r-SPHS of G is a subset $\tilde{H} \subseteq V$ such that for all $u, v \in V$, there exists an r-EV-feasible path P (with respect to $\tilde{H}$) between u, v with $\ell(P) \leq (1 + \delta)\mu(u, v)$. The goal of the *shortest-path hitting-set (SPHS) problem* is to compute a minimum-size r-SPHS of G. In the rest of the paper, for simplicity, we may leave out parameter r and just write SPHS and EV-feasible.

The problem of computing minimum number of charging stations reduces to an instance of the classical hitting-set problem, and is NP-complete by a simple reduction from the vertex-cover problem. Since we are not aware of a proof of the NP-completeness in the literature, we describe the details of the reduction in Section 2. We propose an efficient approximation algorithm for the SPHS problem that exploits the structure of road networks.

Related work. In the last few years, there has been extensive work on a variety of optimization problems on road networks, which are modeled as “sparse” graphs with additional structural properties. In particular, Abraham *et al.* [3, 1, 2] introduced the notion of *highway dimension* to give provable guarantees of efficiency for many popular shortest-path heuristics, such as reach, contraction hierarchies, and transit node; see also [6]. Roughly speaking, the graph G has highway dimension h if, for any $x > 0$ and any vertex $v \in V$, there exist h vertices that intersect all shortest paths of length at least x that are within $O(x)$ distance of v. See Figure 1 for illustration and Section 2 for the definition. Abraham *et al.* argued that real-world road networks have small highway dimension.

Storandt and Funke [21] formulated the problem of placing minimum number of charging stations such that there exists some EV-feasible path between any two vertices. They gave a polynomial-time algorithm that achieves $O(\log n)$ approximation. However, the EV-feasible path computed by their algorithm can be much longer than the shortest path. This drawback was addressed by Funke *et al.* [13]. They require the *shortest path* between any two vertices to be EV-feasible. They modeled the problem as a hitting-set problem (defined in Section 2), and obtained an $O(\log n)$ approximation using a greedy algorithm. Constructing the hitting set instance requires computing as many as $O(n^2)$ shortest paths, and can take $O(n^3)$ time in the worst case, which is formidable when the road network is large. Funke *et al.* [13] applied a number of techniques to speed up the computation, but without provable guarantees of the running time and approximation.

Several variants of the SPHS problem have been studied. For example, the road network may be small so that one can always drive from one location to another without recharging. In these cases, the charging stations are placed to satisfy other constraints. For example, Xiong *et al.* [26] take EV drivers' behavior into consideration and compute a set of charging stations in Singapore that optimizes the equilibrium utility of a congestion game. There are other optimization criteria considered in the literature, such as charging demand coverage [11] and EV access cost [22].

Another set of literature study the EV routing problem. Baum *et al.* [7] gave an algorithm that plans routes minimizing overall trip time, including time for necessary rechargings on the way. Their model allows the charging time to be a function of the remaining battery level. Goodrich and Pszona [14] formulated a bi-criteria path optimization problem, where two objectives (e.g., travel time and energy cost) are optimized, and their algorithm outputs a path that optimizes one objective before reaching some vertex and switches to the other objective afterwards. See also [8, 19] for work on computing energy-efficient paths.

As discussed above, the SPHS problem is an instance of the hitting-set problem, one of the twenty-one problems in Karp's original list of NP-complete problems [17]. The natural greedy algorithm that chooses the element that hits the most remaining sets gives an $O(\log n)$ approximation [9], which is optimal up to $o(1)$ factor unless P=NP [10]. For geometric instances, however, where the input consists of points and shapes (e.g., disks, rectangles), better approximation guarantees can be obtained. For example, a PTAS exists when the shapes are half-spaces in $\mathbb{R}^3$ [18] and $O(\log\log \textsc{opt})$ approximation can be obtained when the shapes are axis-parallel rectangles [5]. Recently, Agarwal and Pan [4] gave near-linear-time approximation algorithms for computing hitting set and set cover of many geometric instances. The hitting-set problem has also been used to compute a subset of vertices that intersect every path [12] or every shortest path [23] that contains at least k vertices.

Our result. We present a bi-criteria approximation algorithm for the SPHS problem by allowing an EV to slightly deviate from the shortest path. Our result is summarized as follows.

▶ **Theorem 1.** *Let $G = (V, E)$, $\ell : V \to \mathbb{R}^+$ be a weighted graph of constant highway dimension h, with $|V| = n$ and $|E| = O(|V|)$. Let $r > 0$ and $\delta \in [\frac{10\alpha}{r}, \frac{2}{15}]$ be two parameters where $\alpha = \max_{e\in E} \ell(e)$, and let κ be the size of a smallest r-SPHS of G. A δ-approximate r-SPHS $\tilde{H} \subseteq V$ of size $O(\kappa \log \kappa)$ can be computed in randomized expected time $O(c_\delta n \log^2 n \log \kappa)$, where $c_\delta = h^{-\log_2 \delta}$.*

In this paper, we assume $10\alpha/r \le \delta \le 2/15$, where the constants 10 and 2/15 are chosen for convenience of the analysis. Since G represents a road network, the length of a road edge in E is much smaller than the range of an average EV. Hence, $10\alpha/r \ll 1$ and δ can be set to a small constant under the assumption. We also assume $|E| = O(|V|)$ since G represents a road network and the average degree of a vertex is usually small.

At a high level, we improve the running time from $O(n^3)$ to near linear by relaxing the shortest-path requirement slightly. The algorithm works in two stages. The first stage computes a small set C of "center" vertices such that there exists a path between any pair of vertices in G that is not much longer than the shortest path and can be decomposed into shortest paths between center vertices, called *critical paths*. Furthermore, C is a δ-approximate r-SPHS, but the size of C may be much larger than κ. The second stage chooses a smaller δ-approximate r-SPHS. In particular, it computes a small-size hitting set for the critical paths. With the assumption that G has constant highway dimension, we show that the number of center vertices is small, and the optimal hitting set for the critical paths has similar size as the optimal SPHS. The algorithm uses the framework in [4], together with the dynamic trees [20] data structure, to efficiently compute a hitting set for critical paths.

Finally, we describe a data structure for the *feasible path* queries that, given two query vertices u, v, computes in $O(\kappa \log^2 \kappa)$ time the sequence of charging stations on an r-EV-feasible path P between u, v with $\ell(P) \le (1+\delta)\mu(u,v)$. The actual path in G can be recovered by performing shortest-path queries in G between adjacent charging stations. Since the highway dimension of G is bounded, each shortest-path query can be answered quickly [3].

2 Preliminaries

In this section, we define several concepts that are used by our algorithm, including the highway dimension and doubling dimension of a graph, and the hitting set and ε-net of a range space. We also describe a proof of the NP-completeness of the SPHS problem.

Given $x > 0$ and a vertex $u \in V$, let $\mathcal{B}(u,x) = \{v \in V \mid \mu(u,v) \le x\}$ be the *ball* of radius x centered at u under the shortest path metric on G.

▶ **Definition 2.** The *highway dimension* of a graph $G = (V, E)$ is the smallest integer h that satisfies the following condition: for all $x > 0$ and $u \in V$, there exists a set $S \subseteq \mathcal{B}(u, 6x)$ of at most h vertices that contains a vertex from every shortest path inside $\mathcal{B}(u, 6x)$ of length more than x.[1]

A metric space has *doubling dimension* d if any ball of radius x is contained in the union of at most 2^d balls of radius $x/2$. We will always use d to denote the doubling dimension of the shortest path metric of G and h to denote the highway dimension of G. Lemma 3 relates these two quantities.

▶ **Lemma 3** ([2]). *$d \le \log_2(h+1)$.*

Let $\Sigma = (X, \mathcal{R})$ be a finite range space where X is a finite set of elements and $\mathcal{R}$ is a family of subsets of X called *ranges*. A subset $H \subseteq X$ is called a *hitting set* of Σ if H intersects every range in $\mathcal{R}$. Given a parameter $\varepsilon \in (0, 1]$ and a weight function $w(\cdot)$ on elements of X, an *ε-net* for Σ is a subset $N \subseteq X$ that intersects every *ε-heavy* range, i.e., every range that has weight at least $\varepsilon w(X)$.

The *VC-dimension* [24] of a range space $\Sigma = (X, \mathcal{R})$ is the largest positive integer b satisfying the following condition: there exists a subset $Y \subseteq X$ with $|Y| = b$ such that $|\{S \cap Y \mid S \in \mathcal{R}\}| = 2^b$. The following ε-net theorem was proved in [16] (see also [15]).

[1] We remark that the original paper [3] that introduces highway dimension uses a constant 4 as the multiplier of the radius of the ball, but leaves open the possibility of larger constants (with adjusted constants in other bounds). We use a larger constant 6 for convenience of our analysis.

▶ **Lemma 4** ([16]). *Given a range space $(X, \mathcal{R})$ of VC-dimension β and parameters $\varepsilon, \phi \in (0,1)$, a set of $O(\frac{\beta}{\varepsilon}(\log \frac{1}{\varepsilon} + \log \frac{1}{\phi})$ independent random samples of X is an ε-net of $(X, \mathcal{R})$ with probability at least $1 - \phi$.*

In this paper, we will be interested in range spaces $\Sigma_G = (V, \mathcal{R})$ where each range in $\mathcal{R}$ corresponds to the vertices on a shortest path in G. Abraham *et al.* [1] showed that the VC-dimension of Σ_G is two when $\mathcal{R}$ contains all shortest paths in G. By the definition of VC-dimension, it is easy to check that the VC-dimension of Σ_G is no more than two when $\mathcal{R}$ contains a subset of all shortest paths in G. It is summarized in the following lemma.

▶ **Lemma 5** ([1]). *The VC-dimension of Σ_G is at most two.*

The decision version of the SPHS problem is as follows: given a graph G, a parameter $r > 0$ and an integer k, determine whether there exists an r-SPHS of G of size at most k.

▶ **Theorem 6.** *The SPHS problem is NP-complete.*

Proof. We reduce the vertex-cover problem to the SPHS problem. Recall that given a graph $G_1 = (V_1, E_1)$, a subset $A \subseteq V_1$ is a vertex cover if $\{u, v\} \cap A \neq \emptyset$ for all $(u, v) \in E_1$. We construct another undirected graph $G_2 = (V_2, E_2)$, where $V_2 = V_1 \cup \{u_e, v_e \mid e = (u, v) \in E_1\}$ and $E_2 = E_1 \cup \{(u, u_e), (v, v_e) \mid e = (u, v) \in E_1\}$, and $\ell(e) = 1\ \forall e \in E_2$. We claim that a vertex cover in G_1 corresponds to a 2-SPHS of G_2. Suppose $S_1 \subseteq V_1$ is a vertex cover for G_1. Then S_1 must be a 2-SPHS of G_2 because every shortest path of length more than 2 in G_2 must contain at least one edge from E_1 in its interior. On the other hand, suppose $S_2 \subseteq V_2$ is a 2-SPHS of G_2. Then every edge $e = (u, v) \in E_1$ is covered by S_2 because u, v are the only interior vertices of the shortest path from u_e to v_e, and one of them must be in S_2. The claim is proved.

Finally, the SPHS problem is in NP because one can verify whether a given set of vertices hits every shortest path of a graph of length more than r in polynomial time. ◀

3 The algorithm

In this section, we describe a bi-criteria approximation algorithm for the SPHS problem, whose worst-case running time is near-linear in the size of the input graph.

Let $\delta \in [10\alpha/r, 2/15]$ be a parameter. We assume that the highway dimension h and the doubling dimension d are constants. We first give a high level overview of the algorithm, which consists of three main steps.

(i) Compute a set $C \subseteq V$ of "center" vertices of size $O(\kappa/\delta^d)$, such that every vertex of V is within distance $O(\delta r)$ from some center in C.

(ii) Construct a set of shortest paths, called *critical paths*, between center vertices of length roughly $r/2$, such that between every pair of vertices in V, an approximately shortest path can be constructed by concatenating critical paths.

(iii) Compute hitting set $\tilde{H}$ for critical paths, and return $\tilde{H}$.

Next, we describe the details of each step in the following subsections.

3.1 Computing centers

We compute the set C of center vertices using a greedy algorithm, which was originally proposed for the k-center problem (i.e., find k vertices so that the distance to the farthest vertex from them is as small as possible). Initially, add an arbitrary vertex c_1 to C; in the

i-th iteration, add to C the vertex c_i that is farthest from C. The algorithm terminates when $\mu(v, C) \le \delta r/8$ for all $v \in V$.

For $i \ge 1$, let C_i be the set of chosen vertices after i iterations.

▶ **Lemma 7.** *During the entire algorithm, for any pair $c_i \ne c_j \in C$, $\mu(c_i, c_j) \ge \delta r/8$.*

Proof. Suppose there exist two centers $c_i, c_j \in C$ with $i < j$ such that $\mu(c_i, c_j) < \delta r/8$. Then $\mu(c_j, C_{j-1}) < \delta r/8$, which means the algorithm terminates before adding c_j to C. ◀

The next lemma upper bounds the number of center vertices added to C.

▶ **Lemma 8.** *$|C| = O(\kappa/\delta^d)$.*

Proof. Let H^* denote the optimal r-SPHS of size κ. Then by the definition of r-SPHS, $\mu(v, H^*) \le r$ for all $v \in V$ because otherwise a shortest path with v as an endpoint is not r-EV feasible. By the same analysis of the greedy algorithm for the k-center problem [25], we can claim that for all $v \in V$, $\mu(v, C_\kappa) \le 2r$. In other words, $V \subseteq \bigcup_{c \in C_\kappa} \mathcal{B}(c, 2r)$. Recall that the doubling dimension of the shortest path metric of G is d. By definition, a radius-$2r$ ball can be covered by $O(\delta^{-d})$ balls of radius $\delta r/16$. Thus, V can be covered by $x = O(\kappa\delta^{-d})$ balls of radius $\delta r/16$. Again by the property of the k-center greedy algorithm, adding x centers greedily to C can guarantee that every vertex of V is within distance $2 \times (\delta r/16) = \delta r/8$ of some center in C. ◀

The greedy algorithm can be implemented efficiently, as follows. Let D denote the diameter of G; then $D \le \alpha n < n\delta r$. We maintain the distance from each vertex of V to C in a priority queue; initially, the distance is ∞ as $C = \emptyset$. Suppose the shortest path distance from c_i to C is x_i when c_i is added to C. To find c_{i+1}, we compute the shortest path tree rooted at c_i that contains vertices of V whose distances to c_i are less than x_i, and updates the priority queue if the distance from some vertex v to C is decreased because of c_i. We then choose the first vertex of the priority queue (farthest from C) to be c_{i+1}.

▶ **Lemma 9.** *The greedy algorithm for computing the set C of centers takes $O(n \log^2 n + m \log n)$ time.*

Proof. To analyze the running time, we divide the above implementation into $O(\log \frac{D}{\delta r})$ phases. In phase j, the farthest distance from a vertex to C lies in $(\frac{D}{2^j}, \frac{D}{2^{j-1}}]$. If a vertex v is traversed when computing the shortest path tree rooted at a center c, then $\mu(v, c) \le D/2^{j-1}$. On the other hand, any two centers chosen in phase j have distance more than $\frac{D}{2^j}$. So there can be at most $2^d = O(1)$ centers that traverse v when computing shortest path tree in phase j. Similarly, each edge is also traversed $O(1)$ times in phase j. It takes $O(\log n)$ time to traverse a vertex and $O(1)$ time to traverse an edge in Dijkstra's algorithm and $O(\log n)$ time to update the priority queue. Therefore, the running time is $O((m + n \log n) \log \frac{D}{\delta r}) = O(n \log^2 n + m \log n)$. ◀

We remark that the set C is a δ-approximate r-SPHS. However, the size of C, $O(\kappa/\delta^d)$, can be very large when δ is small. Our algorithm computes a solution of size $O(\kappa \log \kappa)$.

3.2 Computing critical paths

For each vertex $c \in C$, we construct a shortest path tree T_c, called a *center tree*, rooted at c with radius $r/2$, i.e., T_c contains all vertices of V that are no more than $r/2$ away from c. For every $c' \in C$ with $\mu(c, c') \in [(1-\delta)\frac{r}{2} - \alpha, \frac{r}{2}]$, we add the shortest path $\pi(c, c')$ as a critical path.

▶ **Lemma 10.** *The number of critical paths is $O(\frac{\kappa}{\delta^{2d}})$, and they can be computed in $O(\frac{1}{\delta^d}(m+n\log n))$ time.*

Proof. Consider any center $c \in C$. We bound the number of critical paths that has c as one endpoint. By construction, if there is a critical path between c and some $c' \in C$, then $\mu(c,c') \le r/2$, i.e., $c' \in \mathcal{B} = \mathcal{B}(c, r/2)$. By definition of doubling dimension, $\mathcal{B}$ can be covered by $O(1/\delta^d)$ smaller balls of radius $\delta r/16$. By Lemma 7, there can be at most one center inside each smaller ball, so, there are $O(1/\delta^d)$ centers in $\mathcal{B}$. The bound on the number of critical paths follows. Similarly, a vertex $v \in V$ or an edge $e \in E$ is traversed during the construction of $O(1/\delta^d)$ center trees. Summing over all center trees, the total time is $O(\frac{1}{\delta^d}(m+n\log n))$. ◀

3.3 Computing approximate hitting set

We compute an approximate hitting set of the critical paths using an algorithm framework by Agarwal and Pan [4]: Let $\mathcal{R}$ denote the set of ranges induced by the critical paths, i.e., each range in $\mathcal{R}$ corresponds to the set of interior vertices of a critical path. Let $\mathcal{C} = (V, \mathcal{R})$ be the resulting range space. By Lemma 5, $\mathcal{C}$ has VC-dimension 2. Let λ denote the size of the optimal hitting set of $\mathcal{C}$. We guess an integer $\tilde{\lambda}$ via binary search such that $\lambda \le \tilde{\lambda} < 2\lambda$.

At a high level, the algorithm works in three stages: the *preprocessing* stage removes some vertices and ranges such that no remaining range contains too many vertices; the *weight-assignment* stage assigns a non-negative weight to each vertex so that every range in $\mathcal{R}$ is $(1/2\tilde{\lambda}e)$-heavy; and the *net-construction* stage computes an $(1/2\tilde{\lambda}e)$-net of $\mathcal{C}$. Since every range in $\mathcal{R}$ is $(1/2\tilde{\lambda}e)$-heavy, the third stage computes a hitting set of $\mathcal{C}$.

Preprocessing stage. In this stage, we compute a $\frac{1}{\tilde{\lambda}}$-net H_0 of $(V, \mathcal{R})$ with uniform weights on V, and include H_0 in the final hitting set. We then (conceptually) remove H_0 and all ranges in $\mathcal{R}$ hit by H_0 from consideration. By definition of ε-net, no remaining range in $\mathcal{R}$ contains more than $n/\tilde{\lambda}$ vertices. This property ensures that the weight-assignment stage has small running time. A simple ε-net construction algorithm is described in the net-construction stage. To remove ranges of $\mathcal{R}$ hit by H_0, we traverse all the center trees and mark every critial path hit by H_0, which takes $O(\sum_{c \in C} |T_c|) = O(n/\delta^d)$ time.

Weight-assignment stage. Recall that given a weight function $w : V \to \mathbb{R}_{\ge 0}$, a range $R \in \mathcal{R}$ is called *ε-heavy* if $w(R) \ge \varepsilon w(V)$; otherwise, R is *ε-light*. The algorithm assigns the weights in $O(\log(n/\tilde{\lambda}))$ rounds. Initially, the $w(v) = 1$ for all $v \in V$.

In each round, the algorithm processes every range $R \in \mathcal{R}$ one by one. If R is $\frac{1}{2\tilde{\lambda}}$-light, it doubles the weights of all vertices in R, the so-called *weight-doubling* step, repeatedly until R becomes $\frac{1}{2\tilde{\lambda}}$-heavy. Once R becomes $\frac{1}{2\tilde{\lambda}}$-heavy, it is not processed again in the current round, even though it may become $\frac{1}{2\tilde{\lambda}}$-light again later in the current round while $w(V)$ increases. If $2\tilde{\lambda}$ weight-doubling steps have been performed in the current round, the algorithm aborts the current round and moves to the next round. On the other hand, if all ranges have been processed with less than $2\tilde{\lambda}$ weight-doubling steps, the algorithm terminates.

The argument in [4] shows that if $\tilde{\lambda} \ge \lambda$, the algorithm always terminates with all ranges being ε-heavy with $\varepsilon = \frac{1}{2\tilde{\lambda}e}$. If the algorithm terminates and some ranges are still ε-light, we double the value of $\tilde{\lambda}$ and repeat the algorithm. The data structure described below will be used to compute the current weight of a range and to double it efficiently, the only two nontrivial steps in this stage.

Net-construction stage. The algorithm returns a ε-net, for $\varepsilon = \frac{1}{2\tilde{\lambda}e}$, of $\mathcal{C}$ as a hitting set of $\mathcal{C}$. By Lemma 4, a natural algorithm for computing an ε-net of $(V, \mathcal{R})$ is to draw $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ random samples from V, with respect to the final weights on the vertices in V. We then verify whether the set of samples is an ε-net of $\mathcal{C}$: traverse all the center trees and check whether each ε-heavy critical path is hit. This takes $O(\sum_{c\in C}|T_c|) = O(n/\delta^d)$ time. If the samples do not form an ε-net, we repeat the above steps. In expectation, $O(1)$ repetitions are required. Therefore, an ε-net of the range space $\mathcal{C}$ of size $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ can be computed in $O(\frac{n}{\delta^d} + \frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ expected time.

Data structure. We maintain all the center trees and the weights of vertices in these trees using the *dynamic trees* data structure [20]. The data structure was proposed to maintain a forest of rooted trees where each tree vertex has an arbitrary number of unordered child vertices and the vertices have weights. The main operations supported include:

- *root*(v): Return the root of the tree containing vertex v.
- *link*(v, u): Make vertex v a new child of vertex u by adding edge (v, u). This assumes v, u are in different trees and v is the root of its tree.
- *cut*(v): Delete the edge between vertex v and its parent.
- *path-aggregate*(v): Return an aggregate, such as max/min/sum, of the weights of vertices on the path from v to *root*(v).
- *update*(v, x): Add x to the weight of each vertex on the path from v to *root*(v).

Each of the above operation takes $O(\log\sum_{c\in C}|T_c|) = O(\log n)$ time [20]. In our case, the structure of the center trees remain the same, so we do not use the *link, cut* operations.

We *retrieve the weight* of a critical path using the *path-aggregate* operation, which is the sum of weights of the vertices along a path from some center vertex c to *root*(c). We *double the weight* of an individual vertex v by running *update*$(v, w(v))$ and *update*$(parent(v), -w(v))$. Note that a vertex v can appear in as many as $O(1/\delta^d)$ center trees. Thus, when we update the weight of a vertex v, we make the update for all copies of v in $O(1/\delta^d)$ center trees.

The results of computing an approximate hitting set of $\mathcal{C}$ is summarized as follows.

▶ **Lemma 11.** *A hitting set of $\mathcal{C}$ of size $O(\lambda\log\lambda)$ can be computed in $O((\frac{1}{\delta^d}n\log^2 n + \lambda\log\lambda)\log\lambda)$ expected time, where λ is the size of the optiml hitting set of $\mathcal{C}$.*

Proof. The size of the hitting set is equal to the size of the $\frac{1}{2\tilde{\lambda}e}$-net computed in the net-construction stage of the algorithm, which is $O(\tilde{\lambda}\log\tilde{\lambda}) = O(\lambda\log\lambda)$. The preprocessing and the net-construction stages both involve computing an ε-net, and take time $O(n/\delta^d + \lambda\log\lambda)$. In each round of the weight-assignment stage, retrieving the weights of the ranges in $\mathcal{R}$ takes $O(\frac{\kappa}{\delta^d}\log n) = (\frac{n}{\delta^d}\log n)$ time. There are at most $2\tilde{\lambda}$ weight-doubling steps, and each weight-doubling step updates the weights of no more than $n/\tilde{\lambda}$ vertices. Therefore, the weight-doubling steps take $O(\frac{n}{\delta^d}\log n)$ time in each round. With $O(\log n)$ rounds in the weight-assignment stage and $O(\log\lambda)$ itertions of guessing $\tilde{\lambda}$, the total running time of the algorithm is $O((\frac{1}{\delta^d}n\log^2 n + \lambda\log\lambda)\log\lambda)$. ◀

4 Analysis

We now analyze the performance of our algorithm.

▶ **Lemma 12.** $\lambda = O(h\kappa)$.

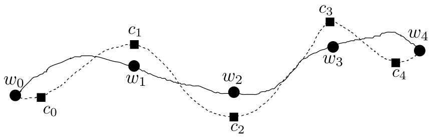


Figure 2 Construction of EV-feasible path $\tilde{P}$ (dashed curve) between w_0 and w_4. The solid curve denotes the shortest path.

Proof. Let H^* denote the optimal r-SPHS of G. By definition, H^* must hit all shortest paths that are longer than r. On the other hand, the critical paths constructed by our algorithm have lengths no more than $r/2$. Let P be a critical path between a pair of vertices u, v. Then there is a vertex $w \in H^*$ with $\mu(u, w) \leq r$. So $P \subseteq \mathcal{B}(w, 3r/2)$. In other words, each critical path is contained in the ball of radius $3r/2$ centered at some vertex in H^*. By definition of highway dimension, for any $w \in H^*$, there exists a subset S of at most h vertices in $\mathcal{B}(w, 3r/2)$ that intersect every shortest path of length more than $r/4$ contained in $\mathcal{B}(w, 3r/2)$. Let $\mathcal{S}$ denote the union of such subsets S in the balls centered at vertices in H^*. With $\delta < 2/15$ and $\alpha \leq \delta r/10$, the interior of each critical path has length more than $r/4$. Therefore, $\mathcal{S}$ hits all the critical paths, and $|\mathcal{S}| = O(h\kappa)$. ◀

Let $\tilde{H}$ denote the hitting set computed by our algorithm. Lemmas 11 and 12 immediately imply the following corollary:

▶ **Corollary 13.** $|\tilde{H}| = O(h\kappa \log(h\kappa))$.

We show that $\tilde{H}$ satisfies the following property.

▶ **Lemma 14.** *$\tilde{H}$ is a δ-approximate r-SPHS of G.*

Proof. If $\mu(u, v) \leq r$, $\pi(u, v)$, the shortest path between u, v, is automatically EV-feasible. We therefore focus on the case $\mu(u, v) > r$. We construct another path $\tilde{P}$ between u, v from $\pi(u, v)$ as follows. For convenience, denote $w_0 = u$ and $w_t = v$. We find vertices $w_1, \cdots, w_{t-1}$ along $\pi(u, v)$ from u to v such that $\mu(w_i, w_{i+1}) \in [(\frac{1}{2} - \frac{\delta}{4})r - \alpha, (\frac{1}{2} - \frac{\delta}{4})r]$, for $i = 1, \cdots, t-1$. Let $c_i \in C$ denote the nearest center to w_i. We set $\tilde{P}$ as the concatenation of the shortest paths $\pi(w_0, c_0), \pi(c_0, c_1), \cdots, \pi(c_{t-1}, c_t), \pi(c_t, w_t)$. See Figure 2. Then

$$\begin{aligned}
\ell(\tilde{P}) &= \mu(w_0, c_0) + \mu(c_t, w_t) + \sum_{i=1}^{t-1} \mu(c_i, c_{i+1}) \\
&\leq \frac{\delta}{4} r + \sum_{i=1}^{t-1} (\mu(c_i, w_i) + \mu(w_i, w_{i+1}) + \mu(w_{i+1}, c_{i+1})) \\
&\leq \frac{\delta}{4} r + \sum_{i=1}^{t-1} (1 + \frac{3\delta}{4}) \mu(w_i, w_{i+1}) \qquad (\delta \leq 2/15) \leq (1 + \delta)\mu(u, v).
\end{aligned}$$

Next, we show that path $\tilde{P}$ is EV-feasible with respect to $\tilde{H}$. By triangle inequality, it is easy to check that $\mu(c_i, c_{i+1}) \in [(\frac{1}{2} - \frac{\delta}{2})r - \alpha, \frac{1}{2}r]$; thus $\pi(c_i, c_{i+1})$ is a critical path and contains a vertex of $\tilde{H}$ in its interior. $\tilde{P}$ is EV-feasible since every subpath of $\tilde{P}$ of length larger than r contains a vertex of $\tilde{H}$. ◀

Putting Lemmas 9, 10, 11, and 12 together, the expected running time of our algorithm is $O(\frac{1}{\delta^d}(m + n \log^2 n \log \kappa) + m \log n) = O(\frac{1}{\delta^d} n \log^2 n \log \kappa)$ with the assumption $m = O(n)$. This bound along with Corollary 13 and Lemma 14 proves Theorem 1.

5 Feasible path query

Given a δ-approximate r-SPHS H, we consider the task of computing the shortest r-EV-feasible path between any two vertices $u, v \in V$ with respect to H. By definition of approximate SPHS, the length of this path is at most $(1+\delta)\mu(u,v)$. A shortest feasible path can be compactly represented by the sequence of charging stations in H it passes through; the distance between any two consecutive stations is at most r. We can recover the whole feasible path by retrieving the shortest paths between consecutive stations in G.

We first show that the δ-approximate r-SPHS $\tilde{H}$ output by our algorithm can be postprocessed and replaced with a smaller δ-approximate r-SPHS $\hat{H}$ such that $|\hat{H} \cap \mathcal{B}(v,r)|$ is small for any $v \in V$.[2] This property of $\hat{H}$ ensures small feasible path query time with respect to a set of charging stations $\hat{H}$.

Postprocessing step. We show that $\tilde{H}$ can be replaced by another approximate r-SPHS $\hat{H}$ such that $|\hat{H}| \le |\tilde{H}|$ and for any $u \in \hat{H}$, $|\mathcal{B}(u,r) \cap \hat{H}| = O(1)$, where the constant depends on the highway dimension of G. The algorithm works as follows.

The algorithm maintains an r-SPHS H. Initially, $H = \tilde{H}$. For each vertex $v \in V$, it also maintains the set $H_v = \{u \in H \mid \mu(u,v) \le r\}$, i.e., $\mathcal{B}(v,r) \cap H$, and the value $|H_v|$. We fix a constant c and call a vertex $v \in V$ *heavy* if $|H_v| > ch \ln h$. At each step, the algorithm checks whether there is a heavy vertex in V. If there is no heavy vertex, it returns the current set H as $\hat{H}$. Otherwise, let v be a heavy vertex. Let $\Sigma_v = (V, \mathcal{R}_v)$ be a range space where $\mathcal{R}_v$ corresponds to critical paths intersecting $\mathcal{B}(v,r)$. Since each critical path has length no more than $r/2$, all the critical paths in $\mathcal{R}_v$ lie inside $\mathcal{B}(v, 3r/2)$. By definition of highway dimension, there exists a hitting set of size h for Σ_v. We can use the same hitting-set algorithm [4] to compute a hitting set X_v of Σ_v of size at most $ch \ln h$ in $O(\frac{1}{\delta^d} n \log^2 n)$ expected time. It then replaces H with $(H \setminus H_v) \cup X_v$. Finally, we compute $\mathcal{B}(u,r)$ for each $u \in X_v$ and update the sets H_w for all w in these balls.

Since v is heavy, each step of the algorithm except the last one reduces the size of H by at least one, so it terminates within $|\tilde{H}|$ rounds. $\hat{H}$ is a δ-approximate r-SPHS since it hits every critical path. Hence, we obtain the following.

▶ **Lemma 15.** *A δ-approximate r-SPHS $\hat{H} \subseteq V$ of size $O(\kappa \log \kappa)$ can be computed in $O(\frac{1}{\delta^d} \kappa n \log^2 n \log \kappa)$ time so that $|\mathcal{B}(v,r) \cap \hat{H}| = O(1)$ for all $v \in V$.*

Feasible path query. A shortest r-EV-feasible path must pass through a sequence of charging stations, and any two consecutive charging stations on the path must be at most r apart. Define the graph $\mathcal{N} = (\hat{H}, \hat{E})$ where $\hat{E} = \{(u,v) \mid \mu(u,v) \le r\}$. For each edge $(u,v) \in \hat{E}$, define $\ell(u,v) = \mu(u,v)$. By Lemma 15, $|\hat{E}| = O(|\hat{H}|) = O(\kappa \log \kappa)$. By constructing $\mathcal{B}(u,r)$ for all $u \in \hat{H}$, we can construct the edges in $\hat{E}$ and their lengths.

As for computing a shortest feasible path between any pair of vertices of G, we maintain, for each $v \in V$, $\hat{H}_v = \mathcal{B}(v,r) \cap \hat{H}$ along with their distances from v. Given $s, t \in V$, we augment $\mathcal{N}$ by adding edges from s to $\hat{H}_s$ and t to $\hat{H}_t$, and compute a shortest path from s to t in $\mathcal{N}$ using the Dijkstra's algorithm. Putting everything together, we obtain the following.

▶ **Theorem 16.** *Let $G = (V, E)$, $\ell : V \to \mathbb{R}^+$ be a weighted graph of constant highway dimension h, with $|V| = n$ and $|E| = O(|V|)$. Let $r > 0$ and $\delta \in [\frac{10\alpha}{r}, \frac{2}{15}]$ be two para-*

[2] We conjecture that $\tilde{H}$ already satisfies $|\mathcal{B}(u,r) \cap \tilde{H}| = O(\log \kappa)$ for all $u \in V$, and no postprocessing is needed, but so far we have run into technical difficulties in proving this conjecture.

meters where $\alpha = \max_{e \in E} \ell(e)$*, and let* κ *be the size of a smallest* r*-SPHS of* G*. In time* $O(\frac{1}{\delta^d}\kappa n \log^2 n \log \kappa)$*, a* δ*-approximate* r*-SPHS* $\hat{H}$ *can be computed and* G *can be processed into a data structure of size* $O(\kappa \log \kappa)$ *such that for any two vertices* $s, t \in V$*, a compact representation of a shortest* r*-EV-feasible path from* s *to* t*, using* $\hat{H}$*, can be computed in* $O(\kappa \log^2 \kappa)$ *time.*

6 Conclusion

In this paper, we presented a bi-criteria approximation algorithm for the r-SPHS problem whose running time is near-linear in n. The algorithm assumes the input graph has constant highway dimension, a concept introduced to give rigorous proofs of efficiency for many popular heuristic shortest path algorithms [3]. Our algorithm is the first for such problems with provable guarantees on the approximation and running time. We also give an algorithm for computing the shortest EV-feasible paths given the set of charging stations computed by the first algorithm.

It is also interesting to know whether it is possible to improve the size of the ε-net for the range space of shortest paths from $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ to $O(\frac{1}{\varepsilon})$. If so, it will improve the approximation ratio of our algorithm from $O(\log \kappa)$ to $O(1)$.

Additionally, no efficient algorithm is known for the maximum coverage version of the SPHS problem. Given a collection $\mathcal{P}$ of input paths in a graph G (may or may not be shortest) and an integer k, the goal is to compute a subset of k vertices such that the number of EV-feasible paths in $\mathcal{P}$ (with respect to the subset) is maximized . This problem is not submodular because it can take more than one vertex to make one long path in $\mathcal{P}$ EV-feasible.

Finally, we have shown that the SPHS problem is NP-complete for general graphs, but we do not known whether it is NP-complete for graphs of constant highway dimension. A proof of NP-completeness may require more insights into the structure of such graphs.

Acknowledgements. We thank Dan Halperin and Eli Packer for introducing the problem to us and for helpful discussions.

References

1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. VC-dimension and shortest path algorithms. In *Proc. 38th Int'l Colloq. Conf. Automata, Languages and Programming*, pages 690–699, 2011.

2 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension and provably efficient shortest path algorithms. *Tech. Report MSRTR-2013-91, Microsoft Research*, 2013.

3 Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proc. 21st Annual ACM-SIAM Symp. Discrete Algorithms*, pages 782–793, 2010.

4 Pankaj K Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Proc. 30th Annual Symp. Comput. Geo.*, page 271, 2014.

5 Boris Aronov, Esther Ezra, and Micha Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, 2010.

6 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. *arXiv preprint*, 2015.

7 Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. In *Proc. 23rd SIGSPATIAL Int'l Conf. Advances in Geo. Info. Syst.*, page 44, 2015.

8 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-optimal routes for electric vehicles. In *Proc. 21st ACM SIGSPATIAL Int'l Conf. Advances in Geo. Info. Syst.*, pages 54–63, 2013.

9 Vasek Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979.

10 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proc. 46th Annual ACM Symp. Theory of Computing*, pages 624–633. ACM, 2014.

11 Inês Frade, Anabela Ribeiro, Gonçalo Gonçalves, and António Antunes. Optimal location of charging stations for electric vehicles in a neighborhood in lisbon, portugal. *Transportation Res. Record: J. Transportation Res. Board*, 2252(2252):91–98, 2011.

12 Stefan Funke, André Nusser, and Sabine Storandt. On k-path covers and their applications. *Proc. VLDB Endowment*, 7(10):893–902, 2014.

13 Stefan Funke, André Nusser, and Sabine Storandt. Placement of loading stations for electric vehicles: No detours necessary! *J. Arti. Intelli. Res.*, pages 633–658, 2015.

14 Michael T Goodrich and Paweł Pszona. Two-phase bicriterion search for finding fast and efficient electric vehicle routes. In *Proc. 22nd ACM SIGSPATIAL Int'l Conf. Advances in Geo. Info. Syst.*, pages 193–202, 2014.

15 Sariel Har-Peled. *Geometric Approximation Algorithms.* American Math. Soc., 2011.

16 David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Comput. Geo.*, 2(2):127–151, 1987.

17 RM Karp. Reducibility among combinatorial problems. *Complexity Comp. Comput.*, 1972.

18 Nabil H Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Comput. Geo.*, 44(4):883–895, 2010.

19 Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient energy-optimal routing for electric vehicles. In *Proc. 25th AAAI Conf. Arti. Intelli.*, 2011.

20 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comp. Syst. Sci.*, 26(3):362–391, 1983.

21 Sabine Storandt and Stefab Funke. Enabling e-mobility: Facility location for battery loading stations. In *Proc. 27th AAAI Conf. Arti. Intelli.*, 2013.

22 Moby Khan T. Donna Chen, Kara M. Kockelman. The electric vehicle charging station location problem: a parking-based assignment method for seattle. In *Proc. 92nd Annual Meet. Transportation Res. Board*, 2013.

23 Yufei Tao, Cheng Sheng, and Jian Pei. On k-skip shortest paths. In *Proc. ACM SIGMOD*, pages 421–432. ACM, 2011.

24 VN Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264, 1971.

25 Vijay V Vazirani. *Approximation algorithms.* Springer Sci. & Business Media, 2013.

26 Yanhai Xiong, Jiarui Gan, Bo An, Chunyan Miao, and Ana LC Bazzan. Optimal electric vehicle charging station placement. In *Proc. 24th Int'l Joint Conf. Arti. Intelli.*, pages 2662–2668, 2015.

Finding k Simple Shortest Paths and Cycles*

Udit Agarwal[1] and Vijaya Ramachandran[2]

1 Dept. of Computer Science, University of Texas, Austin, USA
udit@cs.utexas.edu

2 Dept. of Computer Science, University of Texas, Austin, USA
vlr@cs.utexas.edu

Abstract

We present algorithms and techniques for several problems related to finding multiple simple shortest paths and cycles in a graph. Our main result is a new algorithm for finding k simple shortest paths for all pairs of vertices in a weighted directed graph $G = (V, E)$. For $k = 2$ our algorithm runs in $O(mn + n^2 \log n)$ time where m and n are the number of edges and vertices in G. For $k = 3$ our algorithm runs in $O(mn^2 + n^3 \log n)$ time, which is almost a factor of n faster than the best previous algorithm.

Our approach is based on forming suitable path extensions to find simple shortest paths; this method is different from the 'detour finding' technique used in most of the prior work on simple shortest paths, replacement paths, and distance sensitivity oracles.

We present new algorithms for generating simple cycles and simple paths in G in non-decreasing order of their weight. The algorithm for generating simple paths is much faster, and uses another variant of path extensions.

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

Keywords and phrases Graph Algorithms, Shortest Paths, k Simple Shortest Paths, Enumerating Simple Cycles, Enumerating Simple Paths

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.8

1 Introduction

We present new algorithms and fundamentally new techniques for several problems related to finding multiple simple shortest paths and cycles in a graph.

Computing shortest paths in a weighted directed graph is a very well-studied problem. Let $G = (V, E)$ be a directed graph with non-negative edge weights, with $|V| = n$, $|E| = m$. A shortest path for a single pair of vertices in G, or for a single source, can be computed in $\tilde{O}(m)$ time using Dijkstra's algorithm, and the all pairs shortest paths (APSP) can be computed in $\tilde{O}(mn)$ time [4], where $\tilde{O}$ hides *polylog*(n) factors.

A related problem is one of computing a sequence of k shortest paths, for $k > 1$. If the paths need not be simple, the problem of generating k shortest paths is well understood, and the most efficient algorithm is due to Eppstein [8], which has the following bounds – $O(m + n \log n + k)$ for a single pair of vertices and $O(m + n \log n + kn)$ for single source.

In the k *simple shortest paths (k-SiSP)* problem, given a pair of vertices s, t, the output is a sequence of k simple paths from s to t, where the i-th path in the collection is a shortest simple path in the graph that is not identical to any of the $i - 1$ paths preceding it in the

* This work was supported in part by NSF Grant CCF-1320675. The first author's research was also supported in part by a Calhoun Fellowship.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 8; pp. 8:1–8:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

output. (Note that these k simple shortest paths need not have the same weight.) It is noted in [8] that the k-SiSP problem is more common than the version where a path can contain cycles.

In this paper we consider the problem of generating multiple simple shortest paths (SiSP) and cycles (SiSC) in a weighted directed graph under the following set-ups: the k simple shortest paths for all pairs of vertices (k-APSiSP), k simple shortest paths in the overall graph (k-All-SiSP), and the corresponding problem of finding simple shortest cycles in the overall graph (k-All-SiSC). We obtain significantly faster algorithms for k-APSiSP for small values of k, and fast algorithms, that also appear to be the first nontrivial algorithms, for the remaining two problems for all $k \geq 1$. Implicit in our method for k-All-SiSC are new algorithms for finding k simple shortest cycles through a specified vertex (k-SiSC) and through every vertex (k-ANSiSC) in weighted directed graphs.

The techniques we use in our algorithms are of special interest: We use two *path extension* techniques, a new method for k-APSiSP, and another for k-All-SiSP that is related to a method used in [5] for fully dynamic APSP, but which is still new for the context in which we use it.

1.1 Related Work

For the case when the k shortest paths need not be simple, the all-pairs version (k-APSP) was considered in the classical papers of Lawler [15, 16] and Minieka [17]. The most efficient current algorithm for k-APSP runs the k-SSSP algorithm in [8] on each of the n vertices in turn, leading to a bound of $O(mn + n^2 \log n + kn^2)$. It was noted in Minieka [17] that the all-pairs version of k shortest paths becomes significantly harder when simple paths are required, i.e., that the problem we study here, k-APSiSP, appears to be significantly harder than k-APSP.

Even for a single source-sink pair, the problem of generating k simple shortest paths (k-SiSP) is considerably more challenging than the unrestricted version considered in [8]. Yen's algorithm [24] finds the k simple shortest paths for a specific pair of vertices in $O(k \cdot (mn + n^2 \log n))$. This time bound was improved slightly [9], using Pettie's faster APSP algorithm [18], to $O(k(mn + n^2 \log \log n))$. On the other hand, it is shown in [23] that if the *second* simple shortest path for a single source-sink pair (i.e., $k = 2$ in k-SiSP) can be found in $O(n^{3-\delta})$ time for some $\delta > 0$, then APSP can also be computed in $O(n^{3-\alpha})$ time for some $\alpha > 0$; the latter is a major open problem. Thus, for dense graphs, where $m = \Theta(n^2)$, we cannot expect to improve the $\tilde{O}(mn)$ bound, even for 2-SiSP, unless we solve a major and long-standing open problem for APSP.

The k-SiSP problem is much simpler in the undirected case and is known to be solvable in $O(k(m + n \log n))$ time [14]. For unweighted directed graphs, Roditty and Zwick [19] gave an $\tilde{O}(km\sqrt{n})$ randomized algorithm for directed k-SiSP. They also showed that k-SiSP can be solved with $O(k)$ executions of an algorithm for the 2-SiSP problem.

A problem related to 2-SiSP is the *replacement paths* problem. In the s-t version of this problem, we need to output a shortest path from s to t when an edge on the shortest path p is removed; the output is a collection of $|p|$ paths, each a shortest path from s to t when an edge on p is removed. Clearly, given a solution to the s-t replacement paths problem, the second shortest path from s to t can be computed as the path of minimum weight in this solution. This is essentially the method used in all prior algorithms for 2-SiSP (and with modifications, for k-SiSP), and thus the current fastest algorithms for 2-SiSP and replacement paths have the same time bound. For the all-pairs case that is of interest to us, the output for the replacement paths problem would be $O(n^3)$ paths, where each path is shortest for a specific vertex pair, when a specific edge in its shortest path is removed. In view of the large space needed for this output, in the all-pairs version of replacement paths, the problem of interest is *distance sensitivity oracles (DSO).* Here, the output is a compact representation from which any specific replacement path can be found with $O(1)$ time. The first such oracle was developed in Demetrescu et al. [7], and it has size $O(n^2 \log n)$. The current best construction time for an oracle of this size is $O(mn \log n + n^2 \log^2 n)$ time for a randomized algorithm, and a log factor slower for a deterministic algorithm, given in Bernstein and Karger [3]. Given such an oracle, the output to 2-APSiSP can be computed with $O(n)$ queries for each source-sink pair, i.e., with $O(n^3)$ queries to the DSO.

To the best of our knowledge, for $k > 1$ the problem of generating k simple shortest cycles in the overall graph in non-decreasing order of their weights (k-All-SiSC) has not been studied before, and neither has k-SiSC (k Simple Shortest Cycles through a given node) or k-ANSiSC (k All Nodes Simple Shortest Cycles); for $k = 1$, 1-All-SiSC asks for a minimum weight cycle and 1-ANSiSC is the ANSC problem [25], both of which can be found in $\tilde{O}(mn)$ time, and 1-SiSC can be solved in $\tilde{O}(m + n)$ time. On the other hand, enumerating simple (or *elementary*) cycles in no particular order – which is thus a special case of k-All-SiSC – has been studied extensively [21, 22, 20, 11]. The first polynomial time algorithm was given by Tarjan [20], and ran in $O(kmn)$ time for k cycles. This result was improved to $O(k \cdot m + n)$ by Johnson [11]. We do not expect to match this linear time result for k-All-SiSC since it includes the minimum weight cycle problem for $k = 1$.

In this paper, we concentrate on results for truly sparse graphs with arbitrary non-negative edge weights. Hence we do not consider results for small integers weights or for dense graphs; several subcubic results for such inputs are known using fast matrix multiplication.

1.2 Our Contributions

We present several algorithmic results on finding k simple paths and cycles in a directed graph with non-negative edge weights.[1] A summary of our results is given in Table 1.

Computing k simple shortest paths for all pairs (k-APSiSP) in G. We present a new approach to the k-APSiSP problem, which computes the sets $P_k^*(x, y)$ as defined below. Our method introduces the key notion of a 'nearly k SiSP set', $Q_k(x, y)$, defined as follows.

▶ **Definition 1.1.** Let $G = (V, E)$ be a directed graph with nonnegative edge weights. For $k \geq 2$, and a vertex pair x, y, let $k^* = \min\{r, k\}$, where r is the number of simple paths from x to y in G. Then,

(i) $P_k^*(x, y)$ is the set of k^* *simple shortest paths from x to y* in G

(ii) $Q_k(x, y)$ is the set of k *nearly simple shortest paths from x to y*, defined as follows. If $k^* = k$ and the $k - 1$ simple shortest paths from x to y share the same first edge (x, a) then $Q_k(x, y)$ contains these $k-1$ simple shortest paths, together with the simple shortest path from x to y that does not start with edge (x, a), if such a path exists. Otherwise (i.e, if either the former or latter condition does not hold), $Q_k(x, y) = P_k^*(x, y)$.

[1] Except for k-All-SiSP (see Section 3.1), we can also handle negative edge-weights as long as there are no negative-weight cycles, by applying Johnson's transformation [12] to obtain an equivalent input with nonnegative edge weights. If the resulting edge-weights include weight 0, we will use the pair $(wt(p), len(p))$ as the weight for path p, where $len(p)$ is the number of edges in it; this causes the weight of a proper subpath of p to be smaller than the weight of p.

Our algorithm for k-APSiSP first constructs $Q_k(x,y)$ for all pairs of vertices x,y, and then uses these sets in an efficient algorithm, Compute-APSiSP, to compute the $P_k^*(x,y)$ for all x,y. The latter algorithm runs in time $O(k \cdot n^2 + n^2 \log n)$ for any k, while our method for constructing the $Q_k(x,y)$ depends on k. For $k=2$ we present an $O(mn + n^2 \log n)$ time method to compute the $Q_2(x,y)$ sets; this gives a 2-APSiSP algorithm that matches Yen's bound of $O(mn + n^2 \log n)$ for 2-*SiSP for a single pair of vertices.* It is also faster (by a polylogarithmic factor) than the best algorithm for DSO (distance sensitivity oracles) for the all-pairs replacement paths problem [3]. In fact, we also show that the $Q_2(x,y)$ sets can be computed in $O(n^2)$ time using a DSO, and hence 2-APSiSP can be computed in $O(n^2 \log n)$ time plus the time to construct the DSO.

For $k \geq 3$ our algorithm to compute the Q_k sets makes calls to an algorithm for $(k-1)$-APSiSP, so we combine the two components together in a single recursive method, APSiSP, that takes as input G and k, and outputs the P_k^* sets for all vertex pairs. The time bound for APSiSP increases with k: it is faster than Yen's method for $k=3$ by a factor of n (and hence is faster than the current fastest method by almost a factor of n), it matches Yen for $k=4$, and its performance degrades for larger k.

If a faster algorithm can be designed to compute the Q_k sets, then we can run Compute-APSiSP on its output and hence compute k-APSiSP in additional $O(k \cdot n^2 + n^2 \log n)$ time. Thus, a major open problem left by our results is the design of a faster algorithm to compute the Q_k sets for larger values of k.

New Approach: Computing simple shortest paths without finding detours. Our method for computing k-APSiSP (using the $Q_k(x,y)$ sets) extends an existing simple path in the data structure to create a new simple path by adding a single incoming edge. This approach differs from all previous approaches to finding k simple paths and replacement paths. All known previous algorithms for 2-SiSP compute replacement paths for every edge on the shortest path (by computing suitable 'detours'). In fact, Hershberger et al. [10] present a lower bound for k-SiSP, exclusively for the class of algorithms that use detours, by pointing out that all known algorithms for k-SiSP compute replacement paths, and all known replacement path algorithms use detours. In contrast, our method may enumerate and inspect paths that are not detours, including paths with cycles (e.g., Step 17 in algorithm Compute-APSiSP in Section 2.1). Thus our method is fundamentally new.

Generating k simple shortest cycles and paths (k-All-SiSC, k-SiSC, k-ANSiSC) and k-All-SiSP. We consider the problem of generating the k simple shortest cycles in the graph G in nonincreasing order of their weight (k-All-SiSC). In Section 3 we present an algorithm for k-All-SiSC that runs in $\tilde{O}(k \cdot mn)$ time by generating each successive simple shortest cycle in G in $\tilde{O}(mn)$ time. The same algorithm can be used to enumerate all simple cycles in G in nondecreasing order of their weights. Recall that the related problem of simply enumerating simple cycles in a graph in no particular order was a very well-studied classical problem [21, 22, 20, 11] until an algorithm that generates successive cycles in linear time was obtained [11]. Our algorithm does not match the linear time bound per successive cycle, but it is to be noted that 1-All-SiSC (i.e., the problem of generating a minimum weight cycle) is a very fundamental and well-studied problem for which the current best bound is $\tilde{O}(mn)$.

Our algorithm for k-All-SiSC creates a auxiliary graph on which suitable SiSP computation can be performed to generate the desired output. We give fast algorithms for k-SiSC and k-ANSiSC using the same auxiliary graph.

Complementing our result for k-All-SiSC, we present in Section 3.1 an algorithm for k-All-SiSP that generates each successive simple path in $\tilde{O}(k)$ time if $k < n$, and in $\tilde{O}(n)$

time if $k > n$, after an initial start-up cost of $O(m)$ to find the first path. This time bound is considerably faster than that for k-All-SiSC. Our method, All-SiSP, is again one of extending existing paths by an edge (as is Compute-APSiSP); it is, however, a different path extension method.

Path Extensions. We use two different path extension methods, one for k-APSiSP and the other for k-All-SiSP. Path extensions have been used before in the hidden paths algorithm for APSP [13] and more recently, for fully dynamic APSP [5]. These two path extension methods differ from each other, as noted in [6]. Our path extension method for k-All-SiSP is inspired by a method in [5] to compute 'locally shortest paths' for fully dynamic APSP. Our path extension method for k-APSiSP appears to be new.

Here are the main theorems we establish for our algorithmic results. In all cases, the input is a directed graph $G = (V, E)$ with nonnegative edge weights, and $|V| = n$, $|E| = m$.

▶ **Theorem 1.2.** *Given an integer $k > 1$, and the nearly simple shortest paths sets $Q_k(x, y)$ (Definition 1.1) for all $x, y \in V$, Algorithm* Compute-APSiSP *(Section 2.1) produces the k simple shortest paths for every pair of vertices in $O(k \cdot n^2 + n^2 \log n)$ time.*

▶ **Theorem 1.3.**

(i) *Algorithm* 2-APSiSP *(Section 2.2.1) correctly computes 2-APSiSP in $O(mn + n^2 \log n)$ time.*

(ii) *For $k > 2$, Algorithm* APSiSP *(Section 2.2.2) correctly computes k-APSiSP in $T(m, n, k)$ time, where $T(m, n, k) \leq n \cdot T(m, n, \ k-1) + O(mn + n^2 \cdot (k + \log n))$.*

(iii) *$T(m, n, 3)$, the time bound for algorithm* APSiSP *for $k = 3$, is $O(m \cdot n^2 + n^3 \cdot \log n)$.*

▶ **Theorem 1.4** (k-All-SiSC)). *After an initial start-up cost of $O(mn+n^2 \log n)$ time, we can compute each successive simple shortest cycle in $O(mn + n^2 \log \log n)$ time. This computes k-All-SiSC (Section 3).*

▶ **Theorem 1.5** ((k-All-SiSP)). *After an initial start-up cost of $O(m)$ time to generate the first path, Algorithm* All-SiSP *(Section 3.1) computes each succeeding simple shortest path with the following bounds:*

(i) *amortized $O(k + \log n)$ time if $k = O(n)$ and $O(n + \log k)$ time if $k = \Omega(n)$;*

(ii) *worst-case $O(k \cdot \log n)$ time if $k = O(n)$, and $O(n \cdot \log k)$ time if $k = \Omega(n)$.*

Space Bounds. Our k-APSiSP algorithm uses $O(k^2 \cdot n^2)$ space, which is a factor of k larger than the bound on the size of the output. In contrast, the earlier path extension algorithms for APSP [13] and for fully dynamic APSP [5] use $\Omega(mn)$ space in the worst case. All of our other algorithms use space $O(kn^2)$ or better.

Only proof sketches are given here; the full proofs of most results as well as details of the algorithms for simple cycles are in the arXiv paper [1]. Table 1 lists our main results.

2 The k-APSiSP Algorithm

In this section, we present our algorithm to compute k-APSiSP on a directed graph $G = (V, E)$ with nonnegative edge-weight function wt. The algorithm has two main steps. In the first step it computes the nearly k-SiSP sets $Q_k(x, y)$ for all pairs x, y. In the second step it computes the exact k-SiSP sets $P^*_k(x, y)$ for all x, y using the $Q_k(x, y)$ sets. This second step is the same for any value of k, and we describe this step first in Section 2.1. We then present efficient algorithms to compute the Q_k sets for $k = 2$ and $k > 2$ in Section 2.2.

Table 1 Our results for directed graphs. All algorithms are deterministic. (DSO stands for Distance Sensitivity Oracles).

Problem	Known Results	New Results
2-APSiSP (Sec. 2.2.1)	$O(n^3 + mn\log^2 n)$ (using DSO [3])	$\boldsymbol{O(mn + n^2 \log n)}$
3-APSiSP (Sec. 2.2.2)	$\tilde{O}(mn^3)$ [24]	$\boldsymbol{O(mn^2 + n^3 \log n)}$
k-SiSC (Sec. 2.3)	–	$\boldsymbol{O(k \cdot (mn + n^2 \log\log n))}$
k-ANSiSC (Sec. 2.3)	–	$\boldsymbol{O(mn + n^2 \log n)}$ **if** $\boldsymbol{k = 2}$ **and** $\boldsymbol{O(k \cdot (mn^2 + n^3 \log\log n))}$ **if** $\boldsymbol{k > 2}$
k-All-SiSC (Sec. 3)	–	$\boldsymbol{\tilde{O}(kmn)}$
k-All-SiSP (Sec. 3.1)	–	$\boldsymbol{\tilde{O}(k)}$ **if** $\boldsymbol{k < n}$ **and** $\boldsymbol{\tilde{O}(n)}$ **if** $\boldsymbol{k \geq n}$ **per path amortized, after a startup cost of** $\boldsymbol{O(m)}$

In all of our algorithms we will maintain the paths in each $P_k^*(x,y)$ and $Q_k(x,y)$ set in an array in nondecreasing order of edge-weights.

2.1 The Compute-APSiSP Procedure

In this section we present an algorithm, Compute-APSiSP, to compute k-APSiSP. This algorithm takes as input, the graph G, together with the nearly k-SiSP sets $Q_k(x,y)$, for each pair of distinct vertices x, y, and outputs the k^* simple shortest paths from x to y in the set $P_k^*(x,y)$ for each pair of vertices $x, y \in V$ (note that k^*, which is defined in Definition 1.1, can be different for different vertex pairs x, y). As noted above, the construction of the $Q_k(x,y)$ sets will be described in the next section.

The *right (left) subpath* of a path π is defined as the path obtained by removing the first (last) edge on π. If π is a single edge (x,y) then this path is the vertex y (x).

▶ **Lemma 2.1.** *Suppose there are k simple shortest paths from x to y, all having the same first edge (x,a). Then $\forall i$, $1 \leq i \leq k$, the right subpath of the i-th simple shortest path from x to y has weight equal to the weight of the i-th simple shortest path from a to y.*

Proof. The result is trivial for $k = 1$. If it holds for $k - 1$ and not k, then the k-th lightest path p from a to y must contain x, and then we would have a shorter path from x to y that avoids edge (x,a). ◀

Algorithm Compute-APSiSP computes the $P_k^*(x,y)$ sets by extending an existing path by an edge. In particular, if the k-SiSPs from x to y all use the same first edge (x,a), then it computes the k-th SiSP by extending the k-th SiSP from a to y (otherwise, the sets $P_k^*(x,y)$ are trivially computed from the sets $Q_k(x,y)$). The algorithm first initializes the $P_k^*(x,y)$ sets with the corresponding $Q_k(x,y)$ sets in Step 4. In Step 5, it checks whether the shortest $k-1$ paths in $P_k^*(x,y)$ have the same first edge and if so, by definition of $Q_k(x,y)$, this $P_k^*(x,y)$ may not have been correctly initialized, and may need to update its k-th shortest path to obtain the correct output. In this case, the common first edge (x,a) is added to the set *Extensions*(a,y) in Step 7. We explain this step below.

We define the *k-Left Extended Simple Path (k-LESiP)* $\pi_{xa,y}$ from x to y as the path $\pi_{xa,y} = (x,a) \circ \pi_{a,y}$, where the path $\pi_{a,y}$ is the k-th shortest path in $Q_k(a,y)$, and $\circ$ denotes the concatenation operation. In our algorithm we will construct k-LESiPs for those pairs x, y for which the $k-1$ simple shortest paths all start with the edge (x,a). The algorithm also maintains a set $Extensions(a, y)$ for each pair of distinct vertices a, y; this set contains those edges (x, a) incoming to a which are the first edge on all $k-1$ SiSPs from x to y. In addition to adding the common first edge (x, a) in the $(k-1)$ SiSPs in $P_k^*(x, y)$ to $Extensions(a, y)$ in Step 7, the algorithm creates the k-LESiP with start edge (x, a) and end vertex y using the k-th shortest path in the set $P_k^*(a, y)$, and adds it to heap H in Steps 8-10. Let $\mathcal{U}$ denote the set of $P_k^*(x, y)$ sets which may need to be updated; these are the sets for which the if condition in Step 5 holds.

In the main while loop in Steps 12-17, a min-weight path is extracted in each iteration. We establish below that this min-weight path is added to the corresponding P_k^* in Step 14 or 15 only if it is the k-th SiSP; in this case, its left extensions are created and added to the heap H in Step 17, and we note that some of these paths could be cyclic.

▶ **Lemma 2.2.** *Let $G = (V, E)$ be a directed graph with nonnegative edge weight function wt, and $\forall x, y \in V$, let the set $Q_k(x, y)$ contain the nearly k-SiSPs from x to y. Then, algorithm* Compute-APSiSP *correctly computes the sets $P_k^*(x, y)$ $\forall x, y \in V$.*

Proof. We first show that every path in $P_k^*(x, y)$ is simple. The initialization in Step 4 adds only simple paths. After that, $P_k^*(x, y)$ is updated only if it is in $\mathcal{U}$. Assume so, and let $(x, a) \in Extensions(a, y)$. As the algorithm only extends along the edges in the $Extensions$ sets, every path from x to y in H has (x, a) as first edge. Now if a cyclic path (say $\pi_{xa,y}$) is added to $P_k^*(x, y)$ from H, then it contains a subpath $\pi_{xa',y}$, but this implies that either $\pi_{xa',y}$, or a path from x to y with smaller weight but not using (x, a) as the first edge, is present in $Q_k(x, y)$. This means that the check in Step 15 will be false, and $\pi_{xa,y}$ will not be added to $P_k^*(x, y)$.

To show that $P_k^*(x, y)$ contains the k shortest simple paths from x to y at termination, we observe that it was initialized with $Q_k(x, y)$, so we only need to ensure that the path of largest weight in $P_k^*(x, y)$ is indeed π_{xy}^k, the k^*-th shortest simple path from x to y. We argue this by showing that π_{ay}, the path obtained from π_{xy}^k by removing its first edge (x, a), must be in $P_k^*(a, y)$ and must have been extended to x and added to H. ◀

It is straightforward to see that Algorithm Compute-APSiSP runs in $O(kn^2 + n^2 \log n)$ time and uses $O(kn^2)$ space.

2.2 Computing the Q_k Sets

2.2.1 Computing Q_k for $k = 2$

We now give an $O(mn + n^2 \log n)$ time algorithm to compute $Q_2(x, y)$ for all pairs x, y. This method uses the procedure fast-exclude from Demetrescu et al. [7], which we now briefly describe (full details of this algorithm can be found in [7]).

Given a rooted tree T, edges (u_1, v_1) and (u_2, v_2) on T are *independent*[7] if the subtree of T rooted at v_1 and the subtree of T rooted at v_2 are disjoint. Given the weighted directed graph $G = (V, E)$, the SSSP tree T_s rooted at a source vertex $s \in V$, and a set S of independent edges in T_s, algorithm fast-exclude in [7] computes, for each edge $e \in S$, a shortest path from s to every other vertex in $G - \{e\}$. This algorithm runs in time $O(m + n \log n)$.

We will compute the second path in each $Q_2(x, y)$ set, for a given $x \in V$, by running fast-exclude with x as source, and with the set of outgoing edges from x in the shortest path tree rooted at x, T_x, as the set S. Clearly, this set S is independent, and hence algorithm fast-exclude will produce its specified output. Now consider any vertex $y \neq x$, and let (x, a) be the first edge on the shortest path from x to y in T_x. By its specification,

Algorithm 1 Compute-APSiSP$(G=(V,E), wt, k, \{Q_k(x,y), \forall x,y\})$

```
Initialize:
$H \leftarrow \phi$     $\{H$ is a priority queue.$\}$
for all $x, y \in V, x \neq y$ do
    $P_k^*(x,y) \leftarrow Q_k(x,y)$
    if the $k-1$ shortest paths in $P_k^*(x,y)$ have the same first edge then
        Let $(x,a)$ be the common first edge in the $(k-1)$ shortest paths in $P_k^*(x,y)$
        Add $(x,a)$ to the set $Extensions(a,y)$
        if $|Q_k(a,y)| = k$ then
            $\pi \leftarrow$ the path of largest weight in $Q_k(a,y)$
            $\pi' \leftarrow (x,a) \circ \pi$; add $\pi'$ to $H$ with weight $wt(x,a) + wt(\pi)$
Main Loop:
while $H \neq \phi$ do
    $\pi \leftarrow$ Extract-min$(H)$; let $\pi = (xa, y)$ and $\pi'$ a path of largest weight in $P_k^*(x,y)$
    if $|P_k^*(x,y)| = k-1$ then add $\pi$ to $P_k^*(x,y)$ and set update flag
    else if $wt(\pi) < wt(\pi')$ then replace $\pi'$ with $\pi$ in $P_k^*(x,y)$ and set update flag
    if update flag is set then
        for all $(x',x) \in Extensions(x,y)$ do add $(x',x) \circ \pi$ to $H$ with weight $wt(x',x) + wt(\pi)$
```

Algorithm 2 2-APSiSP$(G=(V,E); wt)$

```
for each $x \in V$ do
    Compute a shortest path in each $Q_2(x,y)$, $y \in V - \{x\}$ (Dijkstra with source $x$)
    Compute the second path in each $Q_2(x,y)$, $y \in V - \{x\}$, using fast-exclude with source $x$ and $S = \{(x,a) \in T_x\}$
Compute-APSiSP($G$, wt, 2, $\{Q_2(x,y), \forall x,y\}$)
```

fast-exclude will compute a shortest path from x to y that avoids edge (x,a) in its output, which is the second path needed for $Q_2(x,y)$. This holds for every vertex $y \in V - \{x\}$. Thus we have:

▶ **Lemma 2.3.** *The $Q_2(x,y)$ sets for pairs x, y can be computed in $O(mn + n^2 \log n)$ time.*

This leads to the following algorithm for 2-APSiSP. Its time bound in Theorem 1.3, part (i) follows from Lemma 2.3 and the time bound for Compute-APSiSP given in Section 2.1.

The space bound is $O(n^2)$ since the Q_2 sets contain $O(n^2)$ paths and the call to Compute-APSiSP takes $O(n^2)$ space. In the full paper [1] we give a simple alternate algorithm that computes the Q_2 sets in $\tilde{O}(mn)$ time if a DSO is available. It is not clear if we can efficiently compute 2-APSiSP directly from a DSO in $\tilde{O}(mn)$ time, without using the Q_2 sets and Compute-APSiSP.

2.2.2 Computing Q_k for $k \geq 3$

Our algorithm will use the following types of sets. For each vertex $x \in V$, let I_x be the set of incoming edges to x. Also, for a vertex $x \in V$, and vertices $a, y \in V - \{x\}$, let $P_k^{*x}(a,y)$ be the set of k simple shortest paths from a to y in $G - I_x$, the graph obtained after removing the incoming edges to x. Recall that we maintain all P^* and Q sets as sorted arrays.

Algorithm APSiSP(G,k) first computes the sets $P_{k-1}^{*x}(a,y)$, for all vertices $a, y \in V$. Then it computes each $Q_k(x,y)$ as the set of all paths in the set $P_{k-1}^*(x,y)$, together with a

Algorithm 3 APSiSP$(G = (V, E),\ wt,\ k)$

```
if $k = 2$ then
    compute $Q_2$ sets using algorithm in Section 2.2.1
else
    for each $x \in V$ do
        $I_x \leftarrow$ set of incoming edges to $x$
        Call APSiSP$(G - I_x, wt, k-1)$ to compute $P^{*x}_{k-1}(u,v)\ \forall u, v \in V$
        for each $y \in V - \{x\}$ do
            $Q_k(x,y) \leftarrow P^{*x}_{k-1}(x,y)$
            for all $(x,a) \in E$ do $count_a \leftarrow$ number of paths in $Q_k(x,y)$ with $(x,a)$ as the first edge
            $Q_k(x,y) \leftarrow Q_k(x,y) \cup \{$ a shortest path in $\bigcup_{\{(x,a) \text{ outgoing from } x\}} (x,a) \circ P^{*x}_{k-1}(a,y)[count_a + 1]\}$
Compute-APSiSP$(G, wt, k, \{Q_k(x,y)\ \forall x, y \in V\})$
```

shortest path in $\bigcup_{\{(x,a) \text{ outgoing from } x\}} \{(x,a) \circ p \mid p \in P^{*x}_{k-1}(a,y)\}$ (which is not present in $P^{*}_{k-1}(x,y)$).

To compute the P^{*x}_{k-1} sets, APSiSP(G, wt, k) recursively calls APSiSP$(G - I_x, wt, k-1)$ n times, for each vertex $x \in V$. Once we have computed the P^{*x}_{k-1} sets, the $Q_k(x,y)$ sets are readily computed as described in steps 8 - 10. After the computation of $Q_k(x,y)$ sets, APSiSP(G, wt, k) calls Compute-APSiSP$(G, wt, k, \{Q_k(x,y)\ \forall x, y \in V\})$ to compute the P^*_k sets. This establishes part (ii) of Theorem 1.3.

Proof of Theorem 1.3, part (iii). The for loop starting in Step 4 is executed n times, and for $k = 3$ the cost of each iteration is dominated by the call to Algorithm 2-APSiSP in Step 6, which takes $O(mn + n^2 \log n)$ time. This contributes $O(mn^2 + n^3 \log n)$ to the total running time. The inner for loop starting in Step 7 is executed n times per iteration of the outer for loop, and the cost of each iteration is $O(k + d_x)$. Summing over all $x \in V$, this contributes $O(kn^2 + mn)$ to the total running time. Step 11 runs in $O(n^2 \log n)$ time as shown in Section 2.1. Thus, the total running time is $O(mn^2 + n^3 \log n)$. ◀

The space bound for APSiSP is $O(k^2 \cdot n^2)$, as the P^*_{k-1} and Q_k sets contain $O(kn^2)$ paths, and each recursive call to APSiSP$(G - I_x, wt, k-1)$ needs to maintain the P^*_{r-1} and Q_r sets at each level of recursion. The call to Compute-APSiSP takes $O(kn^2)$ space as noted earlier.

The performance of Algorithm APSiSP degrades by a factor of n with each increase in k. Thus, it matches Yen's algorithm (applied to all-pairs) for $k = 4$, and for larger values of k its performance is worse than Yen.

2.3 Generating k Simple Shortest Cycles

k-SiSC. This is the problem of generating the k simple shortest cycles through a specific vertex z in G. We can reduce this problem to k-SiSP by forming G'_z, where we replace vertex z by vertices z_i and z_o in G'_z, we place a directed edge of weight 0 from z_i to z_o, and we replace each incoming edge to (outgoing edge from) z with an incoming edge to z_i (outgoing edge from z_o) in G'_z. Then the k-th simple shortest path from z_o to z_i in G'_z can been seen to correspond to the k-th simple shortest cycle through z in G. This gives an $O(k \cdot (mn + n^2 \log \log n))$ time algorithm for computing k-SiSC using [9]. We also observe that we can solve k-SiSP from s to t in G if we have an algorithm for k-SiSC: create G' by adding a new vertex x^* and zero weight edges (x^*, s), (t, x^*), and then call k-SiSC for vertex x^*. Thus k-SiSP and k-SiSC are equivalent in complexity in weighted directed graphs.

k-ANSiSC. This is the problem of generating k simple shortest cycles that pass through a given vertex x, for every vertex $x \in V$. For $k = 1$ this problem can be solved in $O(mn + n^2 \log\log n)$ time by computing APSP [25]. For $k = 2$, we can reduce this problem to k-APSiSP by forming the graph G' where for each vertex x, we replace vertex x in G by vertices x_i and x_o in G', we place a directed edge of weight 0 from x_i to x_o, and we replace each edge (u, x) in G by an edge (u_o, x_i) in G' (and hence we also replace each edge (x, v) in G by an edge (x_o, v_i) in G'). For $k > 2$, k-ANSiSC can be computed in $O(k \cdot n \cdot (mn + n^2 \log\log n))$ time by computing k-SiSC for each vertex.

3 Enumerating Simple Shortest Cycles and Paths

In this section, we first give a method to generate each successive simple shortest cycle in $G = (V, E)$ (k-All-SiSC) and then in Section 3.1 we give a faster method to generate simple paths in nondecreasing order of weight (k-All-SiSP).

Enumerating Simple Shortest Cycles (k-All-SiSC). Our algorithm for k-All-SiSC creates an auxiliary graph $G' = (V', E')$ as in the construction for k-ANSiSC in Section 2.3. Our algorithm also maintains a set $\mathcal{C}$ of candidate simple shortest cycles. Initially, our algorithm computes a shortest cycle for each vertex $j \in V$ by running Dijkstra's algorithm with source vertex j_o on the subgraph G'_j of G' induced on $V'_j = \{x_i, x_o \mid x \geq j\}$, to find a shortest path p from j_o to j_i . We store these shortest cycles in $\mathcal{C}$.

For each $k \geq 1$, we generate the k-th simple shortest cycle in G by choosing a minimum weight cycle in $\mathcal{C}$. Let this cycle corresponds to some vertex r and is the k_r-th SiSP from vertex r_o to vertex r_i in G'_r. We then replace this cycle in $\mathcal{C}$ by computing the $(k_r + 1)$-th SiSP from vertex r_o to r_i in G'_r.

The initialization takes $O(mn + n^2 \log n)$ time for the n calls to Dijkstra's algorithm. Thereafter, we generate each new cycle in $O(mn + n^2 \log\log n)$ time using the k-SiSP algorithm [9], by maintaining the relevant information from the computation of earlier cycles.

3.1 Generating Simple Shortest Paths (k-All-SiSP)

Our algorithm for k-All-SiSP is inspired by the method in [5] for fully dynamic APSP. With each path π we will associate two sets of paths $L(\pi)$ and $R(\pi)$ as described below. Similar sets are used in [5] for 'locally shortest paths' but here they have a different use.

Let $\mathcal{P}$ be a collection of simple paths. For a simple path π_{xy} from x to y in $\mathcal{P}$, its *left extension set* $L(\pi_{xy})$ is the set of simple paths $\pi' \in \mathcal{P}$ such that $\pi' = (x', x) \circ \pi_{xy}$, for some $x' \in V$. Similarly, the *right extension set* $R(\pi_{xy})$ is the set of simple paths $\pi'' = \pi_{xy} \circ (y, y')$ such that $\pi'' \in \mathcal{P}$. For a trivial path $\pi = \langle v \rangle$, $L(\pi)$ is the set of incoming edges to v, and $R(\pi)$ is the set of outgoing edges from v.

Algorithm All-SiSP initializes a priority queue H with the edges in G, and it initializes the extension sets for the vertices in G. In each iteration of the main loop, the algorithm extracts the minimum weight path π in H as the next simple path in the output sequence. It then generates suitable extensions of π to be added to H as follows. Let the first edge on π be (x, a) and the last edge (b, y). Then, All-SiSP left extends π along those edges (x', x) such that there is a path $\pi_{x'b}$ in $L(l(\pi))$; it also requires that $x' \neq y$, since extending

Algorithm 4 All-SiSP$(G = (V, E); wt)$

```
1:  Initialization:
2:  $H \leftarrow \phi$      {$H$ is a priority queue.}
3:  for all $(x, y) \in E$ do
4:      Add $(x, y)$ to priority queue $H$ with $wt(x, y)$ as key; add $(x, y)$ to $L(\langle y \rangle)$ and $R(\langle x \rangle)$
5:  Main loop:
6:  while $H \neq \phi$ do
7:      $\pi \leftarrow$ Extract-min$(H)$; add $\pi$ to the output sequence of simple paths
8:      Let $\pi_{xb} = \ell(\pi)$ and $\pi_{ay} = r(\pi)$ (so $(x, a)$ and $(b, y)$ are the first and last edges on $\pi$)
9:      for all $\pi_{x'b} \in L(\pi_{xb})$ with $x' \neq y$ do
10:         Form $\pi_{x'y} \leftarrow (x', x) \circ \pi$ and add $\pi_{x'y}$ to $H$ with $wt(\pi_{x'y})$ as key
11:         Add $\pi_{x'y}$ to $L(\pi_{xy})$ and to $R(\pi_{x'b})$
12:     for all $\pi_{ay'} \in R(\pi_{ay})$ with $y' \neq x$ do perform steps complementary to Steps 10-11
```

to x' would create a cycle in the path. It forms similar extensions to the right in the for loop starting at Step 12.

To establish Theorem 1.5, we first need to show that every path added to H is simple. All edges added in Step 4 are clearly simple paths. Consider a path σ added to H in Step 10. We show that both $\ell(\sigma)$ and $r(\sigma)$ must already be in H, and hence must be simple paths. So, the only way that σ could contain a cycle is if its first and last vertices are the same. But this is explicitly forbidden in the condition in Step 9. A similar argument applies to Step 12.

To show that no simple path in G is omitted in the sequence of simple shortest paths generated, we observe that if π is a simple path of smallest weight not generated by Algorithm All-SiSP, then $\ell(\pi)$ and $r(\pi)$ must have been generated. We can then show that π will be added to H in the iteration of Step 6 when the heavier of $\ell(\pi)$ and $r(\pi)$ is extracted.

The amortized bound in Theorem 1.5 is obtained by implementing H as a Fibonacci heap and the worst-case bound is obtained by using a binary heap.

4 Discussion

Our k-All-SiSP algorithm is nearly optimal if the paths need to be output. It is also not difficult to see that our bounds for 2-APSiSP and k-All-SiSC (for constant k) are the best possible to within a polylog factor for sparse graphs unless the long-standing $\tilde{O}(mn)$ bounds for APSP and minimum weight cycles are improved. In recent work [2] we give several fine-grained reductions that demonstrate that the minimum weight cycle problem holds a central position for a class of problems that currently have $\tilde{O}(mn)$ time bound on sparse graphs, both directed and undirected.

For undirected graphs, our k-All-SiSP result gives an algorithm with the same bound. Also, our k-APSiSP algorithm works for undirected graphs, and this gives a faster algorithm for $k = 2$ and matches the previous best bound (using [14]) for $k = 3$. However, our algorithms for the three variants of finding simple shortest cycles do not work for undirected graphs. This is addressed in our recent work in [2].

The main open question for k-APSiSP is to come up with faster algorithms to compute the $Q_k(x, y)$ sets for larger values of k. This is the key to a faster k-APSiSP algorithm using our approach, for $k > 2$.

References

1 Udit Agarwal and Vijaya Ramachandran. Finding k simple shortest paths and cycles. *arXiv preprint arXiv:1512.02157*, 2015.

2 Udit Agarwal and Vijaya Ramachandran. Fine-grained reductions and algorithms for shortest cycles, 2016. Manuscript.

3 Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. STOC*, pages 101–110, 2009.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

5 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51:968–992, 2004.

6 Camil Demetrescu and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Trans. Alg. (TALG)*, 2:578–601, 2006.

7 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37:1299–1318, 2008.

8 David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28:652–673, 1998.

9 Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inf. Proc. Lett.*, 109(7):352–355, 2009.

10 John Hershberger, Subhash Suri, and Amit Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Alg. (TALG)*, 3(1):5, 2007.

11 Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84, 1975.

12 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *JACM*, 24(1):1–13, 1977.

13 David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM J. Comput.*, 22(6):1199–1217, 1993.

14 Naoki Katoh, Toshihide Ibaraki, and Hisashi Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12(4):411–427, 1982.

15 Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.

16 Eugene L. Lawler. Comment on a computing the k shortest paths in a graph. *CACM*, 20(8):603–605, 1977.

17 E. Minieka. On computing sets of shortest paths in a graph. *CACM*, 17(6):351–353, 1974.

18 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004.

19 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Alg. (TALG)*, 8(4):33, 2012.

20 Robert Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM J. Comput.*, 2(3):211–216, 2005.

21 J. C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *CACM*, 13:722–726, 1970.

22 H. Weinblatt. A new search algorithm to find the elementary circuits of a graph. *JACM*, 19:43–56, 1972.

23 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. IEEE FOCS*, pages 645–654. IEEE, 2010.

24 Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

25 Raphael Yuster. A shortest cycle for each vertex of a graph. *Inf. Proc. Lett.*, 111(21):1057–1061, 2011.

Packing Short Plane Spanning Trees in Complete Geometric Graphs*†

Oswin Aichholzer[1], Thomas Hackl[2], Matias Korman[3], Alexander Pilz[4], Günter Rote[5], André van Renssen[6], Marcel Roeloffzen[7], and Birgit Vogtenhuber[8]

1 **Institute for Software Technology, Graz University of Technology, Austria**
`oaich@ist.tugraz.at`

2 **Institute for Software Technology, Graz University of Technology, Austria**
`thackl@ist.tugraz.at`

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

4 **Department of Computer Science, ETH Zürich, Switzerland**
`alexander.pilz@inf.ethz.ch`

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

6 **National Institute of Informatics, Tokyo, Japan; and**
JST, ERATO, Kawarabayashi Large Graph Project, Japan
`andre@nii.ac.jp`

7 **National Institute of Informatics, Tokyo, Japan; and**
JST, ERATO, Kawarabayashi Large Graph Project, Japan
`marcel@nii.ac.jp`

8 **Institute for Software Technology, Graz University of Technology, Austria**
`bvogt@ist.tugraz.at`

Abstract

Given a set of points in the plane, we want to establish a connection network between these points that consists of several disjoint layers. Motivated by sensor networks, we want that each layer is spanning and plane, and that no edge is very long (when compared to the minimum length needed to obtain a spanning graph). We consider two different approaches: first we show an almost optimal centralized approach to extract two trees. Then we show a constant factor approximation for a distributed model in which each point can compute its adjacencies using only local information. This second approach may create cycles, but maintains planarity.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Geometric Graphs, Graph Packing, Plane Graphs, Minimum Spanning Tree, Bottleneck Edge

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.9

[*] This research was initiated during the 10th European Research Week on Geometric Graphs (GGWeek 2013), Illgau, Switzerland.

[†] O. A., A. P., and B. V. were partially supported by the ESF EUROCORES programme EuroGIGA – ComPoSe, Austrian Science Fund (FWF): I 648-N18. T. H. was supported by the Austrian Science Fund (FWF): P23629-N18 'Combinatorial Problems on Geometric Graphs'. M. K. was supported in part by the ELC project (MEXT KAKENHI No. 12H00855 and 15H02665). A. P. is supported by an Erwin Schrödinger fellowship, Austrian Science Fund (FWF): J-3847-N35.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 9; pp. 9:1–9:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a set S of n points in the plane and an integer k, we are interested in finding k edge-disjoint non-crossing spanning trees $H_1, H_2, \dots, H_k$ on S such that the length $\mathsf{BE}(H_1 \cup H_2 \cup \dots \cup H_k)$ of the *bottleneck edge* (the longest edge which is used) is as short as possible. Each tree H_i is referred to as a *layer* of G. We require each layer to be non-crossing, but edges from different layers are allowed to cross each other. For $k = 1$, the minimum spanning tree $\mathsf{MST}(S)$ solves the problem: its longest edge $\mathsf{BE}(\mathsf{MST}(S))$ is a lower bound on the bottleneck edge of any spanning subgraph, and it is non-crossing. For larger k, we take $\mathsf{BE}(\mathsf{MST}(S))$ as the yardstick and measure the solution quality in terms of $\mathsf{BE}(\mathsf{MST}(S))$ and k.

The particular variation that we consider comes motivated from the field of sensor networks. Imagine one wants to construct a network so that afterwards communication between sensors is possible. One of the most important requirements for such a network is that we can send messages through it easily. Ideally, we want a method that – given the source, destination, information on the current position (and possibly $O(1)$ additional information) – computes the next node to visit in order to reach our destination.

One of the most famous such methods is called *face routing* [7], which guarantees the delivery under the above constraints provided that the underlying graph is plane. Indeed, when considering local routing algorithms in the literature that are guaranteed to succeed, most route deterministically on a plane spanning subgraph of the underlying graph where the plane subgraph can be computed locally. Even though there exist routing strategies for non-plane graphs, in most cases they route through a plane subgraph (for example, Bose *et al.* [2] showed how to locally identify the edges of the Gabriel graph from the unit disk graph). Extending these algorithms for non-plane graphs is a long-standing open problem.

It seems counter-intuitive that having additional edges cannot help in the delivery of messages. In this paper, we provide a different way to avoid this obstacle. Rather than limiting considerations to one plane graph, we aim to construct several disjoint plane spanning graphs. If we split all the messages among the different layers (and route through each layer with routing strategies that work on plane graphs) we can potentially spread the load among a larger number of edges. Another important feature to consider when creating networks is energy consumption. The required energy for sending a message increases with the distance between the two points (usually with the third or fourth power) [4]. Since we want to avoid high energy consumption at one particular node, it is desirable to apply the bottleneck criterion and to minimize the longest edge [6].

Previous Work. This problem falls into the family of *graph packing* problems, where we are given a graph $G = (V, E)$ and a family $\mathcal{F}$ of subgraphs of G. The aim is to pack pairwise disjoint subgraphs $H_1 = (V, E_1), H_2 = (V, E_2), \dots$ into G. A related problem is the *decomposition* of G. In this case, we also look for disjoint subgraphs but require that $\cup_i E_i = E$. For example, there are known characterizations of when we can decompose the complete graph of n points into paths [9] and stars [8]. Dor and Tarsi [3] showed that to determine whether we can decompose a graph G into subgraphs isomorphic to a given graph H is NP-complete. Aichholzer *et al.* [1] showed that any set of n points contains $\Omega(\sqrt{n})$ disjoint plane spanning trees. This bound has been improved to $\lfloor n/3 \rfloor$ by Garcia [5].

In our case, the graph G consists of the complete graph on S, and $\mathcal{F}$ consists of all plane spanning trees of G. We are interested in minimizing a geometric constraint (Euclidean length of the longest edge among the selected graphs of $\mathcal{F}$). To the best of our knowledge, this is the first packing problem of such type.

Results. We give two different approaches to solve the problem. In Section 2 we give a construction for $k = 2$ trees. This construction is centralized in a classical model that assumes that the position of all points are known and computed in a single place. Our construction guarantees that all edges (except possibly one) have length at most $2\mathsf{BE}(\mathsf{MST}(S))$. The remaining edge has length at most $3\mathsf{BE}(\mathsf{MST}(S))$. We complement this construction with a matching worst-case lower bound.

Following the spirit of sensor networks, in Section 3 we use a different approach to construct k disjoint plane graphs (not necessarily trees). The construction works for any $k \leq n/12$ in an almost local fashion. The only global information that is needed is β: $\mathsf{BE}(\mathsf{MST}(S))$ or some upper bound. Each point of S can compute its adjacencies by only looking at nearby points: those at distance $O(k\beta)$.

A simple adversary argument shows that it is impossible to construct spanning networks locally without knowing $\mathsf{BE}(\mathsf{MST}(S))$ (or an upper bound). The lower bound of Section 2 shows that a neighborhood of radius $\Omega(k\mathsf{BE}(\mathsf{MST}(S)))$ may be needed for the network, so we conclude that our construction is asymptotically optimal in terms of the neighborhood.

For simplicity, throughout the paper we make the usual general position assumption that no three points are colinear. Without this assumption, it might be impossible to obtain more than a single plane layer (for example, when all points lie on a line).

2 Centralized Construction

In this section we look for a centralized algorithm to construct two layers. We start with some properties on the minimum spanning tree of a set of points.

▶ **Lemma 1.** *Let S be a set of points in the plane and let uv and vw be two edges of $\mathsf{MST}(S)$. Then the triangle uvw does not contain any other point of S.*

Proof. Observe that, as v is adjacent to both u and w in $\mathsf{MST}(S)$, uw is the longest edge of the triangle uvw (otherwise one could locally shorten $\mathsf{MST}(S)$).

Suppose for the sake of contradiction that there is a point $p \in S$ in the interior of uvw. We split uvw into two sub-triangles by the line ℓ through v perpendicular to the supporting line of u and w. Let Δ_u be the sub-triangle that has u as a vertex, and assume w.l.o.g. that p lies in Δ_u. Note that the edge uv is the hypotenuse of the right-angled triangle Δ_u and hence $\max\{|pu|, |pv|\} < |uv|$.

Consider the paths in $\mathsf{MST}(S)$ from p to u and v, respectively. Since $\mathsf{MST}(S)$ is a tree, one of the two paths must use the edge uv (as otherwise there would be a cycle). Suppose first that this edge is used in the path to u. By removing the edge uv and adding the edge pu to $\mathsf{MST}(S)$ we would obtain a connected (not necessarily plane) tree whose overall weight is smaller, a contradiction. If the edge used is in the path to v, the addition of edge pv yields a similar contradiction. ◀

▶ **Lemma 2.** *Let S be a set of points in the plane. Let $v \in S$ be a point of degree $k \geq 3$ in $\mathsf{MST}(S)$, with $\{v_0, \ldots, v_{k-1}\}$ being the neighbors of v in $\mathsf{MST}(S)$ in counterclockwise order around v. Then for every triple (v_{i-1}, v_i, v_{i+1}) (indices modulo k), the neighbors of v_i in $\mathsf{MST}(S)$ are inside the wedge W_i that is bounded by the rays vv_{i-1} and vv_{i+1} and contains the edge vv_i.*

Proof. Let $u \in S\backslash\{v\}$ be a neighbor of v_i in $\mathsf{MST}(S)$, and assume for the sake of contradiction that u is not in W_i. Then the edge v_iu intersects the boundary of W_i and hence one of the rays starting at v and going through v_{i-1} and v_{i+1}, respectively. Assume without loss of generality that v_iu intersects the ray from v through v_{i+1}. As $\mathsf{MST}(S)$ is plane, the edge v_iu does not intersect the edge vv_{i+1}. Hence, the triangle (v, v_i, u) contains the point v_{i+1} in its interior. As the path vv_iu is a subgraph of $\mathsf{MST}(S)$, this contradicts Lemma 1. ◀

We denote by $\mathsf{MST}^2(S)$ the square of $\mathsf{MST}(S)$, the graph connecting all pairs of points of S that are at distance at most 2 in $\mathsf{MST}(S)$. We call the edges of $\mathsf{MST}(S)$ *short* edges and all remaining edges of $\mathsf{MST}^2(S)$ *long* edges. For every long edge uw, the points u and w have a unique common neighbor v in $\mathsf{MST}(S)$, which we call the *witness* of uw. We define the *wedge* of uw to be the area that is bounded by the rays vu and vw and contains the segment uw. Next we state a simple fact on crossings of the edges in $\mathsf{MST}^2(S)$.

▶ **Lemma 3.** *Let S be a set of points in the plane. Two edges e and f of $\mathsf{MST}^2(S)$ cross if and only if one of the following two conditions is fulfilled:*

1. *At least one of $\{e, f\}$ is a long edge with witness v and wedge W, and the other edge has v as an endpoint and lies inside W.*
2. *Both of $\{e, f\}$ are long edges with the same witness v, and their wedges are intersecting but none is contained in the other.*

Proof. Clearly, if both e and f are short edges, i.e., edges of $\mathsf{MST}(S)$, then they do not cross. Let $f = uw$ be a long edge with witness v and wedge W. Every edge $e = vz$ of $\mathsf{MST}^2(S)$, $z \in S \setminus \{u, v, w\}$ that lies inside W either crosses f or has z inside the triangle $\Delta = (u, v, w)$. The latter is a contradiction to Lemma 1. Obviously, f is neither crossed by any edge incident to u or w, nor crossed by any edge incident to v but not lying inside W.

It remains to prove that every long edge $e = xz$ of $\mathsf{MST}^2(S)$, $x, z \in S \setminus \{u, v, w\}$ that crosses f fulfills Condition 2. Note that for e to cross f, either e has an endpoint inside Δ or e is also crossing one edge out of $\{uv, vw\} \in \mathsf{MST}(S)$. The former is a contradiction to Lemma 1. If e is a short edge (i.e., an edge of $\mathsf{MST}(S)$), then the latter is a contradiction to the planarity of $\mathsf{MST}(S)$. Hence, e is a long edge (with wedge W') and is also crossing one edge g out of $\{uv, vw\} \in \mathsf{MST}(S)$. This also implies that the wedges W and W' intersect in their interiors but none of W, W' is contained in the other. Finally, if e has witness $y \neq v$, then either g has an end point in the triangle xyz or g crosses one edge out of $\{xy, yz\} \in \mathsf{MST}(S)$. Again, the former is a contradiction to Lemma 1 and the latter is a contradiction to the planarity of $\mathsf{MST}(S)$. Hence the witness of e must be v. ◀

With the above observations we can proceed to show a construction that almost works for two layers. To this end we consider the minimum spanning tree $\mathsf{MST}(S)$ to be rooted at a leaf r. For any $v \in S$, we define its *level* $\ell(v)$ as its distance to r in $\mathsf{MST}(S)$. That is, $\ell(v) = 0$ if and only if $v = r$. Likewise, $\ell(v) = 1$ if and only if v is adjacent to r etc.

For any $v \in S \setminus \{r\}$, we define its *parent* $p(v)$ as the first vertex traversed in the unique shortest path from v to r in $\mathsf{MST}(S)$. Similarly, we define its *grandparent* $g(v)$ as $g(v) = p(p(v))$ if $\ell(v) \geq 2$ and as $g(v) = r$ otherwise (i.e., $g(v) = p(v) = r$ if $\ell(v) = 1$). Each vertex q for which $v = p(q)$ is called a child of v.

▶ Construction 4. Let S be a set of points in the plane and let $\mathsf{MST}(S)$ be rooted at one of its leaves, $r \in S$. We construct two graphs $R = G(S, E_R)$ and $B = G(S, E_B)$ as follows: For any vertex $v_o \in S$ whose level is odd, we add the edge $v_op(v_o)$ to E_R and the edge $v_og(v_o)$ to E_B. For any vertex $v_e \in S \setminus \{r\}$ whose level is even, we add the edge $v_eg(v_e)$ to E_R and the edge $v_ep(v_e)$ to E_B.

For simplicity we say that the edges of $R = G(S, E_R)$ are colored red and the edges of $B = G(S, E_B)$ are colored blue. An edge in both graphs is called red-blue.

▶ **Theorem 5.** *Let MST(S) be rooted at r. The two graphs $R = G(S, E_R)$ and $B = G(S, E_B)$ from Construction 4 fulfill the following properties:*

1. *Both R and B are plane spanning trees.*
2. $\max\{\mathsf{BE}(R), \mathsf{BE}(B)\} \leq 2\mathsf{BE}(\mathsf{MST}(S))$.
3. $E_R \cap E_B = \{rs\}$*, with* $r = p(s)$*, i.e.,* $|E_R \cap E_B| = 1$.

Proof. Recall from Construction 4 that r is a leaf of $\mathsf{MST}(S)$. Hence r has a unique neighbor s in $\mathsf{MST}(S)$ and we have $r = p(s) = g(s)$ and $\ell(s) = 1$. Let $S_o \subset S \setminus \{s\}$ be all $v_o \in S$ whose level $\ell(v_o)$ is odd. Likewise, let $S_e \subset S \setminus \{r\}$ be all $v_e \in S$ whose level $\ell(v_e)$ is even. By the construction, the set of red edges is $E_R = \bigcup_{v_o \in S_o} \{v_o p(v_o)\} \cup \bigcup_{v_e \in S_e} \{v_e g(v_e)\} \cup \{rs\}$ and the set of blue edges is $E_B = \bigcup_{v_o \in S_o} \{v_o g(v_o)\} \cup \bigcup_{v_e \in S_e} \{v_e p(v_e)\} \cup \{rs\}$. Thus, the edge rs is the single shared edge between the sets E_R and E_B, as stated in Property 3.

As E_R and E_B are subsets of the edge set of $\mathsf{MST}^2(S)$, the vertices of every edge in E_R and E_B have link distance at most 2 in $\mathsf{MST}(S)$, and the bound on $\max\{\mathsf{BE}(R), \mathsf{BE}(B)\}$ stated in Property 2 follows.

Further, both R and B are spanning trees, i.e., connected and cycle free graphs, as each vertex except r is connected either to its parent or grandparent in $\mathsf{MST}(S)$. To prove Property 1, it remains to show that both trees are plane.

Assume for the sake of contradiction that an edge f is crossed by an edge e of the same color. Recall that all edges of E_R and E_B are edges of $\mathsf{MST}^2(S)$ whose endpoints have different levels. By Lemma 3, at least one of $\{e, f\}$ has to be a long edge. Without loss of generality let $f = uw$ be a long edge and let v be the witness of f with $\ell(u) = \ell(v) - 1 = \ell(w) - 2$. First note that v cannot be an endpoint of e due to its level. That is, uv is not crossing f (common endpoint) and all other edges incident to v in E_R and E_B are either blue if f is red, or red if f is blue. Further, v cannot be the witness of e due to its level. All edges E_R and E_B with witness v have u as one of its endpoints (as for all other edges with witness v in $\mathsf{MST}^2(S)$, both endpoints have the same level). With u as a shared vertex, the edges e and f cannot cross. As e is neither incident to v nor has v as a witness, e crossing f is a contradiction to Lemma 3. This proves Property 1 and concludes the proof. ◀

The properties of our construction imply a first result stated in the following corollary.

▶ **Corollary 6.** *For any set S of n points in the plane, there exist two plane spanning trees $R = G(S, E_R)$ and $B = G(S, E_B)$ such that $|E_R \cap E_B| = 1$ and* $\max\{\mathsf{BE}(R), \mathsf{BE}(B)\} \leq 2\mathsf{BE}(\mathsf{MST}(S))$.

Construction 4 is almost valid in the sense that only one edge was shared between both trees. In the following we enhance this construction so as to avoid the shared edge.

Let $N^- \subset (S \setminus \{r\})$ be the set of neighbors $v^- \in N^-$ of s in $\mathsf{MST}(S)$ such that the ordered triangle rsv^- is oriented clockwise. Let $N^+ \subset (S \setminus \{r\})$ be the set of neighbors $v^+ \in N^+$ of s in $\mathsf{MST}(S)$ such that the ordered triangle rsv^+ is oriented counter-clockwise. Let T^- be the subtree of $\mathsf{MST}(S)$ that is connected to s via the vertices in N^- and let T^+ be the subtree of $\mathsf{MST}(S)$ that is connected to s via the vertices in N^+. Let $S^- \subset S$ consist of r and the set of vertices from T^- and let $S^+ \subset S$ consist of r and the set of vertices from T^+. Observe that $S^- \cap S^+ = \{r, s\}$. Let $E_R^- \subset E_R$ ($E_B^- \subset E_B$) be the subset of edges that have at least one endpoint in $S^- \setminus \{r, s\}$ and let $E_R^+ \subset E_R$ ($E_B^+ \subset E_B$) be the subset of edges that have at least one endpoint in $S^+ \setminus \{r, s\}$. Note that by this definition $E_R = E_R^- \cup E_R^+ \cup \{rs\}$ and $E_B = E_B^- \cup E_B^+ \cup \{rs\}$. With this we define the subgraphs $R^- = G(S^-, E_R^-)$, $R^+ = G(S^+, E_R^+)$, $B^- = G(S^-, E_B^-)$, and $B^+ = G(S^+, E_B^+)$. The following property follows from Lemma 3.

▶ **Lemma 7.** *For any set S of n points in the plane, let $R = G(S, E_R)$ and $B = G(S, E_B)$ be the graphs from Construction 4. Then no edge in E_R^- crosses an edge in E_B^+ and no edge in E_R^+ crosses any edge in E_B^-.*

Proof. Consider any edge $e \in E_R^-$ that is not incident to r. By Lemma 3, such an edge e can be crossed only by an edge incident to at least one vertex of $S^- \setminus \{r, s\}$. Hence, e does not cross any edge of E_B^+.

Assume for the sake of contradiction that there exists an edge $f \in E_B^+$ that crosses an edge $e \in E_R^-$ that is incident to r. By construction, $e = rz$ is a long edge of $\mathsf{MST}^2(S)$ with witness s and wedge W. By Lemma 3, f has to be incident to s, since s cannot be the witness of any blue edges by construction. If f is a short edge, then f is not in W by our definition of S^- and S^+, which is a contradiction to Lemma 3. Hence, let $f = sc$ be a long edge of $\mathsf{MST}^2(S)$ with witness b. Following Lemma 3, the witness b must be s, which is in contradiction to the fact that s cannot be a witness of any blue edge. This concludes the proof that no edge in E_R^- is crossed by an edge in E_B^+. Symmetric arguments prove that no edge in E_R^+ is crossed by an edge in E_B^-. ◀

With this observation we can now prove that the two spanning trees from Construction 4 actually exist in 4 different color combination variants.

▶ **Lemma 8.** *Let S be a set of n points in the plane. Let $R = G(S, E_R)$ and $B = G(S, E_B)$ be the graphs from Construction 4 and let $R^- = G(S^-, E_R^-)$, $R^+ = G(S^+, E_R^+)$, $B^- = G(S^-, E_B^-)$, and $B^+ = G(S^+, E_B^+)$ be subgraphs as defined above. Then R and B can be recolored to be (1) $R = G(S, E_R)$ and $B = G(S, E_B)$ (the "original coloring"), (2) $R = G(S, E_B)$ and $B = G(S, E_R)$ (the "inverted coloring"), (3) $R = G(S, E_B^- \cup E_R^+ \cup \{rs\})$ and $B = G(S, E_R^- \cup E_B^+ \cup \{rs\})$ (the "− side inverted coloring"), and (4) $R = G(S, E_R^- \cup E_B^+ \cup \{rs\})$ and $B = G(S, E_B^- \cup E_R^+ \cup \{rs\})$ (the "+ side inverted coloring"), such that the properties from Theorem 5 hold for all versions.*

Proof. The statement is trivially true for recolorings (1) and (2). It is easy to observe that this really is corresponding to a simple recoloring. Hence, Properties 2 and 3 of Theorem 5 are also obviously true. By Lemma 7, both R and B are plane for the recolorings (3) and (4) and thus fulfill Property 1 of Theorem 5 as well. ◀

With these tools we now show how to construct two disjoint spanning trees. For technical reasons we use two different constructions based on the existence of a vertex in the minimum spanning tree where no two consecutive adjacent edges span an angle larger than π.

▶ **Theorem 9.** *Consider a set S of n points in the plane for which the minimum spanning tree $\mathsf{MST}(S)$ has a vertex v where between any two consecutive adjacent edges the angle is smaller than π. Then there exist two plane spanning trees $R = G(S, E_R)$ and $B = G(S, E_B)$ such that $E_R \cap E_B = \emptyset$ and $\max\{\mathsf{BE}(R), \mathsf{BE}(B)\} \leq 2\mathsf{BE}(\mathsf{MST}(S))$.*

Proof (sketch). When removing v from the tree, we obtain up to five connected components (assuming general position). For each of these, we individually re-add v and apply Construction 4 with v as the root using one of the variants of Lemma 8. This leaves some components of the tree disconnected, but this is resolved by adding additional edges from v to its neighbors and between adjacent neighbors of v. Full details of the construction and a proof can be found in the full version of this paper. ◀

The remaining case considers that for every vertex in $\mathsf{MST}(S)$ there exist two consecutive adjacent edges that span an angle larger than π. In such an $\mathsf{MST}(S)$, every vertex has degree at most three, since the angle between adjacent edges is at least $\pi/3$.

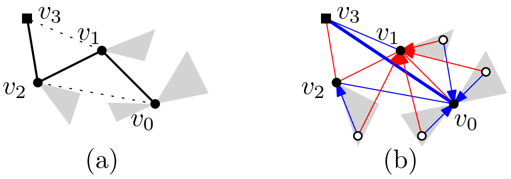


Figure 1 Illustration of one of the cases for the proof of Theorem 10. Grey subtrees indicate potential continuations of the MST and dashed edges indicate edges from $\mathsf{MST}^2(S)$. In (b) colored arrows indicate how the subtrees connect to P. Note that half of these arrows are from Construction 4.

▶ **Theorem 10.** *Consider a set S of $n \geq 4$ points in the plane for which every vertex in the minimum spanning tree $\mathsf{MST}(S)$ has two consecutive adjacent edges spanning an angle larger than π. Then there exist two plane spanning trees $R = G(S, E_R)$ and $B = G(S, E_B)$ such that $E_R \cap E_B = \emptyset$ and $\max\{\mathsf{BE}(R), \mathsf{BE}(B)\} \leq 3\mathsf{BE}(\mathsf{MST}(S))$ (where at most one edge of $E_R \cup E_B$ is larger than $2\mathsf{BE}(\mathsf{MST}(S))$).*

Proof (sketch). This case is dealt with using similar ideas as for Theorem 9. The main difference is that we now use a cluster $P = \{v_0, v_1, v_2, v_3\}$ of 4 points that are connected in $\mathsf{MST}(S)$ to serve as roots for up to three subtrees. The exact choice of P depends on the exact embedding of the tree, which leads to several potential embeddings of P and the subtrees of $\mathsf{MST}(S)$ attached to P. For this proof sketch we focus on one specific case shown in Figure 1, where v_3 is a leaf and v_2, v_1, v_0 form a path that, starting from v_3, takes a left and right turn. For ease of description we root the entire MST at v_3, creating parent and child relations between nodes. The subtrees we consider are T_0, T_1, T_2 defined as follows:

- T_0, consisting of v_1, v_0, and the subtrees rooted at the children of v_0, rooted at v_1.
- T_1, consisting of v_1, v_0 and the subtrees rooted at children of v_1, rooted at v_0.
- T_2, consisting of v_2, v_1 and the subtrees rooted at children of v_2, rooted at v_1.

Each of these trees is colored using one of the variants of Lemma 8, but we remove all edges going to the roots of the respective subtrees, leaving its children disconnected from P. We then re-attach them as follows. The roots of disconnected subtrees of T_0 are connected to v_1, those from T_1 are attached to v_0 and those from T_2 to v_1. By construction, the red and blue trees then form spanning trees with a maximum edge length of $3\mathsf{BE}(\mathsf{MST}(S))$ as all edges except v_0v_3 are part of $\mathsf{MST}^2(S)$, and v_0v_3 is part of $\mathsf{MST}^3(S)$. For planarity, non-crossing of edges that are not v_0v_3 follows relatively easily from Lemma 3 and Theorem 5. To see that v_0v_3 cannot be crossed, one can observe that by Lemma 1 the convex hull of P must be empty and from Lemma 2 and 3 it follows that no edge can cross the convex hull through v_3v_1 to v_1 or v_2. Full details of the construction and correctness arguments can be found in the full version of this paper. ◀

▶ **Corollary 11.** *For any set S of $n \geq 4$ points in the plane, there exist two plane spanning trees $R = G(S, E_R)$ and $B = G(S, E_B)$ such that $E_R \cap E_B = \emptyset$ and $\max\{\mathsf{BE}(R), \mathsf{BE}(B)\} \leq 3\mathsf{BE}(\mathsf{MST}(S))$.*

We now show that the above construction is worst-case optimal.

▶ **Theorem 12.** *For any $n > 3$ and $k > 1$ there exists a set of n points such that for any k disjoint spanning trees, at least one has a bottleneck edge larger than $(k+1)\mathsf{BE}(\mathsf{MST}(S))$.*

Proof. A counterexample simply consists of n points equally distributed on a line segment. (The points can be slightly perturbed to obtain general position.) In this problem instance

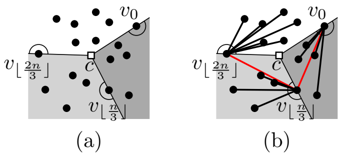

v_0
$v_{\lfloor \frac{2n}{3} \rfloor}$
c
$v_{\lfloor \frac{n}{3} \rfloor}$
(a)
v_0
$v_{\lfloor \frac{2n}{3} \rfloor}$
c
$v_{\lfloor \frac{n}{3} \rfloor}$
(b)

Figure 2 Extracting one layer: (a) The three sectors defined by v_0, $v_{\lfloor \frac{n}{3} \rfloor}$, and $v_{\lfloor \frac{2n}{3} \rfloor}$. (b) Connecting the points to the representative of their sector. The red edges connect the representatives.

there are $kn-(k(k+1)/2)$ edges whose distance is strictly less than $(k+1)\mathsf{BE}(\mathsf{MST}(S)) = k+1$. However, we need $kn-k$ edges for k disjoint trees and thus it is impossible to construct that many trees with sufficiently short edges. ◀

3 Distributed Approach

The previous construction relies heavily on the minimum spanning tree of S. It is well known that this tree cannot be constructed locally, thus we are implicitly assuming that the network is constructed by a single processor that knows the location of all other vertices. In ad-hoc networks, it is often desirable that each vertex can compute its adjacencies using only local information.

In the following, we provide an alternative construction. Although the length of the edges is increased by a constant factor, it has the benefit that it can be constructed locally and that it can be extended to compute k layers. The only global property that is needed is a value β that should be at least $\mathsf{BE}(\mathsf{MST}(S))$. We also note that these plane disjoint graphs are not necessarily trees, as large cycles cannot be detected locally.

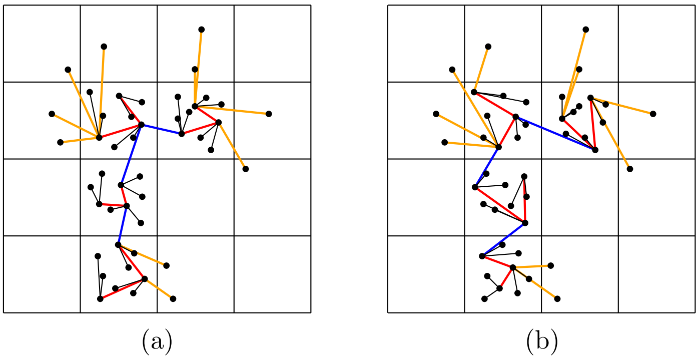
Before we describe our approach, we report the result of García [5] that states that every point set of at least $3k$ points contains k layers. Since the details of this construction are important for our construction and the manuscript is not yet available, we add a proof sketch.

▶ **Theorem 13** ([5]). *Every point set that consists of at least $3k$ points contains k layers.*

Proof. First, recall that for every set of n points, there is a *center point* c such that every line through c splits the point set into two parts that each contain at least $n/3$ points. For ease of explanation, we assume that every line through c contains at most one point. Number the points $v_0, v_1, \ldots, v_{n-1}$ in clockwise circular order around c. We split the plane into three angular regions by the three rays originating from c and passing through v_0, $v_{\lfloor \frac{n}{3} \rfloor}$, and $v_{\lfloor \frac{2n}{3} \rfloor}$, see Figure 2. Since every line through the center contains at least $n/3$ points on each side, the three angular regions are convex. We declare v_0 to be the representative of the angular region between the rays through v_0 and $v_{\lfloor \frac{n}{3} \rfloor}$ and connect the vertices $v_1, ..., v_{\lfloor \frac{n}{3} \rfloor}$ in this region to v_0. Similarly, we assign $v_{\lfloor \frac{n}{3} \rfloor}$ to be the representative of angle between the rays center through $v_{\lfloor \frac{n}{3} \rfloor}$ and $v_{\lfloor \frac{2n}{3} \rfloor}$ and connect vertices $v_{\lfloor \frac{n}{3} \rfloor+1}, ..., v_{\lfloor \frac{2n}{3} \rfloor}$ to $v_{\lfloor \frac{n}{3} \rfloor}$. Finally, we connect vertices $v_{\lfloor \frac{2n}{3} \rfloor+1}, ..., v_{n-1}$ to $v_{\lfloor \frac{2n}{3} \rfloor}$. This results in a non-crossing spanning tree.

For the second tree, we use v_1, $v_{\lfloor \frac{n}{3} \rfloor+1}$, and $v_{\lfloor \frac{2n}{3} \rfloor+1}$, and so on. ◀

While this construction provides a simple method of constructing the k layers, it does not give any guarantee on the length of the longest edge in this construction. To give such a guarantee, we combine it with a bucketing approach: we partition the point set using a grid (whose size will depend on k and β), solve the problem in each box with sufficiently many points independently, and then combine the subproblems to obtain a global solution (see Figure 3).



■ **Figure 3** The distributed approach: a grid is placed over the point set and different representatives construct different graphs ((a) and (b)). The red and black edges form the tree in each dense cell, blue edges connect the dense cells, and orange edges connect the vertices in sparse cells.

We place a grid with cells of height and width $6k\beta$ and classify the points according to which grid cell contains them (if a point lies exactly on the separating lines, pick an arbitrary adjacent cell). We say that a grid cell is a *dense box* if it contains at least $3k$ points of S. Similarly, it is a *sparse box* if it contains points of S but is not dense. We observe that dense and sparse boxes satisfy the following properties.

▶ **Lemma 14.** *Given two non-adjacent boxes B and B', the points in B and B' cannot be connected by edges of length at most β using only points from sparse boxes.*

Proof. Suppose the contrary and let B and B' be two dense boxes s.t. there is a path that uses edges of length at most β between a point in B to a point in B' visiting only points in sparse boxes. This path crosses the sides of a certain number of boxes in a given order; let σ be the sequence of these sides, with adjacent duplicates removed. Observe first that horizontal and vertical sides alternate in σ, as otherwise the path would have to use at least $6k - 1$ points to traverse a sparse box, but there are only at most $3k - 1$. Since B and B' are non-adjacent, w.l.o.g., there is a vertical side s that has two adjacent horizontal sides in σ with different y-coordinates. Hence, between the two horizontal sides, the corresponding part of the path has length at least $6k\beta$, and may use only the points in the two boxes adjacent to s. But since any sparse box contains at most $3k - 1$ points and the distance between two consecutive points along the path is at most β, that part of the path can have length at most $(6k - 1)\beta$, a contradiction. ◀

▶ **Corollary 15.** *Dense boxes are connected by the 8-neighbor topology.*

▶ **Lemma 16.** *Any set S of at least $4 \cdot (3k - 1) + 1$ points with $\beta \geq \mathsf{BE}(\mathsf{MST}(S))$ contains at least one dense box.*

Proof. Assume S consists of only sparse boxes. This implies that the points are distributed over at least five boxes, and thus, there is a pair of boxes that is non-adjacent. Using Lemma 14, this means that these boxes cannot be connected using edges of length at most $\mathsf{BE}(\mathsf{MST}(S))$, a contradiction. ◀

▶ **Lemma 17.** *In any set S of at least $4 \cdot (3k - 1) + 1$ points with $\beta \geq \mathsf{BE}(\mathsf{MST}(S))$, all sparse boxes are adjacent to a dense box.*

Proof. This follows from Lemma 14, since any sparse box that is not adjacent to a dense box cannot be connected to any dense box using edges of length at most $\beta \geq \mathsf{BE}(\mathsf{MST}(S))$. ◀

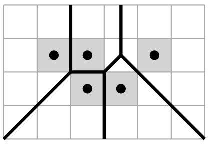


Figure 4 The Voronoi cells of the centers of the dense boxes.

Next, we assign all points to dense boxes. In order to do this, let c_B be the center of a dense box B. Note that c_B is not necessarily the center point of the points in this box. We consider the Voronoi diagram of the centers of all dense boxes and assign a point p to B if p lies in the Voronoi cell of c_B. Let S_B be the set of points of S that are associated with a dense box B. We note that each dense box B gets assigned at least all points in its own box, since in the case of adjacent dense boxes, the boundary of the Voronoi cell coincides with the shared boundary of these boxes (see Figure 4).

Furthermore, we can compute the points assigned to each box locally. By Lemma 17 all sparse boxes are adjacent to a dense box, and hence for any point p in a sparse box B its distance to its nearest center is at most $3\ell/\sqrt{2}$, where $\ell = 6k\beta$. It follows that only the centers of cells of neighbors and neighbors of neighbors need to be considered.

▶ **Lemma 18.** *For any two dense boxes B and B', we have that the convex hulls of S_B and $S_{B'}$ are disjoint.*

Proof. We observe that the convex hull of S_B is contained in the Voronoi cell of c_B. Hence, since the Voronoi cells of different dense boxes are disjoint, the convex hulls of the points assigned to them are also disjoint. ◀

For each dense box B, we apply Theorem 13 on the points inside the dense box to compute k disjoint layers of S_B. Next, we connect all sparse points in S_B to the representative of the sector that contains them in each layer. Since all points in the same sector connect to the same representative and the sectors of the same layer do not overlap, we obtain a plane graph for each layer within the convex hull of each S_B.

Hence, we obtain k pairwise disjoint layers such that in each layer the points associated to each dense box are connected. Moreover, since the created edges stay within the convex hull of each subproblem and by Lemma 18 those hulls are disjoint, each layer is plane. Thus, to assure that each layer is connected, we must connect the construction between dense boxes.

We connect adjacent dense boxes in a tree-like manner using the following rules:

- Always connect a dense box to the dense box below it.
- Always connect a dense box to the dense box to the left of it.
- If neither the box below nor the one to the left of it is dense, connect the box to the dense box diagonally below and to the left of it.
- If neither the box above nor the one to the left of it is dense, connect the box to the dense box diagonally above and to the left of it.

To connect two dense boxes, we find and connect two representatives p and q (one from each dense box) such that p lies in the sector of q and q lies in the sector of p; see Figure 5 (a).

▶ **Lemma 19.** *For any layer and any two adjacent dense boxes B and B', there are two representatives p and q in B and B', respectively, s.t. p lies in the sector of q and q lies in the sector of p.*

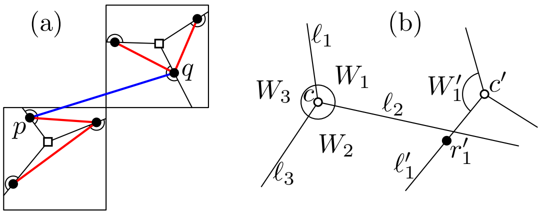


Figure 5 Connecting two dense boxes by means of p and q. The half-circles in (a) indicate which sector each representative covers. The red edges connect the dense boxes internally and the blue edge connects the two dense cells. (b) illustrates the sectors involved in connecting two neighboring dense boxes.

Proof. Consider two boxes B and B' with center points (of their respective point sets) c and c'. Now let W_1 and W_1' with representatives r_1 and r_1' denote the sectors containing c' and c, respectively; see Figure 5. The other sectors W_2 and W_3 of B with representatives r_2 and r_3 are ordered clockwise. We use ℓ_i to denote the ray from c containing r_i. If $r_1 \in W_1'$ and $r_1' \in W_1$ we are done. So assume that $r_1' \notin W_1$, the case when $r_1 \notin W_1'$ (or when both $r_1 \notin W_1'$ and $r_1' \notin W_1$) is symmetric. It follows that r_1' is in sector W_2 if the line segment $c'r_1'$ intersects ℓ_2 or sector W_3 if the segment intersects ℓ_2 and ℓ_3. Assume that r_1' is in sector W_2 (again the argument is symmetric when r_1' is in sector W_3). Now r_2 can be positioned on ℓ_2 between c and the intersection point with $c'r_1'$ or behind this intersection point when viewed from c. In the former case r_1' is in W_2 and r_2 is in W_1' and we are done. In the latter case the segments cr_2 and $c'r_1'$ cross. Since $c, r_2 \in B$ and $c', r_1' \in B'$ this crossing would imply that B and B' are not disjoint, a contradiction. ◀

Now that we have completed the description of the construction, we show that each layer of the resulting graph is plane and connected, and that the length of the longest edge is bounded.

▶ **Lemma 20.** *Each layer is plane.*

Proof. Since dense boxes are internally plane and the addition of edges to the sparse points do not violate planarity, it suffices to show that the edges between dense boxes cannot cross any previously inserted edges and that these edges cannot intersect other edges used to connect dense boxes.

We first show that the edge used to connect boxes B and B' is contained in the union of the Voronoi cells of these two boxes. If B and B' are horizontally or vertically adjacent, the connecting edge stays in the union of the two dense boxes, which is contained in their Voronoi cells. If B and B' are diagonally adjacent, we connect them only if their shared horizontal and vertical neighbors are not dense. This implies that at least the two triangles defined by the sides of B and B' that are adjacent to their contact point are part of the union of the Voronoi cells of these boxes. Hence, the edge used to connect B and B' cannot intersect the Voronoi cell of any other box. Since all points of a dense box in a sector connect to the same representative and these edges lie entirely inside the sector, the edge connecting two adjacent boxes can intersect only at one of the two representatives, but does not cross them. Therefore, an edge connecting two adjacent dense boxes by connecting the corresponding representatives cannot cross any previously inserted edge.

Next, we show that edges connecting two dense boxes cannot cross. Since any edge connecting two dense boxes stays within the union of the Voronoi cells of B and B', the only way for two edges to intersect is if they connect to the same box B and intersect in the Voronoi cell of B. If the connecting edges lie in the same sector of B, they connect to the same representative and thus they cannot cross. If they lie in different sectors of B, the edges lie entirely inside their respective sectors. Since these sectors are disjoint, this implies that the edges cannot intersect. ◀

▶ **Lemma 21.** *Each layer is connected.*

Proof. Since the sectors of the representatives of the dense boxes cover the plane, each point in a sparse box is connected to a representative of the dense box it is assigned to. Hence, showing that the dense boxes are connected, completes the proof.

By Corollary 15, the dense boxes are connected using the 8-neighbor topology. This implies that there is a path between any pair of dense boxes where every step is one to a horizontally, vertically, or diagonally adjacent box. Since we always connect horizontally or vertically adjacent boxes and we connect diagonally adjacent boxes when they share no horizontal and vertical dense neighbor, the layer is connected after adding edges as described in the proof of Lemma 19. ◀

▶ **Lemma 22.** *The distance between a representative in a dense box B and any point connecting to it is at most $12\sqrt{2}k\beta$.*

Proof. Since the representatives of B are connected only to points from dense and sparse boxes adjacent B, the distance between a representative and a point connected to it is at most the length of the diagonal of the 2×2 grid with B as one of its boxes. Since a box has width $6k\beta$, this diagonal has length $2\sqrt{2} \cdot 6k\beta = 12\sqrt{2}k\beta$. ◀

▶ **Theorem 23.** *For all point sets with at least $4(3k-1)+1$ points, we can extract k plane layers with the longest edge having length at most $12\sqrt{2}k\mathsf{BE}(\mathsf{MST}(S))$.*

Acknowledgements. We would like to thank all participants of the 10th European Research Week on Geometric Graphs for fruitful discussions.

References

1 O. Aichholzer, T. Hackl, M. Korman, M. J. van Kreveld, M. Löffler, A. Pilz, B. Speckmann, and E. Welzl. Packing plane spanning trees and paths in complete geometric graphs. In *Proc. Canadian Conference on Computational Geometry (CCCG)*, pages 233–238, 2014.

2 P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

3 D. Dor and M. Tarsi. Graph decomposition is NP-complete: A complete proof of Holyer's conjecture. *SIAM Journal on Computing*, 26(4):1166–1187, 1997.

4 M. Fussen, R. Wattenhofer, and A. Zollinger. Interference arises at the receiver. In *Proc. International Conference on Wireless Networks, Communications, and Mobile Computing (WIRELESSCOM)*, pages 427–432, 2005.

5 A. García. Personal Communication, 2015.

6 M. Korman. Minimizing interference in ad-hoc networks with bounded communication radius. *Information Processing Letters*, 112(19):748–752, 2012.

7 E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. Canadian Conference on Computational Geometry (CCCG)*, pages 51–54, 1999.

8 M. Priesler and M. Tarsi. Multigraph decomposition into stars and into multistars. *Discrete Mathematics*, 296(2-3):235–244, 2005.

9 M. Tarsi. Decomposition of a complete multigraph into simple paths: Nonbalanced handcuffed designs. *Journal of Combinatorial Theory, Series A*, 34(1):60–70, 1983.

Reconstruction of Weakly Simple Polygons from their Edges*

Hugo A. Akitaya[1] **and Csaba D. Tóth**[2]

1 **Tufts University, Medford, MA, USA**
hugo.alves_akitaya@tufts.edu

2 **Tufts University, Medford, MA, USA, and**
California State University Northridge, Los Angeles, CA, USA
cdtoth@acm.org

Abstract

Given n line segments in the plane, do they form the edge set of a *weakly simple polygon*; that is, can the segment endpoints be perturbed by at most ε, for any $\varepsilon > 0$, to obtain a simple polygon? While the analogous question for *simple polygons* can easily be answered in $O(n \log n)$ time, we show that it is NP-complete for weakly simple polygons. We give $O(n)$-time algorithms in two special cases: when all segments are collinear, or the segment endpoints are in general position. These results extend to the variant in which the segments are *directed*, and the counterclockwise traversal of a polygon should follow the orientation.

We study related problems for the case that the union of the n input segments is connected. (i) If each segment can be subdivided into several segments, find the minimum number of subdivision points to form a weakly simple polygon. (ii) If new line segments can be added, find the minimum total length of new segments that creates a weakly simple polygon. We give worst-case upper and lower bounds for both problems.

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

Keywords and phrases simple polygon, line segment, geometric graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.10

1 Introduction

In the design and analysis of geometric algorithms, the input is often assumed to be in general position. This is justified from the theoretical point of view: degenerate cases can typically be handled without increasing the computational complexity, or symbolic perturbation schemes can reduce any input to one in general position [4]. In this paper, we present a geometric problem about simple polygons in the plane, which has a straightforward solution if the input is in general position, but is NP-complete otherwise.

Suppose we are given n line segments in the plane. It is easy to decide in $O(n \log n)$ time whether they form a simple polygon by detecting intersections in a line sweep: if the segments are disjoint apart from common endpoints, then they form a plane graph, and a simple traversal can determine whether the graph is a cycle. If the input segments overlap, more than two segments have a common endpoint, or some segment endpoints lie in the interior of another segment, then they definitely do not form a simple polygon, but they

* Research on this paper was partially supported by NSF awards CCF-1422311 and CCF-1423615. The first author was supported by the Science Without Borders program.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 10; pp. 10:1–10:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

might still be perturbed into a simple polygon (i.e., they form a weakly simple polygon). We study the decision problem for weakly simple polygons in this paper.

Organization and Results. We start with necessary definitions, and formulate the problem of reconstructing a weakly simple polygons from a set of edges (Section 2). We present polynomial-time algorithms when the given segments form a geometric graph or are collinear (Sections 3). The problem in general, however, is strongly NP-hard by a reduction from Planar-Monotone-3SAT (Section 4). Nevertheless, every set of noncrossing line segments in the plane can be turned into the edge set of a weakly simple polygon by (i) subdividing the edges into several edges, or (ii) inserting new edges. In Sections 5 we show that if $G = (V, E)$ is Eulerian, the edges can be subdivided $O(n)$ times to obtain a weakly simple Euler tour. We also show that inserting new edges of total length at most $3\|E\|$ is always sufficient and sometimes necessary to create a weakly simple Euler tour. We conclude with future directions (Section 6). Omitted proofs are available in the full paper [2].

Related Work. Reconstruction of simple polygons from partial information (such as vertices, visibility graphs, visibility angles, cross sections) has been studied for decades [3, 6, 11, 12, 16]. For example, an orthogonal simple polygon can be uniquely reconstruction from its vertices [16], but if the edges have 3 or more directions, the problem becomes NP-hard [12]. For a simple polygon, the set of all edges (studied in this paper) gives complete information: the cyclic order of the edges is easy to recover. In contrast, a set of edges may correspond to exponentially many weakly simple polygons, and the reconstruction problem becomes nontrivial. The problems considered in Section 5 are closely related to geometric graph augmentation and subgraph problems: (i) Can a given plane straight-line graph be augmented with new edges into a simple polygon, a Hamiltonian plane graph, or a 2-connected plane graph [14, 17, 18, 19]? (ii) Does a given a geometric graph contain certain noncrossing subgraphs (e.g., spanning trees or perfect matchings) [15]?

2 Preliminaries

A *polygon* $P = (p_0, \ldots, p_{n-1})$ is a cyclic sequence of points in the plane (*vertices*), where every two consecutive vertices are connected by a line segment (*edge*). The cycle of edges can be parameterized by a piecewise linear curve $\gamma : \mathbb{S}^1 \to \mathbb{R}^2$. Polygon P is *simple* if γ is a *Jordan curve* (i.e., γ is injective); equivalently, if $(p_0, \ldots, p_{n-1})$ is the plane embedding of a Hamiltonian cycle. Polygon P is *weakly simple* if, for every $\varepsilon > 0$, the vertices p_i can be perturbed to points p'_i, $\|p_i p'_i\| < \varepsilon$, such that $P' = (p'_0, \ldots, p'_{n-1})$ is a simple polygon. The function $\|.\|$ denotes the Euclidean length of a line segment. Equivalently, a polygon given by γ is weakly simple if it can be perturbed to a Jordan curve $\gamma' : \mathbb{S}^1 \to \mathbb{R}^2$ such that the Fréchet distance of the two curves is bounded by ε (i.e., $\text{dist}_F(\gamma, \gamma') < \varepsilon$) [7]. We can test whether a polygon $P = (p_0, \ldots, p_{n-1})$, is simple or weakly simple, respectively, in $O(n)$ time [8] and $O(n \log n)$ time [1].

We define the WeaklySimplePolygonReconstruction (WSPR) problem as the following decision problem: Given a multiset E of line segments in $\mathbb{R}^2$, does there exist a weakly simple polygon P whose edge multiset is E? For a multiset E of *directed* segments, we also define Directed-WSPR that asks whether there exists a weakly simple polygon $P = (p_0, \ldots, p_{n-1})$ such that $\{p_i p_{i+1 \bmod n} : 0 \le i \le n-1\} = E$. In both undirected and directed variants, we represent the input segments as a straight-line multigraph $G = (V, E)$,

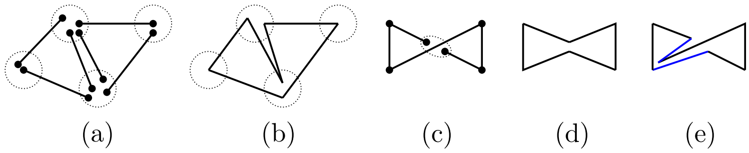


Figure 1 (a) A multi-set of line segments. Circles indicate common segment endpoints. (b) A weakly simple Euler tour. (c) An Eulerian graph that has no weakly simple Euler tour. An edge subdivision (d) or the insertion of two edges (e), yields a weakly simple Euler tour.

where V is the set of all segment endpoints. Note that G may have overlapping edges, and an edge may pass through vertices, so it need not be a *geometric graph*.

Two Necessary Conditions. Two line segments *cross* if they share exactly one interior point. If the multiset of segments E forms a weakly simple polygon, then no two segment cross. This condition can be easily tested in $O(|E| \log |E|)$ time by a line sweep.

If there is a weakly simple polygon $P = (p_0, \ldots, p_{n-1})$ with edge set E, then P is an Euler tour of the graph $G = (V, E)$. (However, an Euler tour need not be weakly simple; see Fig. 1(b)). A graph is Eulerian if and only if it is connected and every vertex has even degree. A *simple* (undirected) plane graph G is Eulerian if and only if its dual graph is bipartite. This result extends to plane *multi*graphs when an edge of multiplicity k is embedded as k interior-disjoint Jordan arcs, that enclose $k-1$ faces. A directed graph is Eulerian if and only if all vertices are part of the same strongly connected component and if, for each vertex, the in-degree equals the out-degree.

3 Special Cases

We show that both WSPR and Directed-WSPR admit polynomial-time algorithms in the special cases that (i) $G = (V, E)$ is a *simple geometric graph*, that is, no two edges overlap, and no vertex lies in the interior of an edge, and (ii) all edges in $G = (V, E)$ are collinear. We assume that G satisfies both necessary conditions.

3.1 Geometric Graphs

Note that in an Eulerian geometric graph the boundary of each face is a weakly simple circuit, where repeated vertices are possible, but there are no repeated edges. The following is a modified version of Hierholzer's algorithm [13]. It computes a weakly simple Euler tour P in the Eulerian graph G, or reports that no such tour exists.

Algorithm A (G)

```
1. 2-color the faces of $G$ white and gray so that the outer face is white; and create a list $L$
   of circuits on the boundaries of the gray faces.
2. If $G$ is directed and the edges around a gray face do not form a directed circuit or if there
   exist both clockwise (cw) and counterclockwise (ccw) circuits in $L$, report that $G$ has no
   weakly simple Euler tour.
3. Choose an arbitrary circuit in $L$, remove it from $L$ and call it $P$.
4. While there is a circuit in $L$, do:
   4.1 Find two consecutive edges, $(u, v)$ and $(v, w)$, along a white face such that $(u, v) \in P$
       and $(v, w) \in C$ for some $C \in L$.
```

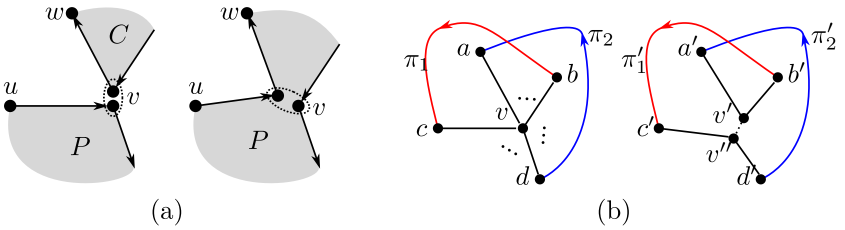


Figure 2 (a) Merging two cycles. The vertices circled by the dotted ellipse correspond to the same vertex v. (b) If v has two consecutive incoming edges (a, v) and (c, v), G does not admit a weakly simple Euler tour.

4.2 Remove C from the list L, and merge C and P by traversing C starting with the edge (v, w) followed by the traversal of P that ends with the edge (u, v); see Figure 2(a).

5. Return P.

▶ **Theorem 1.** *A simple geometric graph $G = (V, E)$ admits a weakly simple Euler tour if and only if G is Eulerian and, if G is directed, the circular order of edges around each vertex alternates between incoming and outgoing. A weakly simple Euler tour, if exists, can be computed in $O(|E|)$ time.*

Proof. If $G = (V, E)$ is undirected, the algorithm construct an Euler tour P [13]; and the tour is weakly simple by construction. In the remainder of the proof, we consider a directed Eulerian geometric graph G. First, we show that if G satisfies the conditions of Theorem 1, then Algorithm A returns a weakly simple Euler tour. If the circular order of edges around each vertex alternates between incoming and outgoing, then all edges on the boundary of a face of G have the same orientation (ccw or cw), and adjacent faces have opposite orientations. Without loss of generality, the edges on the boundaries of white (resp., gray) faces are oriented ccw (resp., cw). Hnece, the condition in step 2 of the algorithm is satisfied.

We show that the Euler tour P constructed by Algorithm A is weakly simple, that is, it can be perturbed into a simple polygon. Initially, each circuit $C = (p_0, \ldots, p_{k-1})$ in L is the boundary of a gray face, and hence it is a simple polygon. Let $C' = (p'_0, \ldots, p'_{k-1})$ be perturbation obtained by moving each point p_i to the interior of the face along an angle bisector of $\angle p_{i-1}p_ip_{i+1}$. Initially P is a weakly simple polygon (one of the circuits). It is enough to show that Step 4.2 maintains a weakly simple polygon, that is, when we merge P and a circuit C, their Jordan curve perturbations P' and C' can also be combined. Edges (u, v) and (v, w) are adjacent to a common white face f_0; they correspond to an edge (p_u, p_v) in P' and (q_v, q_w) in C', where both p_v and q_v lie in the ε-neighborhood of v in two different gray faces adjacent to f_0. We can modify P' and C' in the ε-neighborhood of v, by removing a short Jordan arc from each and reconnecting them across the white face f_0 into a single Jordan curve. By induction, we can obtain a Jordan curve within ε Frèchet distance from the output polygon P. Hence, the algorithm returns a weakly simple Euler tour P.

Now, we show that if G has a vertex v with two consecutive incoming (resp., outgoing) edges (a, v) and (c, v), then G does not admit a weakly simple Euler tour. Suppose, for contradiction, that there exists a weakly simple Euler tour P. Since both (a, v) and (c, v) are directed into v, the tour P contains edge-disjoint paths (a, v, b) and (c, v, d). Since P is weakly simple, the circular order of these four edges incident to v must be as shown in Figure 2(b). The polygon must contain edge-disjoint paths $\pi_1 = (v, b, \ldots, c, v)$ and $\pi_2 = (d, \ldots, a)$. The perturbation of π_1 is $\pi'_1 = (v', b', \ldots, c', v'')$ where $v' \neq v''$. Note that a and d are on opposite

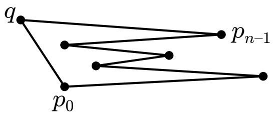


Figure 3 Every collinear Eulerian tour can be transformed in a y-monotonic simple polygon.

sides of the cycle $\pi' \cup v'v''$. The perturbation of π_2, path π_2', can intersect neither π_1' nor $v'v''$, because (a, v) and (c, v) are adjacent to the same face. Hence P is not weakly simple.

Finally, Algorithm A runs in $O(|E|)$ time. Step 1 and 2 can be done by traversing the dual graph of G. Step 4 executes $O(|E|)$ merges, each of which takes constant time. ◀

▶ **Corollary 2.** *A geometric multigraph $G = (V, E)$ admits a weakly simple Euler tour if and only if G is Eulerian. A weakly simple Euler tour, if exists, can be computed in $O(|E|)$ time.*

Proof. Replace every edge e of multiplicity k by k edge-disjoint paths of length two whose interior points are close to the midpoint of e. We obtain a simple Eulerian geometric graph with $|V| + 2|E|$ vertices. Theorem 1 completes the proof. ◀

▶ Remark. In the case that $G = (V, E)$ is a directed multigraph, replace every directed edge (u, v) of multiplicity k by edge-disjoint paths (u, w_i, v), with new (subdivision) vertices w_i, $i = 1, \ldots k$, and denote by G' the resulting simple directed graph. The alternating direction condition of Theorem 1 requires that the multiplicity of (u, v) and (v, u) differ by at most one. If their multiplicities differ by exactly one, then there is a unique way to interleave the replacement paths between u and v. In fact, if any edge of G has odd multiplicity, the alternating direction condition determines the cyclic order of all paths (u, w_i, v), and we can apply Theorem 1 for G'. If, however, all edges of G have even multiplicity, then there are two possibilities for the cyclic orders, both of which yield weakly simple Euler tours by Theorem 1.

3.2 Collinear Line Segments

▶ **Theorem 3.** *If all edges of a graph $G = (V, E)$ are collinear, then every Euler tour of G is a weakly simple polygon.*

Proof. Without loss of generality, assume that all vertices are on the x-axis. Let $\varepsilon > 0$ be given. Let $P = (p_0, \ldots, p_{n-1})$ be an Euler tour of G, and let p_0 be a leftmost vertex. For each vertex p_i, $i \in \{0, \ldots, n-1\}$, create a point p_i' with $x(p_i') = x(p_i)$ and $y(p_i') = i\varepsilon/(2n)$. The polygonal path $(p_0', \ldots, p_{n-1}')$ is strictly y-monotonic and therefore does not cross itself. The edge (p_{n-1}', p_0') can be realized as a one-bend polyline (p_{n-1}', q, p_0') with $q = (-\varepsilon/2, \varepsilon/2)$, which is outside of the axis-aligned bounding box of all other edges. Therefore $P' = (p_0', \ldots, p_{n-1}')$, illustrated in Figure 3, is a simple polygon where $\mathrm{dist}_F(P, P') < \varepsilon$. ◀

4 NP-Completeness

In this section we analyze the general case of WSPR. First we discuss the undirected case and then the direct version.

▶ **Lemma 4.** *Both WSPR and Directed-WSCR are in NP.*

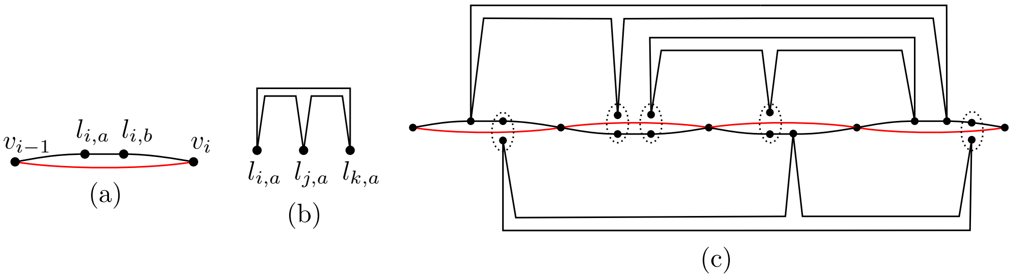


Figure 4 (a) Variable gadget, (b) clause gadget and (c) the reduction from Planar-Monotone-3SAT to undirected WSPR.

Proof. Given a polygon $P = (p_0, \dots, p_{n-1})$ and a (directed) straight-line multigraph $G = (V, E)$, we can check whether P is a (directed) Euler tour in G in $O(|E|)$ time, and whether P is weakly simple in $O(n \log n)$ time [1]. ◀

We prove that both directed and undirected WSPC are strongly NP-hard in the general case by a reduction from Planar-Monotone-3SAT, which is strongly NP-hard [5]. An instance of Planar-Monotone-3SAT consists of a plane bipartite graph G_B whose partite sets are variables nodes and clauses nodes. The variable nodes are on the x-axis, the clause nodes are above or below the x-axis; each clause is adjacent to three variables. A clause is *positive* if it lies above the x-axis, and *negative* otherwise. Planar-Monotone-3SAT asks if there is a binary assignment from {`true`,`false`} to the set of variables such that every positive clause is adjacent to at least one `true` variable and every negative clause is adjacent to at least one `false` variable.

▶ **Lemma 5.** *Undirected WSPR is NP-hard.*

Proof. Given an instance of Planar-Monotone-3SAT, we build an instance of undirected WSPR as shown in Figure 4(c). We split the construction into two basic gadgets. A variable and a clause gadget are shown in Figure 4(a) and (b), respectively. The figure shows collinear edges distorted and colored for clarity. All vertices shown as small black disks are on the x-axis and vertices circled with a dotted ellipse represent the same graph vertex.

First, place vertices $v_0, \dots, v_n$ equally spaced on the variable line from left to right. The variable gadget corresponding to the ith variable consists of two collinear paths between v_{i-1} and v_i, which are called *red* and *black* paths; see Figure 4. The red path is a single edge $v_{i-1}v_i$; and the black path is made of $p + 1$ edges where p is the degree of the i-th variable in the bipartite graph G_B. We assign a vertex in the interior of this path to each edge connected to the variable, naming the vertex $l_{i,a}$ for the edge connecting the i-th variable to the a-th clause. We call such vertices *literal* vertices. The clause gadget is composed of 9 edges arranged in a cycle as shown in Figure 4(b). The three labeled vertices correspond to the literal vertices in the clause gadgets. The planar embedding of the Planar-Monotone-3SAT instance grantees that we can embed the graph of the directed WSPR instance.

Assume that the Planar-Monotone-3SAT instance have a satisfying assignment. We build a weakly simple Euler tour P as follows. Each individual gadget defines a cycle. As in Algorithm A, we will merge the cycles into the polygon P. Every cycle will be traversed clockwise, however, cycles defined by variable gadgets are collinear and there is no clear definition of winding direction for them. We perturb the red edges based on the truth values of the variables. For each variable assigned `true` (resp., `false`), we perturb the red edge to pass below (resp., above) the x-axis. All variable cycles can be safely merged into a single

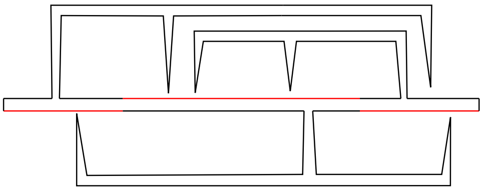


Figure 5 Simple polygon that certifies that an Euler tour of P is weakly simple.

circuit. We merge each clause to the variable cycle through a literal vertex of a `true` variable if the clause is positive or through a literal vertex of a `false` variable otherwise.

To show that P is weakly simple, we build a simple polygon P' within ε Fréchet distance from P as follows (see Figure 5). For each v_i create two vertices v_i^+ and v_i^- located $\varepsilon/2$ above and below v_i respectively. If the solution assigns the i-th variable `true`, move vertices $l_{i,a}$ up by $\varepsilon/2$, replace vertices v_{i-1} and v_i by v_{i-1}^+ and v_i^+ in the black edges (of the corresponding gadget) and by v_{i-1}^- and v_i^- in the red edges. Connect vertices v_0^+ and v_0^- with an edge. Do the same for v_n^+ and v_n^-. If the variable is assigned `false`, do analogous replacements symmetrically about the x-axis. For each clause gadget, choose a literal $l_{i,a}$ with a `true` value, split $l_{i,a}$ into two vertices, $l'_{i,a}$ and $l''_{i,a}$, with the same y-coordinate and ε distance apart, such that they each are incident to one edge of the variable gadget and one edge of the clause gadget. For the other two literals, split $l_{i,a}$ into two vertices, $l_{i,a}^+$ and $l_{i,a}^-$, with the same x-coordinate and $\frac{\varepsilon}{2}$ distance apart, such that the one closer to the x-axis is incident to two edges of the variable gadget, and the other to two edges of the clause gadget. The result is a simple polygon and therefore undirected WSPR have a positive solution.

Now assume that the graph produced by the reduction admits a weakly simple Euler tour P. Then, there exist a simple polygon P' within an arbitrarily small Fréchet distance from P. Such a polygon determines a vertical order between the paths of each variable gadget. Since every literal vertex has degree 4, there are only two possible ways to match its incident edges in a noncrossing manner: matching two horizontal edges and two clause edges, or a horizontal with a clause edge. In both cases, the two horizontal black edges incident to a literal vertex are placed above or below the red path. Therefore, all edges of the black path of a variable gadget are on the same side of its red path. For each variable, assign `true` if the black path of its gadget is above the red path and `false` otherwise. Since each clause gadget needs to be connected to some edge in a variable gadget, if the clause is positive/negative, one of its corresponding variables were assigned `true`/`false`. Hence, the assignment satisfies all clauses and the PLANAR-MONOTONE-3SAT instance have a positive solution. ◀

▶ **Lemma 6.** *Directed-WSPR is NP-hard.*

As a consequence of Lemmas 4, 5, and 6, we have the following result.

▶ **Theorem 7.** *Both WSPR and Directed-WSPR are NP-complete.*

▶ Remark. Our reduction can be modified by perturbing the points in our variable gadgets so that: (i) points belonging to the same gadget are collinear; (ii) no three points, each belonging to a different gadget are collinear; and (iii) no edge crossing is introduced. By reducing from PLANAR-MONOTONE-(2,3)-SAT-3 [10], in which clauses may have two or three literals and each variable can appear only in up to three clauses, we can show that WSPR remain NP-hard even if the number of mutually collinear points is constant.

5 Related problems

Since WSPR is NP-complete in the general case, we study related problems in which a weakly simple polygon is always achievable by allowing edge subdivision and insertion of new edges.

5.1 Edge subdivision

Given a noncrossing graph $G = (V, E)$ where every vertex has even degree and the point set $\bigcup E$ is connected, we define the problem WSPR* as finding a sequence of edge subdivision operations that produces a graph $G^* = (V, E^*)$ that admits a weakly simple Euler tour. The *subdivision* of an edge uv at a vertex $w \in \text{relint}(uv)$ replaces uv by two edges uw and wv.

It is easy to see that WSPR* is always feasible with $O(n^2)$ subdivisions where $n = |V|$. Indeed, subdivide every edge uv recursively at each vertex that lies in the interior of uv. We obtain a connected geometric multigraph with even degrees, which admits a weakly simple Euler tour by Corollary 2. The main result of this section is the following.

▶ **Theorem 8.** *Every noncrossing graph $G = (V, E)$ such that every $v \in V$ has even degree and $\bigcup E$ is connected, can be transformed into a graph $G^* = (V, E^*)$ using $O(|E|)$ edge subdivisions, and this bound cannot be improved.*

Before the proof, we introduce some notation (from [1, 7, 9]). Let $G = (V, E)$ be a noncrossing graph. The transitive closure of the *overlap* relation is an equivalence relation on E. The union of all edges in an equivalence class is called a *bar*, it is a line segment. A vertex $v \in b$ is called *b-odd* if v is incident to an odd number of edges contained in b, or *b-even* otherwise. A vertex can be b-odd and b'-even for different bars b and b' (see Figure 6(b)).

Our algorithm will compute simple paths formed by subdivided edges. Let b be a horizontal bar with vertices $p_1, p_2 \in b$, $x(p_1) \le x(p_2)$. Let $q_1q_2 \in E$ be an edge that contains p_1 and its right endpoint has minimum x-coordinate. A *subdivided paths*, denoted by $\widehat{p_1p_2}$, is a path between p_1 and p_2, defined recursively (see Fig. 6(d)): (i) if $x(p_1) = x(p_2)$, $\widehat{p_1p_2} = \emptyset$; (ii) if $p_2 \in q_1q_2$, then subdivide q_1q_2 into three edges $e_1 = q_1p_1$, $e_2 = p_1p_2$, and $e_3 = p_2q_2$ and put $\widehat{p_1p_2} = (e_2)$; (iii) if $p_2 \notin q_1q_2$, then subdivide q_1q_2 into two edges $e_1 = q_1p_1$ and $e_2 = p_1q_2$, and put $\widehat{p_1p_2} = (e_2) \oplus \widehat{q_2p_2}$, where $\oplus$ denotes concatenation. Consequently, if the segment p_1p_2 contains k vertices, a path $\widehat{p_1p_2}$ can be constructed using at most k edge subdivisions. An example is shown in Figure 6(c).

Proof of Theorem 8. The proof of the upper bound is constructive. The algorithm subdivides edges within each bar independently. Let b be a bar containing m vertices. We apply $O(m)$ edge subdivisions and partition the edges in b into subsets: Subsets $\mathcal{M}^+$ and $\mathcal{M}^-$ will consists of subdivision paths between the intersection points of b with other bars lying above and below b, respectively; all remaining edges will be partitioned into tours (each of which is a weakly simple polygon by Theorem 3). The algorithm is divided into three phases: Phase 1 creates $\mathcal{M}^+$ and $\mathcal{M}^-$; phase 2 forms circuits; and phase 3 establishes common vertices between the subdivision paths and circuits. Refer to Figure 6.

Phase 1. Compute a list B^+ (resp., B^-) of points p in the interior of b such that p is b'-odd for some bar b' that is above or collinear to b (resp., below b). A point can appear more than once in each list if it is odd in multiple bars b'. Sort the lists by $x(p)$, ties are broken by clockwise (resp., counterclockwise) order of the corresponding bars b'. If the left (resp., right) endpoint of b is b-odd, add it to the beginning (resp., end) of the list B^+. If any of the lists have odd cardinality, append the right endpoint of b at the end of the list. Create a

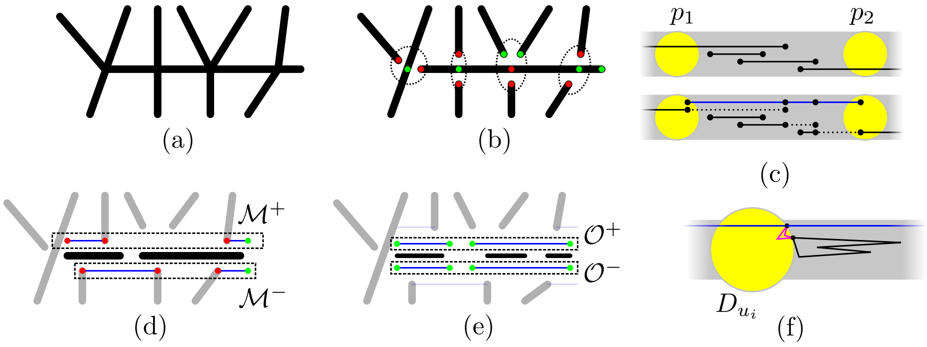


Figure 6 (a) A bar b and its adjacent bars. (b) Each bar b' is shown with its b'-odd and b'-even vertices shown in red and green respectively. (c) The subdivided path $\widehat{p_i p_2}$ is shown in blue. Examples of (d) $\mathcal{M}^+$ and $\mathcal{M}^-$; (e) $\mathcal{O}^+$ and $\mathcal{O}^-$. (f) Connecting a component of $\mathcal{B}'$ to a path in $\mathcal{O}^+$ with two polygonal paths shown in magenta.

perfect matching of consecutive endpoints in each list. Construct edge disjoint subdivided paths between each pair of matched points, and let $\mathcal{M}^+$ and $\mathcal{M}^-$ denote the set of edges in such paths for B^+ and B^-, respectively (see Figure 6(d)).

Phase 2. Let $\mathcal{B}$ be the set of (subdivided) edges that lie on b and are not in $\mathcal{M}^+ \cup \mathcal{M}^-$. The union of edges in $\mathcal{B}$ may be a disconnected point set (e.g., as shown in Figure 6(d)). Let the line segment r_1r_2 be one of the connected components of $\bigcup \mathcal{B}$. Construct two edge disjoint subdivided paths $\widehat{r_1r_2}^+$ and $\widehat{r_1r_2}^-$ from the edges in $\mathcal{B}$. For every path $\widehat{p_1p_2}$ in $\mathcal{M}^+$ (resp., $\mathcal{M}^-$) that overlaps with r_1r_2, identify an edge of $\widehat{r_1r_2}^+$ (resp., $\widehat{r_1r_2}^-$) that contains a vertex of $\widehat{p_1p_2}$ and subdivide it at such vertex (see Figure 6(e)). Let $\mathcal{O}^+$ (resp., $\mathcal{O}^-$) be the set of edges in $\widehat{r_1r_2}^+$ (resp., $\widehat{r_1r_2}^-$) for all components r_1r_2 of the union of edges in $\mathcal{B}$.

Phase 3. Let $\mathcal{B}'$ be the set of edges in $\mathcal{B} \setminus (\mathcal{O}^+ \cup \mathcal{O}^-)$. For every component C of the subgraph induced by $\mathcal{B}'$, let p_0 be the leftmost vertex of C, identify the edge in $\mathcal{O}^+$ that contains p_0 and subdivide it at p_0. This concludes the construction of G^*.

Correctness. Now we prove that G^* admits a weakly simple Euler tour. Notice that G^* is connected since the subdivisions in phase 2 connects every component of $\mathcal{M}^+$ or $\mathcal{M}^-$ to every overlapping component of $\mathcal{O}^+$ or $\mathcal{O}^-$, and phase 3 connects every component of $\mathcal{B}'$ to some component in $\mathcal{O}^+$. Since edge subdivisions do not change the parity of degrees, every vertex in G^* has even degree, hence G^* is Eulerian. We construct an Eulerian geometric graph G' such that every Euler tour in G' is within $\varepsilon/2$ Fréchet distance from an Euler tour in G^*. Theorem 1 will then imply that there exists a simple polygon within ε Fréchet distance from an Euler tour in G^*.

We recall some notation introduced in [7]. For every vertex $v \in V$, let D_v be a disk centered at v of radius $\frac{\varepsilon}{4}$. For a bar b between u_0 and u_k, let D_b be the ε^2 neighborhood of b setminus $D_{u_0} \cup D_{u_k}$. Assume that $\varepsilon \in (0, \frac{1}{4})$ is so small that the disks D_v are pairwise disjoint; a disk D_v intersects D_b only if $v \in b$, and the neighborhoods D_b are pairwise disjoint.

For each bar b with vertices $u_0, \ldots, u_k$, we perturb the edges of E^* contained in b into noncrossing simple polygons and polygonal chains. Embed each subdivided path in $\mathcal{M}^+$ (resp., $\mathcal{M}^-$) $\widehat{u_iu_j}$ in the upper (resp., lower) boundary of the region D_b such that u_i is on the boundary of D_{u_i} and u_j is on the boundary of D_{u_j}. Subdivide D_b with $\ell + 1$ horizontal

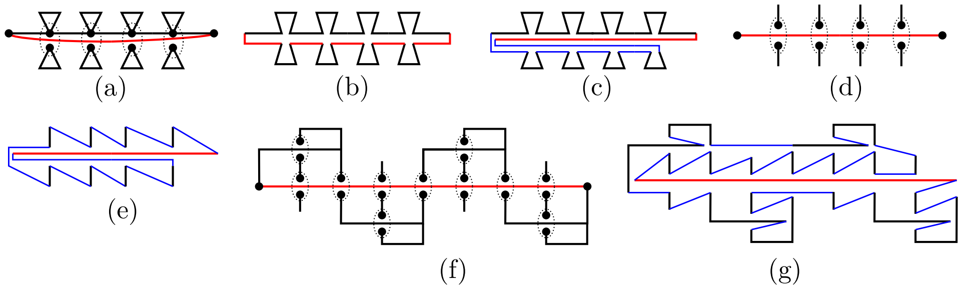


Figure 7 Lower bound constructions.

lines where ℓ is the number of components of the subgraph induced by $\mathcal{B}'$. Embed all edges in $\mathcal{O}^+$ (resp., $\mathcal{O}^-$) in the first (resp., $\ell+1$-th) such line.

Recall that every vertex of G has even degree. If b contains a b-odd vertex p, there must exist a bar b' such that p is b'-odd. Because $\mathcal{M}^+$ and $\mathcal{M}^-$ matches such points, the subgraph induced by $\mathcal{B}$ contain only even degree vertices. By construction the edges in $\mathcal{O}^+ \cup \mathcal{O}^-$ form nonoverlapping disjoint circuits. Hence, the subgraph induced by $\mathcal{B}'$ contains only even degree vertices. Consequently, each of its components is Eulerian and forms a weakly simple polygons that we denote by $\gamma_1(b), \ldots, \gamma_\ell(b)$, sorted by the x-coordinates of their left endpoints. Perturb $\gamma_1(b), \ldots, \gamma_\ell(b)$ into simple polygons that lie in the interior of D_b, separated by one of the $\ell+1$ lines, in this linear order (ties are broken arbitrarily). For $i = 0, \ldots, k$, consider all polygons $\gamma_j(b)$ whose leftmost vertex is u_i. Connect the left endpoints of each such $\gamma_j(b)$ to the copy of u_i in $\mathcal{O}^+$ by two polygonal paths within D_{u_i} (these paths connect different copies of vertex $u_i \in V^*$, see Figure 6(f)). Similarly, for each subdivision performed in phase 2 at u_i of an edge in $\mathcal{O}^+$ (resp., $\mathcal{O}^-$) in a path $\widehat{r_1 r_2}$, connect the copy of u_i in this path to a copy in $\mathcal{M}^+$ (resp., $\mathcal{O}^-$) by two polygonal paths within the disk D_{u_i}. Connect the endpoints of the overlapping paths in $\mathcal{O}^+$ and $\mathcal{O}^-$ (forming a cycle of each), and if the right endpoint of b is not b-odd and was added to B^+, B^-, connect the copies of u_k in $\mathcal{M}^+$ and $\mathcal{M}^-$. For each matching in B^+ and B^- involving a point u_i that is a b'-odd endpoint of a bar b', connect the paths in $\mathcal{M}^+$ or $\mathcal{M}^-$ that correspond to a match in b to the path in $\mathcal{M}^+$ of b' that contains p. Finally, for each point u_i that is the endpoint of a bar b' and is b'-even, connect the corresponding copies of u_i, making the graph induced by all edges containing a point on b connected. This concludes the construction of G'.

Theorem 1 completes the proof: An Euler tour $\widehat{P}$ of G' can be perturbed into a simple polygon P such that $\text{dist}_F(P, \widehat{P}) < \frac{\varepsilon}{2}$. The tour $\widehat{P}$ maps to an Euler tour P^* of G^* by identifying the vertices that lie in the same disk D_v, $v \in V^*$; and $\text{dist}_F(\widehat{P}, P^*) < \frac{\varepsilon}{2}$.

Our lower bound construction is shown in Figure 7(a). It consists of a graph $G = (V, E)$ containing a long edge e_R (shown in red) and a path of $(|E|+5)/7$ non-overlapping collinear edges that connects the endpoints of e_R. Each vertex in the interior of e_R is also incident to two small cycles above and below e_R respectively. Although the graph is Eulerian, it does not admit a weakly simple Euler tour. Each vertex p in the interior of the red edge e_R is incident to two small triangles. Suppose that e_R is *not* subdivided at p. Then p has degree 6. In any perturbation of a weakly simple Euler tour, vertex p is split into 3 copies, each of degree 2, and each lying above or below e_R. Suppose only one copy of p lies below e_R. Then it is incident to two edges of a small triangle below e_R, which is then disconnected from the rest of the graph, a contradiction. Consequently, e_R must be subdivided at all interior vertices. Figure 7(b) shows that $O(|E|)$ subdivisions of e_R suffice in this case. ◀

5.2 Edge insertion

We define the problem WSPR$^+$ as finding a set of edges E^+ such that $G^+ = (V, E \cup E^+)$ admits a weakly simple Euler tour. Denote by $\|E\|$ and $\|E^+\|$, respectively, the sum of the lengths of all edges in E and E^+. If the point set $\bigcup E$ is disconnected, then there is no upper bound on $\|E^+\|$. Otherwise, we can establish worst-case upper and lower bounds for $\|E^+\|$ in terms of $\|E\|$.

▶ **Theorem 9.** *Let $G = (V, E)$ be a noncrossing multigraph such that $\bigcup E$ is a connected point set. Then there exists a set of line segments E^+ such that $\|E^+\| \le 3\|E\|$ and $G^+ = (V, E \cup E^+)$ admits a weakly simple Euler tour.*

Proof. We construct E^+ as follows. Partition E into *bars* (equivalence classes of the transitive closure of the overlap relation on E). Denote by b the union of edges in a bar. W.l.o.g., we may assume that b is horizontal. Denote by $u_0, \ldots, u_k \in V$ the vertices of V along b sorted by x-coordinates (where $b = u_0u_k$). For $i = 1, \ldots, k$, add an edge $u_{i-1}u_i$ to E^+ if the edges of E in the bar cover the line segment $u_{i-1}u_i$ an odd number of times. The old and new edges in the bar b jointly form a graph of even degree that we denote by $G(b)$. By Theorem 3, every component of $G(b)$ admits a weakly simple Euler tour. Finally, add two more copies of edge $u_{i-1}u_i$ to E^+ for all $i = 1, \ldots k$. After repeating the above steps for every bar, we have $\|E^+\| \le 3\|E\|$ and $G^+ = (V, E \cup E^+)$ is Eulerian.

We omit the proof of correctness (which is provided in the full paper [2]), that shows that G^+ admits a weakly simple Euler tour, since it is similar to the proof of Theorem 8. ◀

Lower bound constructions. All our lower bound constructions are graphs $G = (V, E)$ in which an edge connects two points on the boundary of the convex hull of V, denoted ch(V).

▶ **Theorem 10.** *Let $\mathcal{G}$ be a family of noncrossing multigraphs. For $G = (V, E) \in \mathcal{G}$, let E^+ be an edge set of minimum length $\|E^+\|$ such that $G^+ = (V, E \cup E^+)$ admits a weakly simple Euler tour; and let $\lambda(\mathcal{G}) = \sup_{G \in \mathcal{G}} \|E^+\|/\|E\|$. Then:*

1. *$\lambda(\mathcal{G}_1) \ge 1$, where $\mathcal{G}_1 = \{$Eulerian noncrossing multigraphs$\}$.*
2. *$\lambda(\mathcal{G}_2) \ge \frac{6}{5}$, where $\mathcal{G}_2 = \{$connected noncrossing multigraphs$\}$.*
3. *$\lambda(\mathcal{G}_3) \ge 3$, where $\mathcal{G}_3 = \{$noncrossing multigraphs $G = (V, E)$ such that $\bigcup E$ is connected$\}$.*

Proof.

(1) Refer to Figs. 7(a)–(c). Let $n \in \mathbb{N}$ and $\delta \in (0, \frac{1}{3})$. Place vertices $v_i = (i, 0)$, for $i = 0, \ldots, n$, on the x-axis. A red edge of length n connects v_0 and v_n. A black edge of length $1/n$ connects v_{i-1} and v_i for $1 \le i \le n$. A small cycle of length $\delta < \frac{1}{3}$ is placed on each v_i, $1 \le i \in n - 1$, on each side of the x-axis. The total length of the construction is $\|E\| = 2n + 2(n-1)\delta$.

Let $G^+ = (V, E \cup E^+)$ be a multigraph in which P is a weakly simple Euler tour; and let P' be an ε-perturbation into a simple polygon, for some $0 < \varepsilon < \delta$. We define a pair of vertical lines $\ell_i^- : x = i + \delta$ and $\ell_i^+ : x = (i+1) - \delta$, for $0 \le i \le n-1$. The portion of P' between any two of these lines consists of disjoint paths whose endpoints are on the lines. By Morse theory, P' contains an even number of paths between any two of these lines; and the length of such a path is at least the distance between the parallel lines. The input edges already contain two line segments between any two of these lines: a red and a black segment.

We claim that G^+ contains at least 4 paths between ℓ_i^- and ℓ_i^+ for all but at most one index $0 \le i \le n-1$. Indeed, suppose that there are two such paths between ℓ_i^- and ℓ_i^+ and between ℓ_j^- and ℓ_j^+ $(0 \le i < j \le n-1)$. We may assume w.l.o.g. that the black edge is above the red edge between ℓ_i^- and ℓ_i^+. Then the black edge must be above the red edge between ℓ_j^- and ℓ_j^+, as well. Consequently, P' cannot reach the small cycles at $v_{i+1}, \dots, v_j$. This confirms the claim. It follows that $\|E \cup E^+\| \geq (4n-2)(1-2\delta)$. This lower bound tends to $2\|E\|$ as $n \to \infty$ and $\delta n \to 0$.

Due to space restrictions, we omit the proofs for cases 2 and 3. ◀

6 Conclusions

We have shown that WSPR is NP-complete. It follows that the decision version of the problems in Section 5 are also NP-complete: It is NP-complete to find up to k subdivision points to form a weakly simple polygon, or to find an edge set with length up to k that produce a weakly simple polygon. We have shown that $\Theta(|E|)$ subdivision points are always sufficient and sometimes necessary when the input is Eulerian; and new edges of length $\Theta(\|E\|)$ are always sufficient and sometimes necessary when $\bigcup E$ is connected. However, the best constant coefficients are not known in most cases. We conjecture that every noncrossing Eulerian graph $G = (V, E)$ can be augmented into a graph $G^+ = (V, E \cup E^+)$ that admits a weakly simple Euler tour such that $\|E^+\| \leq \|E\|$.

If the segments in E do not form a weakly simple polygon, we can subdivide segments or insert new segments to create a weakly simple polygon. On the other end of the spectrum, a set of n line segment may form an exponential number of weakly simple polygons, even if all segments are collinear. It is an open problem to count exactly how many weakly simple polygons can be obtained from the same set of line segments. Finally, we mention an open problem about reconstructing *simple polygons* from a subset of its edges. It is NP-complete to decide whether a geometric graph $G = (V, E)$ can be augmented into a *simple* polygon $P = (V, E \cup E^+)$ [17]. However, it is not known whether the problem remains NP-hard when G is a perfect matching.

Acknowledgements. We thank Adrian Dumitrescu for bringing this problem to our attention.

References

1 Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba Tóth. Recognizing weakly simple polygons. In *Proc. 32nd Int. Sympos. Comput. Geom.*, volume 51 of *LIPIcs*, Dagstuhl, 2016.

2 Hugo A. Akitaya and Csaba D. Tóth. Reconstruction of weakly simple polygons from theirs edges. Preprint, arXiv, 2016.

3 Gill Barequet, Craig Gotsman, and Avishay Sidlesky. Polygon reconstruction from line cross-sections. In *Proc. 18th Canadian Conf. Comput. Geom.*, Kingston, ON, 2006.

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

5 Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. *Internat. J. Comput. Geom. Appl.*, 22(3):187–205, 2012.

6 Therese Biedl, Stephane Durocher, and Jack Snoeyink. Reconstructing polygons from scanner data. *Theoret. Comput. Sci.*, 412(32):4161–4172, 2011.

7 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms*, pages 1655–1670. SIAM, 2015.

8 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(3):485–524, 1991.

9 Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. On embedding a cycle in a plane graph. *Discrete Math.*, 309(7):1856–1869, 2009.

10 Andreas Darmann, Janosch Döcker, and Britta Dorn. On planar variants of the monotone satisfiability problem with bounded variable appearances. Preprint, `arXiv:1604.05588`, 2016.

11 Yann Disser, Matúš Mihalák, and Peter Widmayer. A polygon is determined by its angles. *Comput. Geom. Theory Appl.*, 44(8):418–426, 2011.

12 Michael Formann and Gerhard J. Woeginger. On the reconstruction of simple polygons. *Bulletin EATCS*, 40:225–230, 1990.

13 Carl Hierholzer and Chr Wiener. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

14 Michael Hoffmann and Csaba D Tóth. Segment endpoint visibility graphs are Hamiltonian. *Comput. Geom. Theory Appl.*, 26(1):47–68, 2003.

15 Klaus Jansen and Gerhard J Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT Numerical Mathematics*, 33(4):580–595, 1993.

16 Joseph O'Rourke. Uniqueness of orthogonal connect-the-dots. In Godfried T. Tousaint, editor, *Computational Morphology*, pages 97–104. North-Holland, 1988.

17 David Rappaport. Computing simple circuits from a set of line segments is NP-complete. *SIAM J. Comput.*, 18(6):1128–1139, 1989.

18 Csaba D Tóth. Connectivity augmentation in planar straight line graphs. *European J. Combin.*, 33(3):408–425, 2012.

19 Masatsugu Urabe and Mamoru Watanabe. On a counterexample to a conjecture of Mirzaian. *Comput. Geom. Theory Appl.*, 2(1):51–53, 1992.

Approximating Smallest Containers for Packing Three-Dimensional Convex Objects*

Helmut Alt[1] **and Nadja Scharf**[2]

1 **Institute of Computer Science, Freie Universität Berlin, Berlin, Germany**
`alt@mi.fu-berlin.de`

2 **Institute of Computer Science, Freie Universität Berlin, Berlin, Germany**
`nadja.scharf@fu-berlin.de`

Abstract

We investigate the problem of computing a minimum-volume container for the non-overlapping packing of a given set of three-dimensional convex objects. Already the simplest versions of the problem are $\mathcal{NP}$-hard so that we cannot expect to find exact polynomial time algorithms. We give constant ratio approximation algorithms for packing axis-parallel (rectangular) cuboids under translation into an axis-parallel (rectangular) cuboid as container, for packing cuboids under rigid motions into an axis-parallel cuboid or into an arbitrary convex container, and for packing convex polyhedra under rigid motions into an axis-parallel cuboid or arbitrary convex container. This work gives the first approximability results for the computation of minimum volume containers for the objects described.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases computational geometry, packing, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.11

1 Introduction

The problem of efficiently packing objects without overlap arises in a large variety of contexts. Apart from the obvious ones, where concrete objects need to be packed for transportation or storage, there are more abstract ones, for example cutting stock or scheduling. Given a set of objects that have to be cut out from the same material the objective is to minimize the waste, i.e., place the pieces to be cut out as close as possible. In the case of scheduling, a list of jobs is given. Each job needs a certain amount of given resources and the aim is to minimize under certain constraints this need of resources such as time, space, or number of machines. Altogether, this situation can be described as a problem of packing high-dimensional cuboids into a strip with bounded side lengths. So, both problems can be viewed as a given list of objects for which a container of minimum size is wanted.

In this work, we consider the more general and abstract problem of packing three-dimensional convex polyhedra into a minimum volume container. All variants of this problem are $\mathcal{NP}$-hard and we will develop constant factor approximation algorithms for some of them. The worst case constant factors are still very high, but probably they will be much lower for realistic inputs. The major aim of this paper, however, is to show the existence of constant factors at all, i.e., that the problems belong to the complexity class APX.

* This work was partially supported by DFG (Deutsche Forschungsgemeinschaft) under grant no. AL253/7-2.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 11; pp. 11:1–11:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Work

So far, there are only few results about finding containers of minimum volume. Related problems include strip packing and bin packing. In two-dimensional strip packing the width of a strip is given and the objects should be packed in order to minimize the length of the strip used. In three dimensions, the rectangular cross section of the strip is fixed. Bin-packing is the problem where the complete container is fixed and the objective is to minimize the number of containers to pack all objects. For both problems usually only translations are allowed to pack the objects.

For two-dimensional bin packing there exists an algorithm with an asymptotic approximation ratio of 1.405 [3] and Bansal et al. proved that there cannot be an APTAS unless $\mathcal{P} = \mathcal{NP}$ [2]. For two-dimensional strip packing there exists an AFPTAS [7]. In three dimensions there are algorithms with an asymptotic approximation ratio of 4.89 for bin packing [9] and an asymptotic approximation ratio of $\frac{3}{2} + \varepsilon$ for strip packing [6]. The best known worst case (non-asymptotic) approximation ratio for three-dimensional strip packing is $\frac{29}{4}$ [5].

For two dimensions, von Niederhäusern [11] gave algorithms for packing rectangles or convex polygons in a minimum-area rectangular container with approximation ratios 3 and 5 respectively. A recent result shows that packing convex polygons under translation into a minimum-area rectangular or convex container can be approximated with ratios 17.45 and 27 respectively [1].

PARTITION can be reduced to one-dimensional bin packing and one-dimensional bin packing is a special case of higher dimensional bin or strip packing. If one-dimensional bin packing could be approximated with a ratio smaller than $\frac{3}{2}$, we could solve PARTITION. Therefore, none of the mentioned problems can be approximated better than with ratio $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$. PARTITION can also be reduced to our problem showing $\mathcal{NP}$-hardness.

Our Results

In this work we give the first approximation results for packing three-dimensional convex objects in a minimum-volume container. For packing axis-parallel rectangular cuboids under translation into an axis-parallel rectangular cuboid as a container, we achieve a $7.25 + \varepsilon$ approximation. If we allow the cuboids to be packed under rigid motions (translation and rotation) then we achieve an approximation ratio of 17.737 for an axis-parallel cuboid as container and an approximation ratio of 29.135 for an arbitrary convex container. For packing convex polyhedra under rigid motions we achieve an approximation ratio of 277.59 for computing an axis-parallel cuboid as container and 511.37 for a convex container.

2 Preliminaries and Reduction to Strip Packing

For most algorithms considered here, the input is a set of rectangular boxes $\mathcal{B} = \{b_1, b_2, \ldots b_n\}$. We denote a box b_i in axis-parallel orientation by a tuple of its height, width and depth (h_i, w_i, d_i). We denote by $h_{\max} = \max\{h_i \mid b_i \in \mathcal{B}\}$, $w_{\max} = \max\{w_i \mid b_i \in \mathcal{B}\}$ and $d_{\max} = \max\{d_i \mid b_i \in \mathcal{B}\}$.

For points P and Q we denote by $\overline{PQ}$ the line segment between P and Q of length $|PQ|$. $\overrightarrow{PQ}$ denotes the vector from P to Q. When we write "axis-parallel container" we mean "axis-parallel rectangular cuboid as a container". We use the term box as a synonym for rectangular cuboid.

Packing under translation means that a separate translation is applied to each object moving it inside the container. The translated objects are not allowed to overlap. Packing under rigid motion means that a (separate) rotation may be applied to each object before it is translated into the container.

▶ **Definition 1** (strip packing)**.** An instance for the *strip packing problem* consists of an axis parallel strip and a set of axis parallel boxes, i.e. in two dimensions the width and in three dimensions the width and the depth are fixed and the objective is to pack the boxes under translation such that the height is minimized.

▶ **Definition 2** (orthogonal minimal container packing – OMCOP)**.** An instance of this problem is a set of convex polyhedra. The aim is to pack these polyhedra non-overlapping such that the minimal axis-parallel container has minimal volume. Variants include the kind of motions allowed or that more specialized objects are to be packed.

This work only considers algorithms in two or three dimensions. For ease of notation we always assume the lower left (front) corner of the container to lie in the origin. V_{opt} denotes the minimal possible volume for a container.

The following algorithm was given by von Niederhäusern [11]. It will be used later as a subroutine. For an example see Figure 1.

Algorithm 1:

```
Input: A list $\mathcal{S}$ of rectangles $r_i$, denoted by their width $w_i$ and height $h_i$, a width for
       the strip $w$
1. Order the rectangles in $\mathcal{S}$ by decreasing width, such that if $i < j$ then $w_i \geq w_j$.
2. Split $\mathcal{S}$ in sublists $\mathcal{S}_j = \left\{ r_i \in \mathcal{S} \mid \frac{w}{2^{j-1}} \geq w_i > \frac{w}{2^j} \right\}$ for $j \geq 1$.
3. Start with packing the rectangles in $\mathcal{S}_1$ on top of each other in the strip $[0, w] \times [0, \infty)$.
4. Split the remaining strip in two substrips with width $\frac{w}{2}$ and pack the rectangles in $\mathcal{S}_2$
   one after another into these substrips. Each rectangle $r_i$ is packed in the substrip with
   current minimal height.
5. Again split the substrips into two and pack $S_3$. Iterate that process until everything is
   packed.
```

▶ Remark. Note that the strip is half filled with rectangles up to the lower boundary of the highest rectangle that touches the upper end of the packing. Otherwise,this rectangle could have been placed lower. That means that the strip is half filled with rectangles except for a part with area at most $w \cdot h_{\max}$.

▶ Remark. Steps 1 and 2 can be done in $\mathcal{O}(n \log n)$ time where n is the size of $\mathcal{S}$. Steps 4 and 5 are presented in a simplified way in order to convey the idea of the algorithm in a more understandable manner. In reality it may happen that sublists $\mathcal{S}_j$ are empty and therefore splitting all substrips until they have the suitable width takes too much time. Hence, we split off a new substrip of suitable width from an existing one only when needed. To maintain all substrips with their currently occupied height, a heap-like data structure is used. Then, we can perform steps 3 to 5 in $\mathcal{O}(n \log n)$ time.

In this section we consider the version of OMCOP where the given objects are axis-parallel boxes that are to be packed under translation. The idea behind the reduction of OMCOP to strip packing is to test different base areas for the strip and to return the result with minimal volume. Assuming that the lower left corner of the base area is located at the origin, we test each point in a set $\mathcal{S}$ as a possible upper right corner for the base area. Testing means that we call a strip packing algorithm with the given boxes and the base area

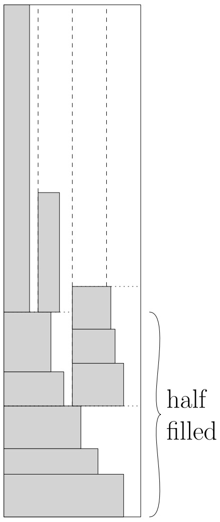


Figure 1 Result of Algorithm 1.

implied by the point of $\mathcal{S}$. $\mathcal{S}$ will be determined by a parameter ε: the smaller ε, the more elements $\mathcal{S}$ contains, the better the approximation ratio gets.

Note that for the width W_{opt} of an optimal container, the following inequalities hold:

1. $W_{\text{opt}} \leq W_\Sigma$, where W_Σ denotes the sum of all widths of the boxes to be packed. It is an upper bound because the width of an optimal container has to be the sum of the widths of some of the objects. Otherwise they can be pushed together reducing the width of the container and thereby its volume.
2. $W_{\text{opt}} \geq w_{\max}$, where $w_{\max}$ denotes the width of the widest box. Since this box needs to be packed, this is a lower bound for the width of the container.

The analogous bounds for the depth of an optimal container hold for the same reasons. In the following H_{opt}, W_{opt}, D_{opt}, and V_{opt} denote the height, width, depth, and volume of the same optimal container. Let $\varepsilon' = \frac{\varepsilon}{2(\varepsilon+\alpha)}$ for a constant α defined later.

The set $\mathcal{S}$ is obtained by dividing the intervals of possible width and depth logarithmically:

$$\begin{aligned}\mathcal{S} = &\{W_\Sigma \left(1-\varepsilon'\right)^i \mid i \in \mathbb{N}, W_\Sigma \left(1-\varepsilon'\right)^i > w_{\max}\} \cup \{w_{\max}\} \times \\ &\{D_\Sigma \left(1-\varepsilon'\right)^j \mid j \in \mathbb{N}, D_\Sigma \left(1-\varepsilon'\right)^j > d_{\max}\} \cup \{d_{\max}\}.\end{aligned}$$

For an example for $\mathcal{S}$ see Figure 2.

▶ **Theorem 3.** *If we use an α-approximation algorithm of runtime $T(n)$ to pack n boxes under translation into the strips and the set $\mathcal{S}$ defined above, we obtain an $(\alpha+\varepsilon)$-approximation algorithm for the* OMCOP *variant where n axis aligned boxes are to be packed under translation. Its runtime is $\mathcal{O}\left(T(n)\frac{\log^2 n}{\varepsilon^2}\right)$.*

Proof. There exist $a, b \in \mathbb{N}$ with $W_\Sigma \left(1-\varepsilon'\right)^{a+1} < W_{\text{opt}} \leq W_\Sigma \left(1-\varepsilon'\right)^a$ and $D_\Sigma \left(1-\varepsilon'\right)^{b+1} < D_{\text{opt}} \leq D_\Sigma \left(1-\varepsilon'\right)^b$. Eventually the boxes will be packed in a strip with base area $W \times D$ with $W = W_\Sigma \left(1-\varepsilon'\right)^a$ and $D = W_\Sigma \left(1-\varepsilon'\right)^b$. Since $W \geq W_{\text{opt}}$ and $D \geq D_{\text{opt}}$, the minimal height for a strip packing with base area $W \times D$ is at most H_{opt}. Therefore, we obtain a packing with height $H \leq \alpha H_{\text{opt}}$. The associated container has volume

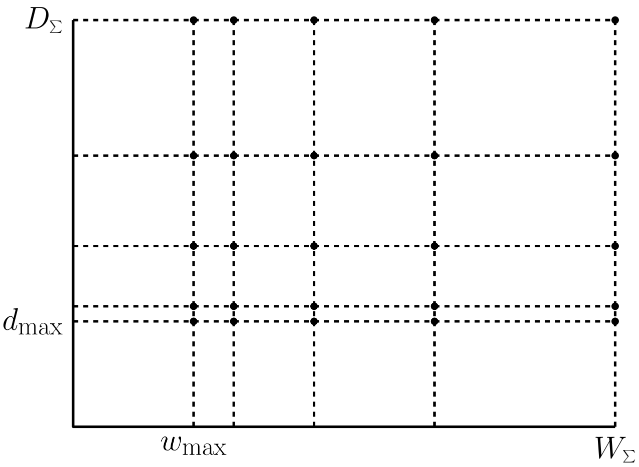


Figure 2 Example for Set $\mathcal{S}$ with $\varepsilon = \frac{3}{4}$ and $\alpha = 1.5$.

V with

$$\begin{aligned} V &= HWD \\ &\le (\alpha H_{\text{opt}}) \left(W_\Sigma (1-\varepsilon')^a\right) \left(D_\Sigma (1-\varepsilon')^b\right) \\ &\le (\alpha H_{\text{opt}}) \left(\frac{W_{\text{opt}}}{1-\varepsilon'}\right) \left(\frac{D_{\text{opt}}}{1-\varepsilon'}\right) \\ &\le \frac{\alpha}{(1-\varepsilon')^2} V_{\text{opt}} \\ &\le \frac{\alpha}{1-2\varepsilon'} V_{\text{opt}} = (\alpha+\varepsilon) V_{\text{opt}} \qquad \text{, since } \varepsilon' = \frac{\varepsilon}{2(\varepsilon+\alpha)}. \end{aligned}$$

The size of $\mathcal{S}$ is

$$\begin{aligned} |\mathcal{S}| &= \left(\left\lceil \log_{\frac{1}{1-\varepsilon'}} W_\Sigma \right\rceil - \left\lfloor \log_{\frac{1}{1-\varepsilon'}} w_{\max} \right\rfloor + 1\right) \left(\left\lceil \log_{\frac{1}{1-\varepsilon'}} D_\Sigma \right\rceil - \left\lfloor \log_{\frac{1}{1-\varepsilon'}} d_{\max} \right\rfloor + 1\right) \\ &= \mathcal{O}\left(\frac{\log^2 n}{(-\log(1-\varepsilon'))^2}\right), \text{ since } \frac{W_\Sigma}{w_{\max}} \le n, \text{ where } n \text{ is the number of boxes} \\ &= \mathcal{O}\left(\frac{\log^2 n}{\varepsilon^2}\right), \end{aligned}$$

since $-\log(1-x) \ge x$ for $x \in [0,1]$ and $\varepsilon' \ge c\varepsilon$ for some constant $c > 0$.

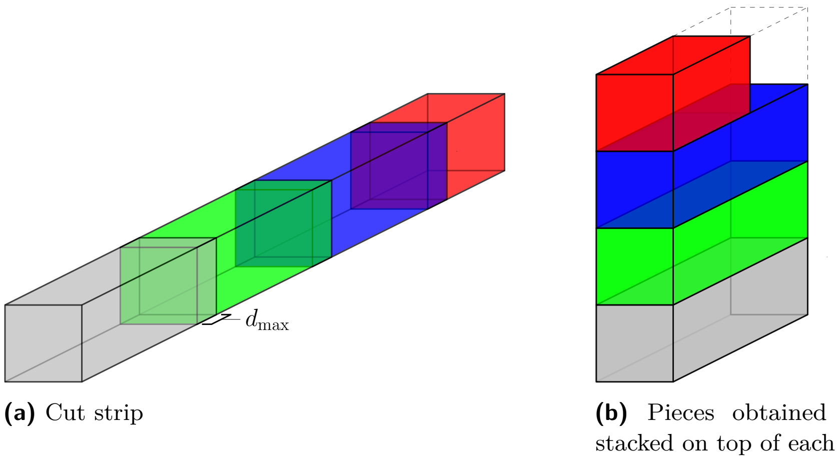
Therefore we get the desired running time. ◀

If we use the algorithm given by Diedrich et al. [5] to pack the boxes into the strips, we obtain the following corollary.

▶ **Corollary 4.** *There exists a $(7.25+\varepsilon)$-approximation algorithm for packing axis-parallel boxes under translation into a minimum volume axis-parallel box with running time polynomial in both the input size and $\frac{1}{\varepsilon}$.*

3 Algorithms for Variants of OMCOP

In this section, we will give algorithms for variants of OMCOP. The basic idea is to get rid of the third dimension by dividing the set of objects into sets of objects with similar height



(a) Cut strip

(b) Pieces obtained from one strip stacked on top of each other

Figure 3

and then packing those using an algorithm for two-dimensional boxes. These containers then get cut into pieces with equal base area and the pieces will be stacked on top of each other.

3.1 Packing Cuboids under Translation

Even though this algorithm gets outperformed by the construction in the previous section, we state it here as base for the algorithms for the other variants. Let $\alpha \in (0,1)$ and $c > 1$ be two parameters that we will choose later.

```
Algorithm 2:
  Input: A set of axis parallel boxes $\mathcal{B} = \{b_1, \dots, b_n\}$
1. Partition $\mathcal{B}$ into subsets of boxes that have almost the same height:
   $\mathcal{B}_j = \left\{b_i \in \mathcal{B} \mid h_{\max} \cdot \alpha^j < h_i \leq h_{\max} \cdot \alpha^{j-1}\right\}$.
2. Pack the boxes of every $\mathcal{B}_j$ into a strip with width $w_{\max}$ and height $h_{\max} \cdot \alpha^{j-1}$
   considering the depth of the boxes instead of the height, i.e., the strip grows into the
   depth. This is done by applying Algorithm 1 to pack the lower facets of the boxes
   (rectangles) into the lower facet of the strip (2d-strip).
3. Divide the strips into pieces with depth $(c-1) \cdot d_{\max}$, ignoring the last part of the
   strip of depth $d_{\max}$. (Parts of boxes contained in this part of the strip will be covered
   in step 5 anyway.)
4. Assign each box to the piece its front lies in.
5. Extend each piece to depth $c \cdot d_{\max}$ such that every assigned box lies entirely in the
   piece.
6. Stack the pieces on top of each other.
```

For an illustration of steps 3 to 6 see Figure 3. The first step can be done in $\mathcal{O}(n)$ time. The second step needs time $\mathcal{O}(n \log n)$ (see Remarks on Algorithm 1). The rest can be done in linear time. Therefore, Algorithm 2 runs in $\mathcal{O}(n \log n)$ time. We obtain

▶ **Theorem 5.** *For suitable values of c and α Algorithm 2 computes a $\left(\frac{3}{\sqrt[3]{2}-1} \approx 11.542\right)$-approximation for the variant of three-dimensional* OMCOP *where n axis parallel cuboids are packed under translation in $\mathcal{O}(n \log n)$ time.*

Proof. Let D_j denote the depth of the strip obtained in step 2 for the boxes in $\mathcal{B}_j$. Then we get by step 3 $\left\lceil \frac{D_j - d_{\max}}{(c-1)d_{\max}} \right\rceil$ pieces. After step 5 each piece has volume $c \cdot d_{\max} w_{\max} h_{\max} \alpha^{j-1}$.

Consider the total volume V_j of the pieces obtained for the subset $\mathcal{B}_j$:

$$\begin{aligned} V_j &= c \cdot d_{\max} \left\lceil \frac{D_j - d_{\max}}{(c-1)\, d_{\max}} \right\rceil w_{\max} h_{\max} \alpha^{j-1} \\ &< \frac{c}{c-1} \left(D_j - d_{\max}\right) w_{\max} h_{\max} \alpha^{j-1} + c \cdot d_{\max} w_{\max} h_{\max} \alpha^{j-1}. \end{aligned}$$

We know from the two-dimensional packing algorithm that the base area of the strip is half filled with boxes except for the last part of depth at most $d_{\max}$ (see Remarks on Algorithm 1), so $\left(D_j - d_{\max}\right) w_{\max} \leq 2 \sum_{b \in \mathcal{B}_j} A_B\left(b\right)$ where $A_B\left(b\right)$ denotes the base area of box b. We also know that for every $b_i \in \mathcal{B}_j$ the inequality $h_{\max} \alpha^{j-1} < \frac{h_i}{\alpha}$ holds. Therefore, we get for the total volume of the packing V that

$$\begin{aligned} V &\leq \sum_j \left(\frac{c}{c-1} \left(D_j - d_{\max}\right) w_{\max} h_{\max} \alpha^{j-1} + c \cdot d_{\max} w_{\max} h_{\max} \alpha^{j-1} \right) \\ &\leq \sum_j \left(\frac{2c}{\alpha\left(c-1\right)} \sum_{b \in \mathcal{B}_j} V\left(b\right) + c \cdot w_{\max} \cdot d_{\max} \cdot h_{\max} \alpha^{j-1} \right) \\ &\leq \frac{2c}{\alpha\left(c-1\right)} \underbrace{\sum_{b \in \mathcal{B}} V\left(b\right)}_{\leq V_{\text{opt}}} + c \cdot \underbrace{w_{\max} \cdot d_{\max} \cdot h_{\max}}_{\leq V_{\text{opt}}} \cdot \sum_{l=0}^{\infty} \alpha^l \qquad (1) \\ &\leq \left(\frac{2c}{\alpha\left(c-1\right)} + \frac{c}{1-\alpha} \right) V_{\text{opt}}. \qquad (2) \end{aligned}$$

The factor before V_{opt} in term (2) is minimized if the partial derivatives with respect to c and α are 0. Solving the resulting system of equations we get $c = \sqrt[3]{2} + 1 \approx 2.2599$ and $\alpha = \frac{1}{3}\left(2 - \sqrt[3]{4} + \sqrt[3]{2}\right) \approx 0.5575$. This gives an approximation ratio of $\frac{3}{\sqrt[3]{2}-1} \approx 11.542$. ◀

3.2 Packing Cuboids under Rigid Motions

3.2.1 Cuboid as Container

Now we consider the variant of OMCOP where the objects to be packed are boxes and rigid motions are allowed. Let V_{opt} denote the volume of an optimal container for the given setting. We basically use the algorithm stated above but with an extra preprocessing step, namely rotating every box $b_i \in \mathcal{B}$ such that it becomes axis parallel and $h_i \geq w_i \geq d_i$. This can be done in $\mathcal{O}(n)$ time. To prove the performance bound of this algorithm we need the following lemma.

▶ **Lemma 6.** *If every $b_i = (h_i, w_i, d_i) \in \mathcal{B}$ is oriented such that $h_i \geq w_i \geq d_i$, then $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq \sqrt{6} \cdot V_{opt}$.*

Proof. Since an optimal container has to contain the box determining $h_{\max}$, it contains a line segment of length $h_{\max}$. The projection of that line segment on at least one of the axes has to have length at least $\frac{1}{\sqrt{3}} h_{\max}$. W.l.o.g. let this axis be the x-axis. Therefore, the optimal container has an extent of at least $\frac{1}{\sqrt{3}} h_{\max}$ in x-direction.

Since every box is at least as high as wide, a box with width $w_{\max}$ contains a disk D with diameter $w_{\max}$ and so the optimal container does. Observe that D contains a diametric line segment l which is parallel to the y-z-plane. Consequently, the projection of l and therefore the one of the whole box on the y-axis or on the z-axis has a length of at least $\frac{1}{\sqrt{2}} w_{\max}$. W.l.o.g. let this be the y-axis.

A box with depth $d_{\max}$ contains a sphere with diameter $d_{\max}$. The projection of this sphere on any axis has length at least $d_{\max}$.

Summarizing, each optimal box has volume at least $\frac{1}{\sqrt{6}} h_{\max} \cdot w_{\max} \cdot d_{\max}$ ◀

Observe that every argument leading to inequality (1) still holds for this variant of the algorithm. Using Lemma 6 to estimate $h_{\max} \cdot w_{\max} \cdot d_{\max}$ we get an approximation factor of $\frac{2c}{\alpha(c-1)} + \frac{c \cdot \sqrt{6}}{1-\alpha}$. Minimizing this expression as before yields the following theorem.

▶ **Theorem 7.** *The given algorithm computes a* 17.738*-approximation for the variant of three-dimensional* OMCOP *where n axis parallel cuboids are packed under rigid motions in* $\mathcal{O}(n \log n)$ *time.*

3.2.2 Convex Container

If we allow a convex container instead of an orthogonal container, we can use the same algorithm but adapt the analysis. The arguments leading to inequality (1) still hold since they only use the total volume of the boxes as estimate for the volume of an optimal container. To estimate $h_{\max} \cdot w_{\max} \cdot d_{\max}$, we use the following lemma. Note that V_{opt} here denotes the volume of a minimal convex container instead of an axis parallel container.

▶ **Lemma 8.** *If every $b_i = (h_i, w_i, d_i) \in \mathcal{B}$ is oriented such that $h_i \geq w_i \geq d_i$, then $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq 6 \cdot V_{opt}$.*

Proof. Consider the line segment, disk and sphere from the proof of Lemma 6. The line segment has length $h_{\max}$. The disk with diameter $w_{\max}$ contains a line segment of length $w_{\max}$ that is perpendicular to the first line segment. The sphere with diameter $d_{\max}$ contains a line segment of length $d_{\max}$ that is perpendicular to the first two line segments. It is well known (see, e.g., Lemma 6 from [8]) that the convex hull of these three line segments has a volume of at least $\frac{1}{6} h_{\max} w_{\max} d_{\max}$. ◀

This leads with inequality (1) to the approximation ratio $\frac{2c}{\alpha(c-1)} + \frac{c \cdot 6}{1-\alpha}$. Minimizing this term as before yields the following theorem.

▶ **Theorem 9.** *Using the algorithm described in section 3.2 we get a* 29.135*-approximation for packing n axis parallel boxes under rigid motions into a smallest-volume convex container in time $\mathcal{O}(n \log n)$.*

3.3 Packing Convex Polyhedra under Rigid Motions

3.3.1 Cuboid as Container

We use the algorithm from the previous sections to pack convex polyhedra under rigid motions into an axis-parallel box of minimal volume. To do so, we add another preprocessing step where we compute a bounding box for every polyhedron according to the following lemma. We then pack these boxes with the algorithm discussed in the previous section.

▶ **Lemma 10.** *For every m-vertex convex polyhedron K in $\mathbb{R}^d$, there is a box B that contains K with $V(B) \leq d! V(K)$ that can be computed in $O(d^2 m^2)$ time, or $O(m \log m)$ time if $d = 3$.*

Proof by induction on the dimension d. In one dimension, the Lemma holds obviously.

In higher dimensions d, let P, Q be two points of K with maximum distance and $|PQ| = l$. Let π_P be the hyperplane normal to $\overline{PQ}$ in the point P. Let K' be the orthogonal projection of K onto π_P. By the inductive hypothesis there is a $(d-1)$-dimensional box B' containing K'

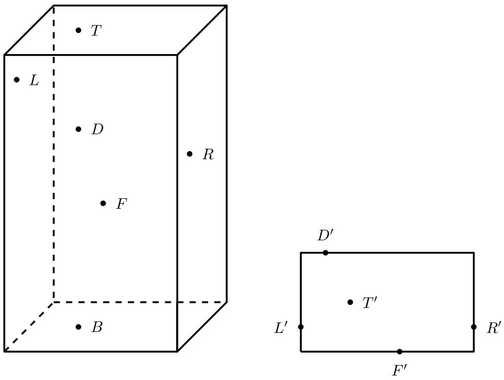


Figure 4 Box with a point of the enclosed polyhedron in every facet and the projection of the box on a plane perpendicular to $\overline{TB}$. By construction, the images of T and B under the projection are the same.

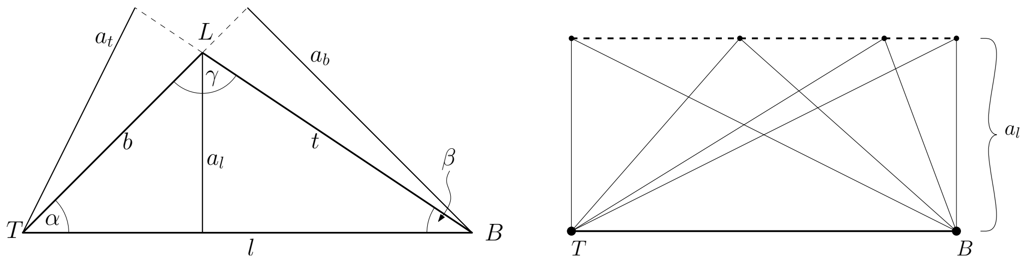
for which $V'(B') \leq (d-1)!V'(K')$ where V' denotes the $(d-1)$-dimensional volume. Then K is contained in the box B with base B' and height l and $V(B) = lV'(B') \leq l(d-1)!V'(K')$. It is well known (see e.g. [8]) that for any convex body K, its projection K' on some hyperplane π_P, and a line segment l perpendicular to π_P, it holds: $V(K) \geq \frac{1}{d} \cdot l \cdot V'(K')$. Hence, we get for the volume of B: $V(B) \leq d!V(K)$. B can be computed by testing every pair of vertices to find P and Q that have maximal distance. This takes $\mathcal{O}(dm^2)$ time. Then K gets projected on a hyperplane perpendicular to $\overline{PQ}$. This is possible in $\mathcal{O}(dm)$ time. Then we proceed recursively with the projection of K. In total we need $\mathcal{O}(d^2m^2)$. The asymptotically fastest algorithm for dimension three however has runtime $\mathcal{O}(m \log m)$, see [10]. ◀

The construction in the proof of Lemma 10 is the same as in Lemma 7 from [8]. We get a total running time of $\mathcal{O}(m \log m)$ for computing the bounding boxes of three-dimensional polyhedra with m vertices in total.

For the analysis of the algorithm presented in this section we need several notations and lemmata that follow. Consider the box $b = (h, w, d)$ obtained from the polyhedron p by Lemma 10 after the algorithm rotated it in axis-parallel position such that $h \geq w \geq d$. Notice that in every facet of b lies at least one point of p. We call the top and bottom one T and B, which are unique by construction. In the left and right facet of b, we choose such a point from each and call them L and R. By construction, the distance from them to the front facet has to be the same. We do the same for the front and rear facet and call them F and D respectively. We know from the construction that $|TB| = h$ and $\overline{TB}$ is parallel to the longest edge of b. If we project the polyhedron onto a plane perpendicular to $\overline{TB}$, we call the images of T, L, R, F and D under the projection T', L', R', F' and D', respectively. See Figure 4 for illustration. Due to the construction of b, $|L'R'| = w$ holds.

▶ **Lemma 11.** *Let $b = (h, w, d)$ with $h \geq w \geq d$ be the enclosing box obtained for polyhedron p by the algorithm from Lemma 10. Then, parallel to any given plane, p contains a line segment of length at least $w \cdot \frac{1}{\sqrt{5}}$.*

Proof. Consider the points T, B, L and R as described above. The distance between line segment $\overline{TB}$ and L or the distance between line segment $\overline{TB}$ and R is at least $\frac{w}{2}$. Let w.l.o.g. L be the point with larger distance to $\overline{TB}$. Consider the triangle $\triangle(T, B, L)$ with edges and angles labeled according to Figure 5a. Notice that $\alpha \leq 90°$ and $\beta \leq 90°$. Let a_t be the height of the triangle on edge t, a_b on edge b, and a_l on edge l.



(a) Labelled triangle $\triangle(T, B, L)$ **(b)** Possible triangles $\triangle(T, B, L)$

Figure 5

Due to the construction of $\triangle(T, B, L)$, we know that $a_l \geq \frac{w}{2}$. We will later show that $a_b \geq \frac{w}{\sqrt{5}}$ and $a_t \geq \frac{w}{\sqrt{5}}$. If we choose a plane parallel to the given one, such that the intersection between the plane and $\triangle(T, B, L)$ contains T, B or L but is not only one point, then we know that the intersection is at least a line segment with length at least $\min(a_t, a_b, a_l) \geq \frac{w}{\sqrt{5}}$ which completes the proof. It remains to show that $a_t, a_b \geq \frac{w}{\sqrt{5}}$.

We only show that $a_b \geq \frac{w}{\sqrt{5}}$ since the proof for a_t is analogous. Figure 5b depicts possible triangles with given distance $|TB|$ and height a_l. a_b is the distance between B and the line defined by T and L. Since $\beta \leq 90°$ this distance is minimal for $\beta = 90°$.

Let A be the area of $\triangle(T, B, L)$ with $\beta = 90°$.

It holds

$$\frac{a_l \cdot |TB|}{2} = A = \frac{a_b \cdot |TL|}{2}.$$

Hence

$$a_l \cdot h = a_b \cdot \sqrt{h^2 + a_l^2},$$

since $|TB| = h$ and using Pythagoras' theorem for replacing $|TL|$. That gives

$$\begin{aligned} a_b &= \frac{a_l \cdot h}{\sqrt{h^2 + a_l^2}} \\ &= \frac{1}{\sqrt{\frac{1}{a_l^2} + \frac{1}{h^2}}} \\ &\geq \frac{1}{\sqrt{\frac{4}{w^2} + \frac{1}{w^2}}} \\ &= \frac{w}{\sqrt{5}} \end{aligned}$$

◀

▶ **Lemma 12.** *Let $b = (h, w, d)$ with $h \geq w \geq d$ be the enclosing box obtained for a convex polyhedron p by the algorithm from Lemma 10. Then the projection of p onto any arbitrary line g has length at least $\frac{1}{8\sqrt{3}}d$.*

This Lemma is shown by an elaborate construction, where we find four line segments inside p such that the projection of at least one of them onto g has length at least $\frac{1}{8\sqrt{3}}d$. See Appendix A for the complete proof.

Summarized, the algorithm for packing convex polyhedra works as follows: First, we compute a bounding box for every polyhedron with the algorithm from Lemma 10, then we rotate each box b_i together with its contained polyhedron p_i in an axis-prallel orientation such that $h_i \geq w_i \geq d_i$. Finally, we run Algorithm 2 with the rotated boxes.

Now consider the polyhedra p_1, p_2, p_3 that determine $h_{\max}$, $w_{\max}$ and $d_{\max}$ in the placement of the enclosing boxes the described algorithm computes. p_1 contains a line segment of length $h_{\max}$ and so its projection to at least one of the axes is at least $\frac{1}{\sqrt{3}} h_{\max}$. W.l.o.g. let this axis be the x-axis. Furthermore, by Lemma 11 the projection of p_2 onto the y-z-plane contains a line of length at least $\frac{1}{\sqrt{5}} w_{\max}$. Therefore, the projection of p_2 onto the y-axis or the one onto the z-axis has length at least $\frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{5}} w_{\max} = \frac{1}{\sqrt{10}} w_{\max}$. The projection of p_3 to the remaining axis has length at least $\frac{1}{8\sqrt{3}} d_{\max}$ by Lemma 12. An axis parallel box with minimal volume containing p_1, p_2, p_3 has at least the described side lengths and so we get the following lemma:

▶ **Lemma 13.** *For packing convex polyhedra under rigid motions into a minimum-volume axis parallel container, the following inequality holds:* $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq 24\sqrt{10} V_{opt}$.

From Lemma 10 we know that the volume of the smallest enclosing box for a polyhedron is at most 6 times the volume of the polyhedron. With the previous lemma and this knowledge we derive the following approximation ratio from inequality (1):

$$\frac{12c}{\alpha\,(c-1)} + \frac{c \cdot 24\sqrt{10}}{1-\alpha}. \tag{3}$$

The running time of this algorithm is determined by the computation of the bounding boxes and the packing of these boxes: $\mathcal{O}\left(m \log m + n \log n\right) = \mathcal{O}\left(m \log m\right)$ where m is the total number of vertices of the polyhedra. Hence, by minimizing term (3) as before we get the following theorem.

▶ **Theorem 14.** *The given algorithm computes an orthogonal container with volume at most* 277.59 *times the volume of an orthogonal minimal container for the variant of three-dimensional* OMCOP *where a set of convex polyhedra having m vertices in total are to be packed under rigid motions. The runtime of the algorithm is* $\mathcal{O}\left(m \log m\right)$.

3.3.2 Convex Container

Next, we show that the algorithm from the previous section is not only a constant factor approximation for the smallest axis parallel cuboid under rigid motions but even for the smallest convex container. Of course, the approximation factor is higher and, first, we get the following lemma instead of Lemma 13:

▶ **Lemma 15.** *For packing convex polyhedra under rigid motions into a minimum-volume convex container, the following inequality holds:* $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq 48\sqrt{15} V_{opt}$.

Proof. As before let p_1, p_2, p_3 be the polytopes that determine $h_{\max}$, $w_{\max}$ and $d_{\max}$. p_1 contains a line segment of length $h_{\max}$. By Lemma 11, p_2 contains a line segment of length $\frac{w_{\max}}{\sqrt{5}}$ that is perpendicular to the first line segment. By Lemma 12, p_3 contains a line segment with length $\frac{d_{\max}}{8\sqrt{3}}$ that is perpendicular to the first two lines. Since any convex body containing three pairwise perpendicular line segments of length a, b, c has volume at least $\frac{1}{6} abc$ (cf. Lemma 6 in [8]), we get a lower bound on the volume of the convex hull which is also a lower bound for the volume of an optimal container. ◀

As before we use Lemma 10 and the previous lemma to estimate inequality (1) and obtain the following approximation ratio: $\frac{12c}{\alpha(c-1)} + \frac{c \cdot 48\sqrt{15}}{1-\alpha}$. Minimizing this term as before yields the following result.

▶ **Theorem 16.** *The algorithm given in Section 3.3 computes a convex container with volume at most* 511.37 *times the volume of a minimal convex container for packing a set of convex polyhedra having* m *vertices in total under rigid motions in time* $\mathcal{O}(m \log m)$*.*

References

1 Helmut Alt, Mark de Berg, and Christian Knauer. Approximating minimum-area rectangular and convex containers for packing convex polygons. In *Proc. 23th Annual European Symp. Algorithms (ESA)*, pages 25–34, 2015.

2 Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Math. Oper. Res.*, 31(1):31–49, 2006.

3 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *Proc. 25th Annual ACM-SIAM Symp. Discrete Algorithms (SODA)*, pages 13–25, 2014.

4 H. S. M. Coxeter. *Introduction to geometry*, page 12. John Wiley & Sons, Inc., New York-London-Sydney, second edition, 1969.

5 Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3D orthogonal knapsack. *J. Comput. Sci. Tech.*, 23(5):749–762, 2008.

6 Klaus Jansen and Lars Prädel. A new asymptotic approximation algorithm for 3-dimensional strip packing. In *Proc. 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 327–338, 2014.

7 Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Math. Oper. Res.*, 25(4):645–656, 2000.

8 A. M. Macbeath. A compactness theorem for affine equivalence-classes of convex regions. *Canadian J. Math.*, 3:54–61, 1951.

9 F. K. Miyazawa and Y. Wakabayashi. Three-dimensional packings with rotations. *Comput. Oper. Res.*, 36(10):2801–2815, 2009.

10 Edgar A. Ramos. An optimal deterministic algorithm for computing the diameter of a three-dimensional point set. *Discrete & Computational Geometry*, 26(2):233–244, 2001.

11 Léonard von Niederhäusern. Packing polygons: Research of approximation algorithms. Master's thesis, Freie Universität Berlin and École Polytechnique Fédérale de Lausanne, 2014.

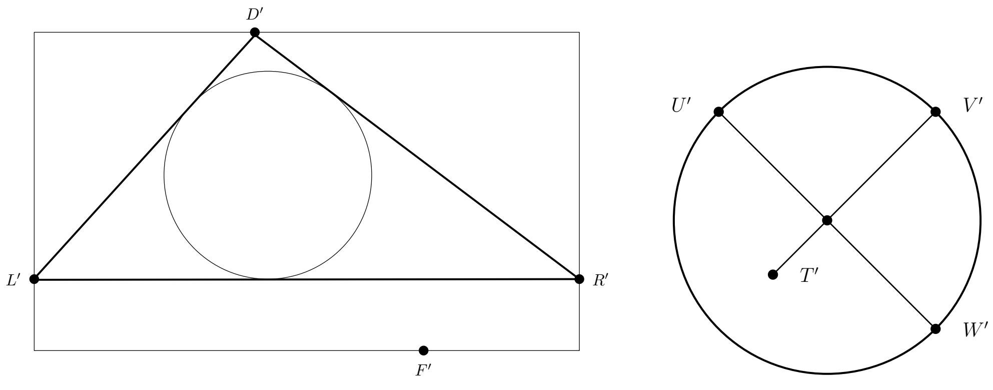
A Proof of Lemma 12

We construct four line segments inside of p such that the projection of at least one of them onto the line has the desired length.

Consider the projection of p onto a plane perpendicular to $\overline{TB}$ as described above (Figure 4). Then $\triangle(L', R', F')$ or $\triangle(L', R', D')$ has an area $A \geq \frac{dw}{4}$. The perimeter of the projection of the box, namely $2(w + d)$, gives an upper bound for the perimeter u of the triangles. It is well known (see, e.g.,[4]) that the radius of the incircle of a triangle with area A and perimeter u is $r = \frac{2A}{u}$. Hence, we know that the projection of p contains a circle with radius r where

$$r = \frac{2A}{u} \geq \frac{dw}{4(d+w)} \geq \frac{1}{8}d \text{ , since } d \leq w. \tag{4}$$

See Figure 6a for an example.



(a) Circle in the projection of p that has radius at least $\frac{1}{8}d$

(b) Construction of U', V' and W'

Figure 6

Now we can find points U', V', W' in the projection, such that U', V', W' lie on the circle with radius r and $|T'V'| = k \geq r$, $|U'W'| = l = 2r$ and $\overline{T'V'} \perp \overline{U'W'}$. To obtain V', we shoot a ray from T' through the center of the circle until we hit the circle and call this point V'. $\overline{U'W'}$ is the diameter of the circle perpendicular to $\overline{T'V'}$. See Figure 6b for an example.

Let U, V, W be preimages of U', V', W' under the projection. Hence, they lie inside p. The line segments whose projections on the given line g we consider are $\overline{BT}$, $\overline{BV}$, $\overline{VT}$ and $\overline{WU}$.

The length of the projection of a line segment onto g is the scalar product of the vector between the endpoints of the line segment and a unit vector with same direction as g. To simplify the computation of the scalar product, we define the coordinate system as follows: B is equal to the origin. T lies on the z-axis. The y-coordinate of V is 0. Then U and W have the same x-coordinate. Now we have

$$\overrightarrow{BT} = \begin{pmatrix} 0 \\ 0 \\ h \end{pmatrix} \quad \overrightarrow{BV} = \begin{pmatrix} k \\ 0 \\ h_V \end{pmatrix} \quad \overrightarrow{VT} = \begin{pmatrix} -k \\ 0 \\ h - h_V \end{pmatrix} \quad \overrightarrow{WU} = \begin{pmatrix} 0 \\ l \\ h_{WU} \end{pmatrix},$$

for values k, l with properties described above, and h_V, h_{WU} where $0 \leq h_V \leq h$ and $|h_{WU}| \leq h$. Let $\overrightarrow{g} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ be the direction of g in the defined coordinate system, with $|\overrightarrow{g}| = 1$. We now look at the lengths of the projections of the line segments onto the given line and distinguish four cases.

Case 1: $|x| \geq \frac{1}{\sqrt{3}}$. Then, using inequality (4), if $\operatorname{sgn}(z) = \operatorname{sgn}(x)$

$$|\overrightarrow{BV} \cdot \overrightarrow{g}| \geq k|x| \geq r|x| \geq \frac{1}{\sqrt{3} \cdot 8} d$$

or, if $\operatorname{sgn}(z) \neq \operatorname{sgn}(x)$

$$|\overrightarrow{VT} \cdot \overrightarrow{g}| \geq k|x| \geq \frac{1}{\sqrt{3} \cdot 8} d.$$

Case 2: $\boldsymbol{|z| \cdot h \geq \frac{1}{\sqrt{3}\cdot 8}d}$. Note, that this inequality is satisfied if $|z| \geq \frac{1}{\sqrt{3}}$. Then

$$|\overrightarrow{BT} \cdot \overrightarrow{g}| = h \cdot |z| \geq \frac{1}{\sqrt{3} \cdot 8}d.$$

Case 3: $\boldsymbol{|y| \geq \frac{1}{\sqrt{3}}}$ and $\boldsymbol{\operatorname{sgn}(y) = \operatorname{sgn}(h_{WU}z)}$. Then

$$|\overrightarrow{WU} \cdot \overrightarrow{g}| \geq l|y| \geq \frac{1}{\sqrt{3} \cdot 8}d.$$

Case 4: $\boldsymbol{|y| \geq \frac{1}{\sqrt{3}}}$ and $\boldsymbol{\operatorname{sgn}(y) \neq \operatorname{sgn}(h_{WU}z)}$ and $\boldsymbol{|z| \cdot h < \frac{1}{\sqrt{3}\cdot 8}d}$. Note: $|h_{WU}z| \leq h|z| < \frac{1}{\sqrt{3}\cdot 8}d$ and $l|y| = 2r|y| \geq \frac{2}{\sqrt{3}\cdot 8}d$, hence

$$|\overrightarrow{WU} \cdot \overrightarrow{g}| = l|y| - |h_{WU}z| \geq \frac{1}{\sqrt{3} \cdot 8}d.$$

Since $|\overrightarrow{g}| = 1$, $|x| \geq \frac{1}{\sqrt{3}}$ or $|y| \geq \frac{1}{\sqrt{3}}$ or $|z| \geq \frac{1}{\sqrt{3}}$ holds. Hence, at least one of the 4 cases occurs because $h \geq d$.

Mind the Gap: Essentially Optimal Algorithms for Online Dictionary Matching with One Gap*

Amihood Amir[1], Tsvi Kopelowitz[2], Avivit Levy[3], Seth Pettie[4], Ely Porat[5], and B. Riva Shalom[6]

1 **Bar-Ilan University, Ramat Gan, Israel; and**
Johns Hopkins University, Baltimore, USA
`amir@cs.biu.ac.il`

2 **University of Michigan, Ann Arbor, USA**
`kopelot@gmail.com`

3 **Shenkar College, Ramat Gan, Israel**
`avivitlevy@shenkar.ac.il`

4 **University of Michigan, Ann Arbor, USA**
`pettie@umich.edu`

5 **Bar-Ilan University, Ramat Gan, Israel**
`porately@cs.biu.ac.il`

6 **Shenkar College, Ramat Gan, Israel**
`rivash@shenkar.ac.il`

Abstract

We examine the complexity of the online Dictionary Matching with One Gap Problem (DMOG) which is the following. Preprocess a dictionary D of d patterns, where each pattern contains a special *gap* symbol that can match any string, so that given a text that arrives online, a character at a time, we can report all of the patterns from D that are suffixes of the text that has arrived so far, before the next character arrives. In more general versions the gap symbols are associated with *bounds* determining the possible lengths of matching strings. Online DMOG captures the difficulty in a bottleneck procedure for cyber-security, as many digital signatures of viruses manifest themselves as patterns with a single gap.

In this paper, we demonstrate that the difficulty in obtaining efficient solutions for the DMOG problem, even in the offline setting, can be traced back to the infamous 3SUM conjecture. We show a conditional lower bound of $\Omega(\delta(G_D)+op)$ time per text character, where G_D is a bipartite graph that captures the structure of D, $\delta(G_D)$ is the *degeneracy* of this graph, and *op* is the output size. Moreover, we show a conditional lower bound in terms of the magnitude of gaps for the bounded case, thereby showing that some known offline upper bounds are essentially optimal.

We also provide matching upper-bounds (up to sub-polynomial factors), in terms of the degeneracy, for the online DMOG problem. In particular, we introduce algorithms whose time cost depends linearly on $\delta(G_D)$. Our algorithms make use of graph orientations, together with some additional techniques. These algorithms are of practical interest since although $\delta(G_D)$ can be as large as $\sqrt{d}$, and even larger if G_D is a multi-graph, it is typically a very small constant in practice. Finally, when $\delta(G_D)$ is large we are able to obtain even more efficient solutions.

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

Keywords and phrases Pattern matching, Dictionary matching, 3SUM, Triangle reporting

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.12

* Partially supported by ISF grant 571/14, BSF grant 2014028, and NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 12; pp. 12:1–12:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Understanding the computational limitations of algorithmic problems often leads to algorithms that are efficient for inputs that are seen in practice. This paper, which stemmed from an industrial-acdemic connection [31], is a prime example of such a case. We focus on an aspect of Cyber-security which is a critical modern challenge. Network intrusion detection systems (NIDS) perform protocol analysis, content searching and content matching, in order to detect harmful software. Such malware may appear non-contiguously, scattered across several packets, which necessitates matching *gapped* patterns.

A *gapped pattern* P is one of the form $P_1\ \{\alpha, \beta\}\ P_2$, where each subpattern P_1, P_2 is a string over alphabet Σ, and $\{\alpha, \beta\}$ matches any substring of length at least α and at most β, which are called the *gap bounds*. Gapped patterns may contain more that one gap, however, those considered in NIDS systems typically have at most *one* gap, and are a serious bottleneck in such applications [31]. Analyzing the set of gapped patterns considered by the SNORT software rules shows that 77% of the patterns have at most one gap, and more than 44% of the patterns containing gaps have only one gap. Therefore, an efficient solution for this case is of special interest. Though the gapped pattern matching problem arose over 20 years ago in computational biology applications [28, 19] and has been revisited many times in the intervening years (e.g. [27, 10, 25, 9, 16, 29, 32]), in this paper we study what is apparently a mild generalization of the problem that has nonetheless resisted many researcher's attempts at finding a definitive efficient solution.

The set of d patterns to be detected, called a *dictionary*, could be quite large. While dictionary matching is well studied (see, e.g. [2, 4, 12, 5, 15]), NIDS applications motivate the *dictionary matching with one gap* problem, defined formally as follows.

▶ **Definition 1.** The *Dictionary Matching with One Gap Problem (DMOG)*, is:

Input: A text T of length $|T|$ over alphabet Σ, and a dictionary D of d gapped patterns $P_1, \ldots, P_d$ over alphabet Σ where each pattern has at most one gap.

Output: All locations in T where a pattern $P_i \in D$, $1 \leq i \leq d$, ends.

In the offline DMOG problem T and D are presented all at once. We study the more practical *online* DMOG problem. The dictionary D can be preprocessed in advance, resulting in a data structure. Given this data structure the text T is presented one character at a time, and when a character arrives the subset of patterns with a match ending at this character should be reported before the next character arrives. Three cost measures are of interest: a preprocessing time, a time per character, and a time per match reported. Online DMOG is a serious bottleneck for NIDS, though it has received much attention from both the industry and the academic community.

1.1 Previous Work

Finding efficient solutions for DMOG has proven to be a difficult algorithmic challenge as, unfortunately, little progress has been obtained on this problem even though many researchers in the pattern matching community and the industry have tackled it. Table 1 describes a summary and comparison of previous work. It illustrates that previous formalizations of the problem, either do not enable detection of all intrusions or are incapable of detecting them in an online setting, and therefore, are inadequate for NIDS applications. Table 1 also demonstrates that our upper bounds are essentially optimal (assuming some popular conjectures, as described in Section 2).

Table 1 Comparison of previous work and some new results. The parameters: lsc is the longest suffix chain of subpatterns in D, $socc$ is the number of subpatterns occurrences in T, op is the number of pattern occurrences in T, α^* and β^* are the minimum left and maximum right gap borders in the non-uniformly bounded case, $\delta(G_D)$ is the degeneracy of the graph G_D representing dictionary D.

	Preprocessing Time	Total Query Time	Algorithm Type	Remark
[24]	none	$\tilde{O}(\|T\| + \|D\|)$	online	reports only first occurrence
[32]	$O(\|D\|)$	$\tilde{O}(\|T\| + d)$	online	reports only first occurrence
[18]	$O(\|D\|)$	$O(\|T\| \cdot lsc + socc)$	online	reports one occurrence per pattern and location
[7]	$\tilde{O}(\|D\|)$	$\tilde{O}(\|T\|(\beta - \alpha) + op)$	offline	DMOG
[20]	$O(\|D\|)$	$\tilde{O}(\|T\|(\beta^* - \alpha^*) + op)$	offline	DMOG
This paper	$O(\|D\|)$	$\tilde{O}(\|T\| \cdot \delta(G_D) \cdot lsc + op)$	online	DMOG
This paper	$O(\|D\|)$ $O(\|D\|)$	$\Omega(\|T\| \cdot \delta(G_D)^{1-o(1)} + op)$ $\Omega(\|T\| \cdot (\beta - \alpha)^{1-o(1)} + op)$	online or offline	DMOG

1.2 New Results

The DMOG problem has several natural parameters, e.g., $|D|, d$, and the magnitude of the gap. We establish almost sharp upper and lower bounds for the cases of unbounded gaps ($\alpha = 0, \beta = \infty$), uniformly bounded gaps where all patterns have the same bounds, α and β, on their gap, and the most general non-uniform gaps version, where each pattern $P_i \in D$ has its own gap bounds, α_i and β_i. We show that the complexity of DMOG actually depends on a "hidden" parameter that is a function of the *structure* of the gapped patterns. The dictionary D can be represented as a graph G_D, which is a multi-graph in the non-uniformly bounded gaps case, where vertices correspond to first or second subpatterns and edges correspond to patterns[1]. We use the notion of graph *degeneracy* $\delta(G_D)$ which is defined as follow. The degeneracy of an undirected graph $G = (V, E)$ is $\delta(G) = \max_{U \subseteq V} \min_{u \in U} d_{G_U}(u)$, where d_{G_U} is the degree of u in the subgraph of G induced by U. In words, the degeneracy of G is the largest minimum degree of any subgraph of G. A non-multi graph G with m edges has $\delta(G) = O(\sqrt{m})$, and a clique has $\delta(G) = \Theta(\sqrt{m})$. The degeneracy of a multi-graph can be much higher.

Vertex-triangle queries. A key component in understanding both the upper and lower bounds for DMOG is the *vertex-triangles* problem, where the goal is to preprocess a graph so that given a query vertex u we may list all triangles that contain u. The vertex-triangles problem, besides being a natural graph problem, is of particular interest here since, as will be demonstrated in Section 2, it is reducible to DMOG. Our reduction demonstrates that the complexity of the DMOG problem already emerges when all patterns are of the form of two characters separated by an unbounded gap. This simplified online DMOG problem is

[1] While it may be more natural to consider a directed or bipartite graph, the notion of degeneracy ignores directions, and so let G_D here be an undirected graph for sake of explaining the notion.

equivalent to the following *Induced Subgraph* (ISG) problem. Preprocess a directed graph $G = (V, E)$ such that given a query sequence of vertices online (these vertices need not be all of V), after vertex v_i arrives, all edges $(v_j, v_i) \in E$ with $j < i$ are reported. Notice that answering consecutive queries is done independently. Thus, characters and gapped patterns in DMOG correspond to vertices and edges in ISG, respectively. We show that vertex-triangles queries are reducible to ISG.

This reduction serves two purposes. First, in Section 2 we prove a *conditional lower bound* (CLB) for DMOG based on the 3SUM conjecture by combining a reduction from triangle enumeration to the vertex-triangles problem with our new reduction from the vertex-triangles problem to DMOG. Our lower bound states that any online DMOG algorithm with low preprocessing and reporting costs must spend $\Omega(\delta(G_D)^{1-o(1)})$ per character, assuming the 3SUM conjecture. Interestingly, the path for proving this CLB deviates from the common conceptual paradigms for proving lower bounds conditioned on the 3SUM conjecture, and is of independent interest. In particular, the common paradigm considers set-disjointness or set-intersection type problems, which correspond to edge triangle queries, while here we consider vertex-triangle queries. Moreover, our CLB holds for the *offline* case as well, and can be rephrased in terms of other parameters. For example, in the DMOG problem with uniform gaps $\{\alpha, \beta\}$, we prove that the per character cost of scanning T must be $\Omega((\beta - \alpha)^{1-o(1)})$. This gives some indication that some recent algorithms for the offline version of DMOG problem are essentially optimal ([7, 20]).

Second, in Section 3 we provide optimal solutions (under the 3SUM conjecture), up to subpolynomial factors, for ISG and, therefore, also for vertex-triangles queries, with $O(|E|)$ preprocessing time and $O(\delta(G) + op)$ time per each vertex, where *op* is the size of the output due to the vertex arrival. The connection between ISG and DMOG led us to extend the techniques used to solve ISG, combine them with additional ideas and techniques, thereby introduce several new online DMOG algorithms whose dependence on $\delta(G)$ is linear. Thus, graph degeneracy seems to capture the intrinsic complexity of the problem. On the other hand, the statement of our general algorithmic results is actually a bit more complicated as it depends on other parameters of the input, namely *lsc*, the length of the *longest suffix chain* in the dictionary, i.e., the longest sequence of dictionary subpatterns such that each is a proper suffix of the next. While the parameter *lsc* could theoretically be as large as d, in practice it is very small [31]. Nevertheless, we also present algorithms that in the most dense cases reduce the dependence on *lsc*.

Lower bounds leading to practical upper bounds. After trying to tackle the DMOG problem from the upper bound perspective, we suspected that a lower bound could be proven, and indeed were successful in showing a connection to the 3SUM conjecture. The CLB proof provides insight for the inherent difficulty in solving DMOG, but is also unfortunate news for those attempting to find efficient upper bounds. Fortunately, a careful examination of the reduction from 3SUM to DMOG reveals that the CLB from the 3SUM conjecture can be phrased in terms of $\delta(G_D)$, which turns out to be a small constant in the input instances considered by NIDS (according to an analysis of the graph created using SNORT rules) [31]. This lead to designing algorithms whose runtime can be expressed in terms of $\delta(G_D)$, and can therefore be helpful in practical settings. The following table summarizes our upper-bounds for DMOG.

The design of our algorithms stem from a solution for ISG using $O(m)$ preprocessing time and $O(\delta(G) + op)$ query time. This solution for ISG is extended to solutions for the various DMOG versions. However, since subpatterns can be suffixes of each other, up to *lsc* vertices

Table 2 A summary of upper bounds for DMOG described in this paper. Unbounded, uniform and non-uniform refer to the type of gap bounds under consideration. M is the maximal length of a subpattern in the dictionary D.

Gaps Type	Preprocessing Time	Query Time per Text Character	Space
unbounded	$O(\|D\|)$	$O(\delta(G_D) \cdot lsc + op)$	$O(\|D\|)$
uniform	$O(\|D\|)$	$O(\delta(G_D) \cdot lsc + op)$	$O(\|D\| + lsc(\beta - \alpha + M) + \alpha)$
non-uniform	$O(\|D\|)$	$\tilde{O}(\delta(G_D) \cdot lsc + op)$	$\tilde{O}(\|D\| + lsc(\beta^* - \alpha^* + M) + \alpha^*)$
uniform	$O(\|D\|)$	$O(lsc + \sqrt{lsc \cdot d} + op)$	$O(\|D\| + lsc(\beta - \alpha + M) + \alpha)$
non-uniform	$O(\|D\|$ $+d(\beta^* - \alpha^*))$	$\tilde{O}(\sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + op)$	$\tilde{O}(\|D\| + d(\beta^* - \alpha^*)$ $+\sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + \alpha^*)$

can arrive simultaneously in G_D, the time of our algorithms have a multiplicative factor of lsc. We emphasize that we are not the first to introduce the lsc factor even in solutions for simplified relaxations of the DMOG problem [18]. Also, since subpatterns may be long, we must accommodate a delay in the time a vertex corresponding to a second subpattern is treated as if it has arrived, thus inducing a minor additive space usage. Finally, in [6] we obtain more efficient bounds that depend linearly on $\sqrt{lsc \cdot d}$ when $\delta(D_G) \geq \sqrt{\frac{d}{lsc}}$, by first considering special types of graph orientations, called *threshold* orientations, and then carefully applying data-structure techniques. Notice that while in the uniformly bounded case we have $\delta(G_D) = O(\sqrt{d})$, in the non-uniform case $\delta(G_D)$ could be much higher and so these new algorithms become a vast improvement.

Paper Contributions. The main contributions of this paper are:

- Obtaining algorithms for DMOG that are asymptotically fast for practical inputs.
- Proving matching conditional lower bounds (up to sub-polynomial factors) from the 3SUM conjecture, which in particular deviate from the common paradigm of such proofs.
- Formalizing the ISG problem. This problem serves in this paper for supplying a deeper understanding of the DMOG problem, but is also of independent interest.

Paper Organization. Section 2 describes our conditional lower bounds. In Section 3 we introduce a solution for ISG, which is then extended to simplified versions of the uniformly and non-uniformly bounded DMOG problems in Sections 3.1 and 3.2. In Section 4.1, the ISG algorithms are extended to solutions for the various DMOG versions. More details and results appear in [6].

2 3SUM: Conditional Lower Bounds

In this section we prove that conditioned on the 3SUM conjecture we can prove lower bounds for the vertex-triangles problem, the ISG problem, and the (offline) unbounded DMOG problem. Since the other two versions of DMOG (uniformly and non-uniformly bounded) can solve the unbounded DMOG version, the lower bounds hold for these problems as well.

Background. Polynomial (unconditional) lower bounds for data structure problems are considered beyond the reach of current techniques. Thus, it has recently become popular to prove CLBs based on the *conjectured* hardness of some problem. One of the most popular conjectures for CLBs is that the 3SUM problem (given n integers determine if any three sum to zero) cannot be solved in truly subquadratic time, where truly subquadratic time is $O(n^{2-\Omega(1)})$ time. This conjecture holds even if the algorithm is allowed to use randomization (see e.g. [30, 1, 23, 17]). In this section we show that the infamous 3SUM problem can be reduced to DMOG, which sheds some light on the difficulty of the DMOG problem. Interestingly, our reduction does not follow the common paradigm for proving CLBs based on the 3SUM conjecture, providing a new approach for reductions from 3SUM. This approach is of independent interest, and is described next.

Triangles. Pătraşcu [30] showed that 3SUM can be reduced to enumerating triangles in a tripartite graph. Kopelowitz, Pettie, and Porat [23] provided more efficient reductions, thereby showing that many known triangle enumeration algorithms ([21, 13, 11, 22]) are essentially and conditionally optimal, up to subpolynomial factors. Hence, the offline version of triangle enumeration is well understood. The following two indexing versions of the triangle enumeration problem are a natural extension of the offline problem. In the *edge-triangles* problem the goal is to preprocess a graph so that given a query edge e all triangles containing e are listed. The vertex-triangles problem is defined above. Clearly, both these versions solve the triangle enumeration problem, which immediately gives lower bounds conditioned on the 3SUM conjecture.

The edge-triangles problem on a tripartite graph corresponds to preprocessing a family F of sets over a universe U in order to support set intersection queries in which given two sets $S, S' \in F$ the goal is to enumerate the elements in $S \cap S'$ (see [23]). Indeed, the task of preprocessing F to support set-intersection enumeration queries, and hence edge-triangles, is well studied [14, 22]. Furthermore, the set intersection problem has been used extensively as a tool for proving that many algorithmic problems are as hard as solving 3SUM [30, 1, 23]. However, the vertex-triangles problem has yet to be considered directly[2].

The Lower Bounds. We use the vertex-triangles problem in order to show that the ISG problem is hard, and thus the simplest DMOG version of (offline) unbounded setting is 3SUM-hard. Our proof begins from the conditional lower bounds for triangle enumeration introduced in [23]. The most significant conditional lower bounds that we prove are stated by the following theorems. Due to space limitations the proofs of these theorems, together with some more conditional lower bounds, are given in Appendix A. To understand the statements of the following theorems, when the total query time of an algorithm can be formulated as $O(t_q + op \cdot t_r)$ time, we say that t_q is the query time and t_r is the reporting time.

▶ **Theorem 2.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the ISG problem on a graph G with m edges, if the amortized expected preprocessing time is $O(m \cdot \delta(G)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected query time must be $\Omega((\delta(G))^{1-o(1)})$.*

▶ **Theorem 3.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the DMOG problem on a dictionary D with d patterns, if the amortized expected preprocessing time is $O(|D| \cdot \delta(G_D)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected query time must be $\Omega((\delta(G_D))^{1-o(1)})$.*

[2] The closely related problem of deciding whether a given vertex is contained by any triangle (a decision version) has been addressed [8].

3 The Induced Subgraph Problem

An Upper Bound via Graph Orientations. In graph orientations the goal is to *orient* the graph edges while providing some guarantee on the out-degrees of the vertices. Formally, an orientation of an undirected graph $G = (V, E)$ is called a *c-orientation* if every vertex has out-degree at most $c \geq 1$. The notion of graph *degeneracy* is closely related to graph orientations [3]. Chiba and Nishizeki [13] linear time greedy algorithm assigns a $\delta(G)$-orientation of G. We preprocess G using this algorithm, thereby obtaining a c-orientation with $c = \delta(G)$, and use it for solving ISG problem as follows. First, we view an orientation as assigning "responsibility" for all data transfers occurring on an edge to one of its endpoints, depending on the direction of the edge in the orientation (regardless of the actual direction of the edge in the input graph G). We exploit this distinction by using the notation of an edge $e = (u, v)$ as oriented from u to v, while e could be directed either from u to v or from v to u. We say that u is *responsible* for e, and that e is *assigned* to u. Furthermore, u is a *responsible-neighbor* of v and v is an *assigned-neighbor* of u.

The Bipartite Graph. We begin by converting $G = (V, E)$ to a bipartite graph by creating two copies of V called L (the left vertices) and R (the right vertices). For every edge $(u, v) \in E$ we add an edge in the bipartite graph from $u_L \in L$ to $v_R \in R$, where u_L is a copy of u and v_R is a copy of v. All edges are originally directed from L to R (before the orientation). Furthermore, each vertex in V that arrives during query time is replaced by its two copies, first the copy from R and then the copy from L. This ordering guarantees that a self loop in G is not mistakenly reported the first time its single vertex arrives. Notice that the degeneracy of G is unchanged, up to constant factors, due to this reduction. From here onwards we assume that G is already in this bipartite representation.

The unbounded case discussion and the omitted proofs appear in [6].

3.1 Uniformly Bounded Edge Occurrences

In this case, the ISG problem is restricted with two positive integer parameters α and β so an edge (v_j, v_i) can only be reported if $\alpha < i - j \leq \beta + 1$ (recall that i and j are arrival times of v_i and v_j, respectively). The interval between β time units ago and α time units ago is called the *active window*. It is maintained via a list $\mathcal{L}_\beta$ of the last β vertices. In addition, each vertex $v \in R$ maintains a *reporting list* $\mathcal{L}_v$, which is a linked list containing the responsible-neighbors of v which have appeared during the active window, without repetition. Furthermore, each vertex $u \in L$ has an ordered list of time stamps τ_u of the times u arrived in the current active window.

At query time i, $\mathcal{L}_\beta$ is updated by removing $v_{i-\beta-1}$ and inserting v_i, which is the vertex arriving at time i. If $v_{i-\beta-1} \in L$ then the time stamp of $i - \beta - 1$ is removed from $\tau_{v_{i-\beta-1}}$. In case $\tau_{v_{i-\beta-1}}$ becomes empty then we remove $v_{i-\beta-1}$ from all of the reporting lists of its assigned-neighbors.

When a vertex v_i arrives, the data structures of the vertices are updated accordingly, as follows. If $v_i \in R$,

1. The elements of the reporting list $\mathcal{L}_{v_i}$ are scanned and their edges (u, v_i) are reported according to τ_u.
2. The edges for which v_i is their responsible-neighbour are scanned, and those for which the assigned-neighbour u is marked as arrived are reported.

If $v_{i-\alpha-1} \in L$,

1. $v_{i-\alpha-1}$ is marked as arrived.
2. If $\tau_{v_{i-\alpha-1}}$ was empty before the current arrival, then $v_{i-\alpha-1}$ is added to the reporting lists of its assigned neighbours.
3. $i-\alpha-1$ is added to $\tau_{v_{i-\alpha-1}}$.

▶ **Theorem 4.** *The Induced Subgraph problem with uniformly bounded edge occurrences on a graph G with m edges and n vertices can be solved with $O(m+n)$ preprocessing time, $O(\delta(G)+op)$ time per query vertex, where op is the number of edges reported at vertex arrival, and $O(m+\beta)$ space.*

3.2 Non-Uniformly Bounded Edge Occurrences

In non-uniformly bounded edge occurrences each edge $e=(v_j,v_i)$ has its own boundaries $[\alpha_e,\beta_e]$ and can only be reported if $\alpha_e < i-j < \beta_e+1$. Notice that in this case the input is a multi-graph. The active window for this ISG version is the time window between $\beta^* = \max_{e\in E}\{\beta_e\}$ and $\alpha^* = \min_{e\in E}\{\alpha_e\}$ time units ago.

Similar to Section 3.1, a dynamic list $\mathcal{L}_{\beta^*}$ of the last β^* vertices that have appeared is maintained. However, this approach of a general active window introduces a new challenge. If τ_u includes all the appearances of u within the active window, as was done in Section 3.1, when a vertex $v_i \in R$ arrives, the information in $\mathcal{L}_{v_i}$ cannot be automatically reported, as some of the appearances of nodes $u \in \mathcal{L}_{v_i}$ are not within the gaps of edge (u,v_i), thus only part of their τ_u list needs to be reported. A naive filtering considers for each $u \in \mathcal{L}_{v_i}$ a scan of τ_u and reports only time stamps j where $i-\beta_e < j < i-\alpha_e$, which sums up to $\beta^*-\alpha^*$ time per query vertex. To avoid an overhead in query time, our filtering mechanism checks all appearances of all responsible-neighbours of v_i in a batched query, where each responsible-neighbour appearance is filtered according to the edge's gaps. This is achieved by maintaining for each vertex $v \in R$ a fully dynamic data structure S_v for supporting 4-sided 2-dimensional orthogonal range reporting queries instead of $\mathcal{L}_v$. Given an $[x_0,y_0]\times[x_1,y_1]$-range, it returns the points of S_v that have (x,y) coordinates in the given range. For each responsible-neighbor $v_i \in L$ of v that arrived in the active window, where $e=(v_i,v)$, the point $(i+\alpha_e+1, i+\beta_e+1)$ is inserted into S_v, yielding the occurrences in S_v are from the "point of view" of v.

To implement S_v, we use Mortensen's data structure [26] that supports the set of $|S_v|$ points from $\mathbb{R}^2$ with $O(|S_v|\log^{7/8+\epsilon}|S_v|)$ words of space, insertion and deletion time of $O(\log^{7/8+\epsilon}|S_v|)$ and $O(\frac{\log|S_v|}{\log\log|S_v|}+op)$ time for range reporting queries on S_v, where *op* is the size of the output.

When a vertex v_i arrives at query time i, in addition to adding it to $\mathcal{L}_{\beta^*}$, the following happens. If $v_i \in R$,

1. A range query of $[0,i]\times[i,\infty]$ is performed over S_{v_i}. The edges representing the range output are reported.
2. The edges for which v_i is their responsible-neighbour are scanned, and those for which the assigned-neighbour u is marked as arrived are reported according to a search in their time stamp.

If $v_{i-\alpha^*-1} \in L$,

1. $v_{i-\alpha^*-1}$ is marked as arrived.
2. For each assigned-neighbour v, such that $e=(v_{i-\alpha^*-1},v)$, $(i-\alpha^*+\alpha_e, i-\alpha^*+\beta_e)$ is inserted to S_v.
3. $i-\alpha^*-1$ is added to $\tau_{v_{i-\alpha^*-1}}$.

▶ **Theorem 5.** *The Induced Subgraph problem with non-uniformly bounded edge occurrences on a graph G with m edges and n vertices can be solved with $O(m+n)$ preprocessing time, $\tilde{O}(\delta(G)+op)$ time per query vertex, where op is the number of edges reported due to the vertex arriving, and $\tilde{O}(m+\delta(G)(\beta^*-\alpha^*)+\alpha^*)$ space.*

4 Solving DMOG

4.1 DMOG via Graph Orientations

When extending ISG to online DMOG, the longer subpatterns introduce new challenges that need to be addressed. It is helpful to still consider the bipartite graph presentation of the DMOG instance, where vertices correspond to subpatterns and edges correspond to patterns. The algorithms from Section 3 are used as basic building blocks in our algorithms for DMOG by treating a subpattern arriving as the vertex arriving in the appropriate graph, while addressing the difficulties that arise from subpatterns being arbitrarily long strings.

First, a mechanism for determining when a subpattern arrives is needed. One way of doing this is by using the the Aho-Corasick (AC) Automaton [2], using a standard binary encoding technique so that each character costs $O(\log|\Sigma|)$ worst-case time. For simplicity we assume that $|\Sigma|$ is constant. However, while in the ISG problem each character corresponds to the arrival of at most one subpattern, in the DMOG with unbounded gaps each arriving character may correspond to several subpatterns which all arrive at once, since a subpattern could be a proper suffix of another subpattern. We, therefore, phrase the complexities of our algorithms in terms of lsc, which is the maximum number of vertices in the bipartite graph that arrive due to a character arrival. This induces a multiplicative overhead of at most lsc in the query time per text character relative to the time used by the ISG algorithms.

Finally, there is an issue arising from subpatterns no longer being of length one, which for simplicity we first discuss this in the unbounded case. When $u \in L$ arrives and it has an assigned vertex $v \in R$ where m_v is the length of the subpattern associated with v, then we do not want to report the edge (u,v) until at least m_v-1 time units have passed, since the appearance of the subpattern of v should not overlap with the appearance of the subpattern of u. Similarly, in the bounded case, we must delay the removal of u from $\mathcal{L}_v$ by at least m_v-1 time units. Notice that if we remove u from $\mathcal{L}_v$ after a delay of m_v-1, then we may be forced to remove a large number of such vertices at a given time. We, therefore, delay the removal of u by $M-1$ time units, where M is the length of the longest subpattern that corresponds to a vertex in R. This solves the issue of synchronization, however, some of the reporting lists now have elements that should not be reported. Nevertheless, we can spend time in a reporting list that corresponds to the size of the output using standard list and pointer techniques.

Combining these ideas with the algorithms in Section 3 gives Theorems 6, 7 and 8.

▶ **Theorem 6.** *The DMOG problem with one gap and unbounded gap borders can be solved with $O(|D|)$ preprocessing time, $O(\delta(G_D)\cdot lsc+op)$ time per text character, where op is the number of patterns that are reported due to the character arriving, and $O(|D|)$ space.*

▶ **Theorem 7.** *The DMOG problem with uniformly bounded gap borders can be solved such that dictionary patterns are reported online in: $O(|D|)$ preprocessing time, $O(\delta(G_D)\cdot lsc+op)$ time per text character, where op is the number of patterns that are reported due to the character arriving, and $O(|D|+lsc\cdot(\beta-\alpha+M)+\alpha)$ space.*

▶ **Theorem 8.** *The DMOG problem with non-uniformly bounded gap borders can be solved such that dictionary patterns are reported online in: $O(|D|)$ preprocessing time, $\tilde{O}(\delta(G_D)\cdot$*

$lsc + op$) time per text character, where op is the number of patterns that are reported due to the character arriving, and $\tilde{O}(|D| + lsc \cdot \delta(G_D)(\beta^* - \alpha^* + M) + \alpha^*)$ space.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 434–443, 2014.

2 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.

3 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

4 Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved dynamic dictionary matching. *Inf. Comput.*, 119(2):258–282, 1995.

5 Amihood Amir, Dmitry Keselman, Gad M. Landau, Moshe Lewenstein, Noa Lewenstein, and Michael Rodeh. Text indexing and dictionary matching with one error. *J. Algorithms*, 37(2):309–325, 2000. doi:10.1006/jagm.2000.1104.

6 Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap. *CoRR*, abs/1503.07563, 2015.

7 Amihood Amir, Avivit Levy, Ely Porat, and B. Riva Shalom. Dictionary matching with one gap. In *CPM*, pages 11–20, 2014.

8 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012.

9 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci.*, 443:25–34, 2012.

10 Philip Bille and Mikkel Thorup. Regular expression matching with multi-strings and intervals. In *Proc. of SODA*, pages 1297–1308, 2010.

11 A. Bjørklund, R. Pagh, V. Vassilevska Williams, and U. Zwick. Listing triangles. In *Proceedings 41st Int'l Colloquium on Automata, Languages, and Programming (ICALP (I))*, pages 223–234, 2014.

12 Gerth Stølting Brodal and Leszek Gasieniec. Approximate dictionary queries. In *Proc. of CPM*, pages 65–74, 1996.

13 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. doi:10.1137/0214017.

14 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010. doi:10.1016/j.tcs.2010.06.002.

15 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. of STOC*, pages 91–100, 2004.

16 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. *Inf. Retr.*, 11(4):335–357, 2008.

17 A. Grønlund and S. Pettie. Threesomes, degenerates, and love triangles. *CoRR*, abs/1404.0799, 2014.

18 Tuukka Haapasalo, Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. Online dictionary matching with variable-length gaps. In *Proc. of SEA*, pages 76–87, 2011.

19 Kay Hofmann, Philipp Bucher, Laurent Falquet, and Amos Bairoch. The PROSITE database, its status in 1999. *Nucleic Acids Research*, 27(1):215–219, 1999.

20 Wing-Kai Hon, Tak-Wah Lam, Rahul Shah, Sharma V. Thankachan, Hing-Fung Ting, and Yilin Yang. Dictionary matching with uneven gaps. In *CPM*, pages 247–260, 2015.

21 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic set intersection. In *WADS*, 2015.

23 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2016.

24 Gregory Kucherov and Michaël Rusinowitch. Matching a set of strings with variable length don't cares. *Theor. Comput. Sci.*, 178(1-2):129–154, 1997.

25 Michele Morgante, Alberto Policriti, Nicola Vitacolonna, and Andrea Zuccolo. Structured motifs search. *Journal of Computational Biology*, 12(8):1065–1082, 2005.

26 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006.

27 Eugene W. Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 39(2):430–448, 1992.

28 G. Myers and G. Mehldau. A system for pattern matching applications on biosequences. *CABIOS*, 9(3):299–314, 1993.

29 Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *Journal of Computational Biology*, 10(6):903–923, 2003. doi:10.1089/106652703322756140.

30 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *STOC*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

31 Verint. Personal communication, 2013.

32 Meng Zhang, Yi Zhang, and Liang Hu. A faster algorithm for matching a set of patterns with variable length don't cares. *Inf. Process. Lett.*, 110(6):216–220, 2010. doi:10.1016/j.ipl.2009.12.007.

A Full Details for Section 2

▶ **Theorem 9** ([23][3]). *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. Then for any constant $0 < x < 1/2$, any algorithm for enumerating all triangles in a graph G with m edges, $\Theta(m^{1-x})$ vertices, and $\hat{d} = \delta(G) = \Theta(m^x)$, where $\hat{d}$ is the average degree of a vertex in G, must spend $\Omega(m \cdot \delta(G)^{1-o(1)})$ expected time.*

▶ **Theorem 10.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the vertex-triangles problem on a graph G with m edges, if the amortized expected preprocessing time is $O(m \cdot \delta(G)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected query time must be at least $\Omega((\hat{d} \cdot \delta(G))^{1-o(1)})$, where $\hat{d}$ is the degree of the queried vertex.*

Proof. We reduce the triangle enumeration problem considered in Theorem 9 to the vertex-triangles problem. We preprocess G and then answer vertex-triangles queries on each of the m^{1-x} vertices thereby enumerating all of the triangles in G. If we assume a sub-polynomial reporting time, then by Theorem 9 either the preprocessing takes $\Omega(m \cdot \delta(G)^{1-o(1)})$ time or each query must cost at least $\Omega(\frac{m \cdot \delta(G)^{1-o(1)}}{m^{1-x}}) = \Omega((m^x \delta(G))^{1-o(1)}) = \Omega((\hat{d} \cdot \delta(G))^{1-o(1)})$ time. ◀

We are now ready to prove Theorems 2 and 3.

Proof of Theorem 2 and Theorem 3. We reduce the vertex-triangles problem considered in Theorem 10 to ISG as follows. We preprocess the graph G for ISG queries. Now, to

[3] The actual statement in [23] refers to the arboricity of G instead of the degeneracy of G. However, both terms are the same, up to a factor of 2.

answer a vertex-triangle query on some vertex u, we input all of the neighbors of u into the ISG algorithm. Thus, there is a one-to-one correspondence between the edges reported by the ISG algorithm and the triangles in the output of the vertex-triangles query. Since each vertex-triangle query must cost $\Omega(\hat{d} \cdot \delta(G)^{1-o(1)})$ amortized expected time then the amortized expected time spent for each of the $\hat{d}$ neighbors of u must be at least $\Omega(\delta(G)^{1-o(1)})$ amortized expected time. Since ISG is a special case of DMOG, and given Theorem 2, the proof of Theorem 3 follows directly. ◀

▶ **Theorem 11.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the uniformly bounded DMOG problem on a dictionary D with d patterns, if the amortized expected preprocessing time is $O(|D| \cdot \delta(G_D)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected time spent on each text character must be at least $\Omega((\beta - \alpha)^{1-o(1)})$.*

Proof. The proof is similar to the proofs of Theorems 2 and 3. First, we convert the input graph G of the vertex-triangles problem to a tripartite graph G_T by creating three copies of the vertices V_1, V_2, V_3 and for each edge (u, v) in G we add 6 edges to G_T between all possible copies of u and v. We also add a dummy vertex to G_T with degree 0. Each triangle in G corresponds to a constant number of triangles in G_T. Let α be any positive integer and let $\beta = \alpha + 2\hat{d}$. We use ISG to solve vertex-triangles queries in Theorem 2, but we only ask queries on the neighbors of vertices in V_1 in a specially tailored way as follows. We first list the neighbors of u from V_2, followed by α copies of the dummy vertex, and then list the neighbors from V_3. From the construction of the tripartite graph and the input to the ISG algorithm, two vertices of an edge that is part of the output of the ISG algorithm must be separated in the input list by at least α vertices, and by at most the length of the list which is β. Thus, the time spent on each vertex must be at least $\Omega(\delta(G)^{1-o(1)}) = \Omega((m^x)^{1-o(1)}) = \Omega((\beta - \alpha)^{1-o(1)})$ amortized expected time. Since ISG is a special case of DMOG, the theorem follows directly. ◀

Clustered Planarity with Pipes*

Patrizio Angelini[1] and Giordano Da Lozzo[2]

1 **Tübingen University, Tübingen, Germany**
angelini@informatik.uni-tuebingen.de
2 **Roma Tre University, Rome, Italy**
dalozzo@dia.uniroma3.it

Abstract

We study the version of the C-Planarity problem in which edges connecting the same pair of clusters must be grouped into pipes, which generalizes the Strip Planarity problem. We give algorithms to decide several families of instances for the two variants in which the order of the pipes around each cluster is given as part of the input or can be chosen by the algorithm.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Clustered Planarity, FPT, SEFE, Graph Drawing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.13

1 Introduction

Visualizing clustered graphs is a challenging task with several applications in the analysis of networks that exhibit a hierarchical structure. The most established criterion for a readable visualization of these graphs has been formalized in the notion of *c-planarity*, introduced by Feng, Cohen, and Eades [12] in 1995. Given a *clustered graph* $\mathcal{C}(G, \mathcal{T})$ (*c-graph*), that is, a graph G equipped with a recursive clustering $\mathcal{T}$ of its vertices, problem C-Planarity asks whether there exist a planar drawing of G and a representation of each cluster as a topological disk enclosing all and only its vertices, such that no *"unnecessary"* crossings occur between disks and edges, or between disks. Ever since its introduction, this problem has been attracting a great deal of research. However, the question about its computational complexity withstood the attack of several powerful algorithmic tools, as the Hanani-Tutte theorem [13, 15], the SPQR-tree machinery [9], and the Simultaneous PQ-ordering framework [5].

The clustering of a c-graph $\mathcal{C}(G, \mathcal{T})$ is described by a rooted tree $\mathcal{T}$ whose leaves are the vertices of G and whose each internal node μ, except for the root, represents a *cluster* containing all and only the leaves of the subtree of $\mathcal{T}$ rooted at μ. A c-graph is *flat* if $\mathcal{T}$ has height 2. The *clusters-adjacency graph* G_A of a flat c-graph is the graph obtained by contracting each cluster into a single vertex and by removing multi-edges and loops.

Cortese *et al.* [10] introduced a variant of C-Planarity for flat c-graphs, which we call C-Planarity with Embedded Pipes, whose input is a flat c-graph together with a planar drawing of its clusters-adjacency graph, where vertices are represented by disks and edges by pipes. The goal is to produce a c-planar drawing in which each vertex lies inside the disk representing the cluster it belongs to and each inter-cluster edge lies inside the corresponding

* This article reports on work supported by the U.S. Defense Advanced Research Projects Agency (DARPA) under agreement no. AFRL FA8750-15-2-0092. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This work was also supported in part by DFG grant Ka812/17-1 and by MIUR Project "AMANDA" under PRIN 2012C4E3KT.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 13; pp. 13:1–13:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

pipe. In [10] this problem is solved when the underlying graph is a cycle. Chang, Erickson, and Xu [8] observed that in this case the problem is equivalent to determining whether a closed walk of length n in a simple plane graph is weakly simple, and improved the time complexity to $O(n \log n)$. For the special case in which the clusters-adjacency graph is a path, known by the name of Strip Planarity, there exist polynomial-time algorithms when the underlying graph has a fixed planar embedding [2] and when it is a tree [13].

We remark that polynomial-time algorithms for the C-Planarity problem are known under strong limitations on the number or on the arrangement of the components in the clusters. A *component* of a cluster is a maximal connected subgraph induced by its vertices. In particular, C-Planarity can be decided in linear time when each cluster contains one connected component [9, 12] (the c-graph is *c-connected*). However, even when each cluster contains at most two connected components, polynomial-time algorithms are known only when further restrictions are imposed on the c-graph [5, 14]. The results we show in this paper are also based on imposing constraints on the number of certain types of components.

A component is *multi-edge* if it is incident to at least two inter-cluster edges, otherwise it is *single-edge*. Also, it is *passing* if it is adjacent to vertices belonging to at least two other clusters in $\mathcal{T}$, otherwise it is *originating*. For Strip Planarity the originating components can be further distinguished into *source* and *sink* components, based on whether the inter-cluster edges incident to them only belong to the lower or to the upper strip.

Our contributions. We give polynomial-time algorithms for instances of Strip Planarity with a unique source component (Section 3) and for instances of C-Planarity with Embedded Pipes with certain combinations of originating and passing multi-edge components in the clusters (Section 4). Finally, in Section 5 we introduce a generalization of C-Planarity with Embedded Pipes, which we call C-Planarity with Pipes, in which the inter-cluster edges are still required to be grouped into pipes, but the order of the pipes around each disk is not prescribed by the input. By introducing a new characterization of C-Planarity, we give an FPT algorithm for C-Planarity with Pipes that runs in $g(K, c) \cdot O(n^2)$ time, with $g(K, c) \in O(K^{c(K-2)})$, where K is the maximum number of multi-edge components in a cluster and c is the number of clusters with at least two multi-edge components. We remark that our results imply polynomial-time algorithms for all the three problems in the case in which each cluster contains at most two components.

Due to space limitations, complete proofs are deferred to the full version of the paper [1].

2 Preliminaries

For the standard definitions on planar graphs, planar drawings, planar embeddings, and connectivity we point the reader to [11]. We call *rotation scheme* the clockwise circular ordering of the edges around each vertex in a planar embedding, and refer to the containment relationships between vertices and cycles in the embedding as *relative positions*. Also, if block of a 1-connected graph consists of a single edge, we call it *trivial*, otherwise *non-trivial*.

PQ-trees. A *PQ-tree* [7] T is an unrooted tree, whose leaves are the elements of a set A and whose internal nodes are either *P-nodes* or *Q-nodes*, that can be used to represent all and only the circular orderings $\mathcal{O}(T)$ on A satisfying a given set of *consecutivity constraints* on subsets of A. The orderings in $\mathcal{O}(T)$ are all and only the circular orderings on the leaves of T obtained by arbitrarily ordering the neighbours of each P-node and by arbitrarily selecting for each Q-node a given circular ordering on its neighbours or its reverse ordering.

Connectivity. A *k-cut* of a graph is a set of at most k vertices whose removal disconnects the graph. A graph with no 1-cut is *biconnected*. The maximal biconnected components of a graph are its *blocks*. Without loss of generality, we will assume that the clusters-adjacency graph of $\mathcal{C}(G, \mathcal{T})$ is connected and that for every component c of every cluster $\mu \in \mathcal{T}$:

(i) there exists at least an inter-cluster edge incident to c,

(ii) every block of c that is a leaf in the block-cut-vertex tree of c contains at least a vertex v such that v is not a cut-vertex of c and it is incident to at least an inter-cluster edge, and

(iii) if there exists exactly one vertex in c that is incident to inter-cluster edges, then c consists of a single vertex.

Simultaneous Embedding with Fixed Edges. Given planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, problem SEFE asks whether there exist planar drawings Γ_1 of G_1 and Γ_2 of G_2 such that (i) any vertex $v \in V$ is mapped to the same point in Γ_1 and Γ_2 and (ii) any edge $e \in E_1 \cap E_2$ is mapped to the same curve in Γ_1 and Γ_2. Graphs $G_\cap = (V, E_1 \cap E_2)$ and $G_\cup = (V, E_1 \cup E_2)$ are the *common* and the *union* graph, respectively. See [4] for a survey.

We state here a theorem on SEFE that will be fundamental for our results. Even though this theorem has never been explicitly stated in the literature, it can be easily deduced from known results [6]. We discuss this in the full version of the paper [1].

▶ **Theorem 1.** *Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two planar graphs whose common graph $G_\cap = (V, E_1 \cap E_2)$ is a forest and whose cut-vertices are incident to at most two non-trivial blocks. It can be tested in $O(|V|^2)$ time whether $\langle G_1, G_2 \rangle$ admits a SEFE.*

3 Single-source Strip Planarity

In this section we prove a result of the same flavour as that by Bertolazzi *et al.* [3] for the upward planarity testing of single-source digraphs. Namely, we show that instances of Strip Planarity with a unique source component can be tested efficiently. The Strip Planarity problem takes in input a planar graph $G = (V, E)$ and a mapping $\gamma : V \to \{1, \dots, k\}$ of each vertex to one of k unbounded horizontal strips such that, for any edge $(u, v) \in E$, it holds $|\gamma(u) - \gamma(v)| \leq 1$. The goal is to find a planar drawing of G in which vertices lie inside the corresponding strips and edges cross the boundary of any strip at most once. This problem is equivalent to C-Planarity with Embedded Pipes when G_A is a path [2].

We start with an auxiliary lemma. An instance $\langle G, \gamma \rangle$ of Strip Planarity on $k > 1$ strips is *spined* if there exists a path $(v_1, \dots, v_k)$ in G such that $\gamma(v_i) = i$, vertex v_k is the unique vertex in the k-th strip, and each vertex v_i with $i \neq 1$ induces a component in the i-th strip. Path $(v_1, \dots, v_k)$ is the *spine path* of $\langle G, \gamma \rangle$ and (v_i, v_{i+1}) is the i-th edge of this path.

▶ **Lemma 2.** *Any positive spined instance $\langle G, \gamma \rangle$ of* Strip Planarity *admits a strip-planar drawing in which the intersection point between the first edge of the spine path of $\langle G, \gamma \rangle$ and the horizontal line separating the first and the second strip is the left-most intersection point between any inter-strip edge and such a line.*

▶ **Lemma 3.** *Let $\langle G = (V, E), \gamma \rangle$ be a spined instance of* Strip Planarity *on $k > 1$ strips with a unique source component c. It is possible to construct in linear time an equivalent spined instance $\langle G' = (V', E'), \gamma' \rangle$ on $k - 1$ strips with a unique source component c'.*

Proof Sketch. First note that the source component c lies in the first strip. We construct an auxiliary planar graph G_c as follows. Initialize $G_c = c$ and add a dummy vertex v to it.

For each inter-strip edge e incident to a vertex u in c, add to G_c a dummy vertex v_e and edges (v, v_e) and (v_e, u). If G_c has cut-vertices, then let B_c be the block of G_c that contains v. Then, construct a PQ-tree $\mathcal{T}_c$ representing all possible orders of the edges around v in a planar embedding of B_c. This can be done by applying the planarity testing algorithm by Booth and Lueker [7], in such a way that v is the last vertex of the *st*-numbering of B_c. Note that the leaves of PQ-tree $\mathcal{T}_c$ are in one-to-one correspondence with the vertices v_e in B_c. We construct a *representative graph* $G_{\mathcal{T}_c}$ from $\mathcal{T}_c$, as described in [12], composed of (i) *wheel* graphs (that is, graphs consisting of a cycle, called *rim*, and of a *central vertex* connected to every vertex of the rim), of (ii) edges connecting vertices of different rims not creating any simple cycle that contains vertices belonging to more than one wheel, and of (iii) vertices of degree 1, which are in one-to-one correspondence with the leaves of $\mathcal{T}_c$ (an hence with the dummy vertices v_e in B_c), each connected to a vertex of some rim. As proved in [12], in any planar embedding of $G_{\mathcal{T}_c}$ in which all the degree-1 vertices are incident to the same face, the order in which such vertices appear in a Eulerian tour of such a face is in $O(\mathcal{T}_c)$.

Construct $\langle G', \gamma' \rangle$ as follows. For $i = 2, \dots, k$ and for each vertex v with $\gamma(v) = i$, add v to V' and set $\gamma'(v) = i - 1$, that is, assign all the vertices of the i-th strip of $\langle G, \gamma \rangle$, with $i \geq 2$, to the $(i-1)$-th strip of $\langle G', \gamma' \rangle$. Further, add to E' all edges in $E \cap (V' \times V')$. Also, add all vertices and edges of $G_{\mathcal{T}_c}$ to V' and to E', respectively, and set $\gamma'(u) = 1$ for each vertex u of $G_{\mathcal{T}_c}$. Finally, for each inter-strip edge $e = (x, y)$ in E with $\gamma(x) = 1$ and $\gamma(y) = 2$, add to E' an intra-strip edge between y and the degree-1 vertex of $G_{\mathcal{T}_c}$ corresponding to v_e.

Instance $\langle G', \gamma' \rangle$ can be constructed in linear time [7, 12] and its size is linear in the one of $\langle G, \gamma \rangle$. Further, $\langle G', \gamma' \rangle$ has a unique source component, which contains $G_{\mathcal{T}_c}$ as a subgraph, and is spined. We now show the equivalence between the two instances.

Suppose that $\langle G, \gamma \rangle$ admits a strip-planar drawing Γ. Note that all the vertices of c incident to inter-strip edges lie on the outer face of c in Γ. To construct a strip-planar drawing Γ' of $\langle G', \gamma' \rangle$, subdivide each inter-strip edge incident to c with a dummy vertex v_e lying in the interior of the first strip of Γ. By the construction of $\mathcal{T}_c$ and of $G_{\mathcal{T}_c}$, each vertex v_e corresponds to exactly one degree-1 vertex of $G_{\mathcal{T}_c}$. Let c^+ be the subgraph of G induced by the vertices in c and by all the vertices v_e. Since the order in which the vertices v_e appear in a Eulerian tour of the outer face of c^+ in Γ is in $O(\mathcal{T}_c)$, we can replace the drawing of c^+ in Γ with a drawing of $G_{\mathcal{T}_c}$ in which each degree-1 vertex is mapped to its corresponding vertex v_e. To obtain Γ', we merge the first two strips of Γ into the first strip of Γ'.

Suppose that $\langle G', \gamma' \rangle$ admits a strip-planar drawing Γ', we show how to construct a strip-planar drawing Γ of $\langle G, \gamma \rangle$. First, by Lemma 2, we can assume that in Γ' the intersection point between the first edge of the spine path of $\langle G', \gamma' \rangle$ and the line separating the first and the second strip in Γ' is the left-most intersection point between any edge (x, y) with $\gamma(x) = 1$ and $\gamma(y) = 2$ and such a line. Further, we can assume the following.

▶ **Claim 4.** *For every wheel W in $G_{\mathcal{T}_c}$, the rim of W contains in its interior its central vertex and no other vertex in Γ'.*

Initialize Γ as the drawing in Γ' of the subinstance of $\langle G', \gamma' \rangle$ induced by the vertices not in $G_{\mathcal{T}_c}$, where the i-th strip in Γ' is mapped to the $(i+1)$-th strip in Γ. First, draw $G_{\mathcal{T}_c}$ in the first strip of Γ as it is drawn in Γ'. Then, draw each inter-strip edge (x, y) with y in $G_{\mathcal{T}_c}$, which corresponds to an intra-strip edge incident to $G_{\mathcal{T}_c}$ in Γ', as a curve composed of six parts. The first part coincides with the drawing of (x, y) in Γ'; the second is a curve arbitrarily close to the drawing in Γ' of a path in $G_{\mathcal{T}_c}$ from y to the first vertex v_1 of the spine path of $\langle G', \gamma' \rangle$; the third is a curve arbitrarily close to the drawing in Γ' of the first edge of the spine path of $\langle G', \gamma' \rangle$ till a point p in the interior of the first strip of Γ' and arbitrarily close to the boundary of the second strip; the fourth is a horizontal segment connecting p to a point q lying to the left of Γ'; the fifth is a vertical segment connecting q to a point r in the interior of the first strip of Γ; and the sixth is a curve connecting r to y. By Claim 4, the degree-1 vertices of $G_{\mathcal{T}_c}$ lie on its outer face in Γ' (and hence in Γ). Thus, the inter-strip edges incident to $G_{\mathcal{T}_c}$ can be drawn without crossings, as they preserve the same containment relationship between vertices and cycles in Γ as the corresponding intra-strip edges in Γ'.

Let H be the graph obtained from B_c by subdividing each edge e incident to v with a dummy vertex v_e and by removing v. Replace the drawing of $G_{\mathcal{T}_c}$ in Γ with a planar drawing of H such that the vertices v_e appear in a Eulerian tour of its outer face in the same clockwise order as the corresponding degree-1 vertices appear in a Eulerian tour of the outer face of $G_{\mathcal{T}_c}$ in Γ. Recall that these vertices are on the outer face of $G_{\mathcal{T}_c}$ in Γ, by Claim 4. Such a drawing of H exists since this order is in $O(\mathcal{T}_c)$ [12]. To complete Γ, for each cut-vertex z of G_c separating B_c from a subgraph G_z of G_c, draw graph G_z arbitrarily close to z. Note that no vertex of G_z, except possibly for z, is incident to an inter-strip edge. ◀

Let $\langle G, \gamma \rangle$ be an instance of Strip Planarity on $k > 1$ strips satisfying the properties of Lemma 3. By applying this lemma $k-1$ times, we obtain an instance of Strip Planarity on $k = 1$ strips, that is, an instance whose strip-planarity coincides with the planarity of its underlying graph, which can be tested in linear time [7]. Hence, we get the following.

▶ **Lemma 5.** *Let $\langle G = (V, E), \gamma \rangle$ be a spined instance of* Strip Planarity *on $k > 1$ strips with a unique source component c. It is possible to decide in $O(k \times n)$ time whether $\langle G, \gamma \rangle$ admits a strip-planar drawing.*

Given an instance of Strip Planarity, one can create $O(n)$ spined instances by attaching the spine path to each of the $O(n)$ vertices in the first strip. The next theorem follows.

▶ **Theorem 6.** *Let $\langle G, \gamma \rangle$ be an instance of* Strip Planarity *on k strips such that there exists a unique source component c. It is possible to decide in $O(n^3)$ time whether $\langle G, \gamma \rangle$ admits a strip-planar drawing.*

4 C-Planarity with Embedded Pipes

In this section we show that the C-Planarity with Embedded Pipes problem is solvable in quadratic time for a notable family of instances.

Let c be an originating component belonging to a cluster $\mu \in \mathcal{T}$ and let $\nu \neq \mu \in \mathcal{T}$ be the cluster to which the vertices of c are adjacent to. We say that c is *originating from μ to ν*.

▶ **Lemma 7.** *Let $\langle \mathcal{C}(G, \mathcal{T}), \Gamma_A \rangle$ be an instance of* C-Planarity with Embedded Pipes *and let $\mathcal{S}$ be the maximum number of originating multi-edge components in a cluster that are incident to the same pipe. It is possible to construct in linear time an equivalent instance $\langle G_1, G_2 \rangle$ of SEFE such that (i) $G_\cap$ is a spanning forest, (ii) each cut-vertex of $G_2 = (V, E_2)$ is incident to at most one non-trivial block, and (iii) each cut-vertex of $G_1 = (V, E_1)$ is incident to at most $\mathcal{S}$ non-trivial blocks.*

Proof. We show how to construct $\langle G_1, G_2 \rangle$ starting from $\langle \mathcal{C}(G, \mathcal{T}), \Gamma_A \rangle$. The *frame gadget* H is an embedded planar graph defined as follows. Refer to Fig. 1.a. For each intersection point between a disk representing a cluster $\mu \in \mathcal{T}$ and a segment delimiting a pipe representing an edge of G_A incident to μ in the drawing Γ_A of G_A, we add a vertex at this point. This results in a planar drawing of a graph; we set H to be this graph. We call *disk cycle* of μ the cycle in H obtained from the disk of μ in Γ_A. Similarly, we call *pipe cycle* of an edge (μ, ν)

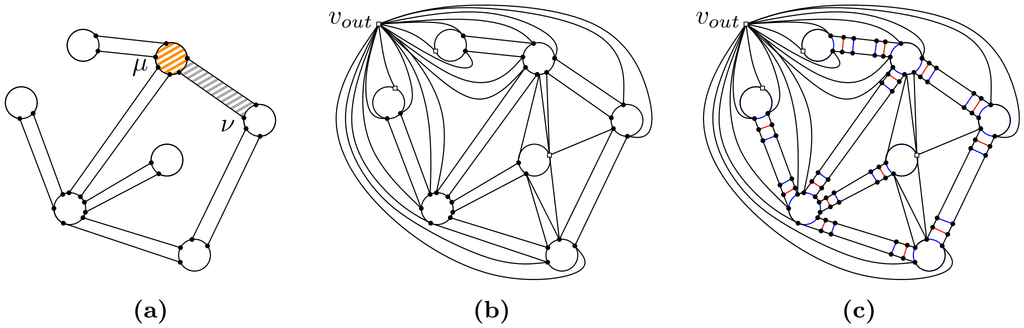


Figure 1 (a) Drawing Γ_A of the clusters-adjacency graph G_A, with vertices at the intersections of disks and pipes. The disk cycle for cluster μ and the pipe cycle for edge (μ, ν) of G_A are orange and gray tiled regions, respectively. (b) Frame gadget H. (c) Partial instance $\langle G_1, G_2 \rangle$ of SEFE constructed from Γ_A; graphs G_1, G_2, and $G_\cap$ are subdivisions of triconnected planar graphs.

of G_A the cycle in H obtained from the pipe representing edge (μ, ν) in Γ_A. Note that, for clusters incident to exactly one pipe, this operation introduced two copies of the same edge; subdivide with a dummy vertex the copy that is not incident to the interior of this pipe. Then, add a vertex v_{out} in the outer face of H, connected to all the vertices incident to this face, and triangulate all the faces of H not corresponding to the interior of any cluster cycle or of any pipe cycle, hence obtaining a triconnected embedded planar graph. See Fig. 1.b.

Initialize $G_\cap = H$. For each edge $e \in E(H)$ separating a pipe from a disk, remove e from G_1 (not from G_2); this implies that disk cycles and pipe cycles only belong to G_2. Further, for each two edges e' and e'' corresponding to the two segments $(u_{\mu,\nu}, u_{\nu,\mu})$ and $(v_{\mu,\nu}, v_{\nu,\mu})$ delimiting a pipe representing an edge (μ, ν) of G_A, subdivide e' with four dummy vertices $a'_{\mu,\nu}, b'_{\mu,\nu}, b'_{\nu,\mu}, a'_{\nu,\mu}$, and e'' with four dummy vertices $a''_{\mu,\nu}, b''_{\mu,\nu}, b''_{\nu,\mu}, a''_{\nu,\mu}$, and add edges $(a'_{\mu,\nu}, a''_{\mu,\nu})$ and $(a'_{\nu,\mu}, a''_{\nu,\mu})$ to G_1 and edges $(b'_{\mu,\nu}, b''_{\mu,\nu})$ and $(b'_{\nu,\mu}, b''_{\nu,\mu})$ to G_2.

For each cluster $\mu \in \mathcal{T}$, augment $\langle G_1, G_2 \rangle$ as follows; see Fig. 2.a. Subdivide an edge of $G_\cap$ that corresponds to a portion of the boundary of the disk representing μ in Γ_A with a dummy vertex γ_μ, and add to $G_\cap$ a star C_μ, whose central vertex is adjacent to γ_μ, with a leaf $z(c_i)$ for each multi-edge component c_i of μ. Also, add to $G_\cap$ each component c_i of μ. Finally, for each edge (μ, ν) of G_A, subdivide $(v_{\mu,\nu}, a'_{\mu,\nu})$ with a dummy vertex $\alpha_{\mu,\nu}$ and $(a''_{\mu,\nu}, b''_{\mu,\nu})$ with a dummy vertex $\beta_{\mu,\nu}$. Add to $G_\cap$ a star $A_{\mu,\nu}$ ($B_{\mu,\nu}$), whose central vertex is adjacent to $\alpha_{\mu,\nu}$ (is identified with $\beta_{\mu,\nu}$), with a leaf $a_\mu(e)$ (a leaf $b_\mu(e)$) for each inter-cluster edge e incident to a component of μ and to a component in ν. To complete $\langle G_1, G_2 \rangle$, add the following edges only belonging to G_1 and to G_2. For each inter-cluster edge $e = (x, y)$ with $x \in \mu$ and $y \in \nu$, add to G_1 edges $(x, a_\mu(e))$, $(y, a_\nu(e))$, and $(b_\mu(e), b_\nu(e))$, and add to G_2 edges $(a_\mu(e), b_\mu(e))$ and $(a_\nu(e), b_\nu(e))$. Also, for each vertex x of a component c_i of a cluster μ such that x is incident to at least an inter-cluster edge, add to G_2 an edge $(x, z(c_i))$.

Clearly, $\langle G_1, G_2 \rangle$ can be constructed in linear time. We now prove that G_1 and G_2 satisfy the properties of the lemma. We note that G_1 and G_2 are connected, since each vertex of a component c_i is connected to the frame gadget by means of paths in G_1 and in G_2 passing through stars $A_{\mu,\nu}$ and C_μ, respectively. Also, for each cluster $\mu \in \mathcal{T}$, graph G_2 contains cut-vertices γ_μ, the center of star C_μ, and vertices $z(c_i)$, for each component c_i of μ. However, vertex γ_μ is incident to exactly one non-trivial block, that is, the one containing all the vertices and edges of the frame gadget; the center of C_μ is incident only to non-trivial blocks; and vertices $z(c_i)$, for each component c_i of μ, are incident to at most one non-trivial block, that is, the one containing all the vertices and edges in c_i. Also, for each cluster $\mu \in \mathcal{T}$,

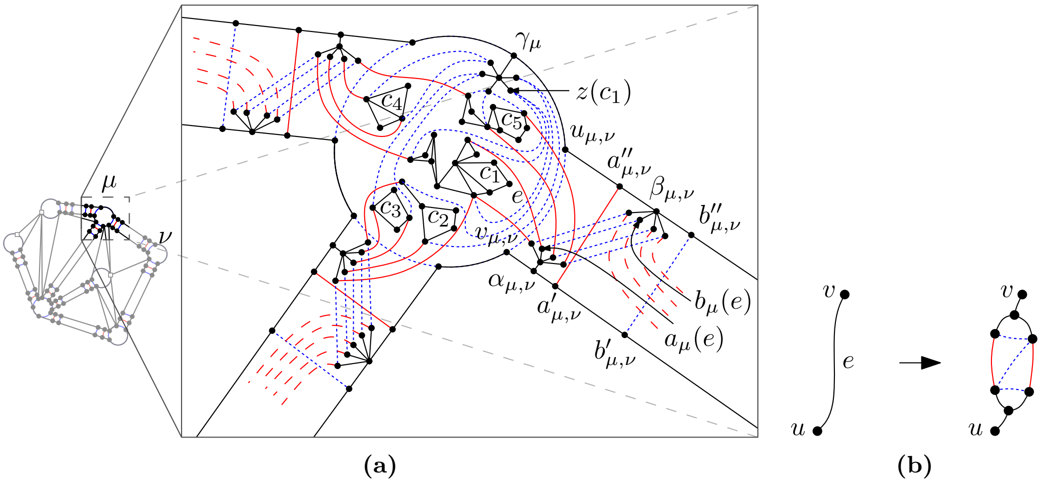


Figure 2 (a) Augmentation of instance $\langle G_1, G_2 \rangle$ focused on cluster $\mu \in \mathcal{T}$. (b) Replacing an edge $e = (u, v)$ to make $G_\cap$ acyclic.

all the passing components in μ belong to the biconnected component of G_1 containing all the vertices and edges of the frame gadget, while each multi-edge component originating from μ to a cluster ν determines a non-trivial block incident to $\alpha_{\mu,\nu}$, and each single-edge originating component from μ to a cluster ν determines a trivial block incident to $\alpha_{\mu,\nu}$. Since the number of multi-edge components originating from any cluster to any other cluster is at most $\mathcal{S}$, graph G_1 satisfies the required properties. The following claim implies that $G_\cap$ can be transformed into a spanning forest without altering the properties of $\langle G_1, G_2 \rangle$.

▶ **Claim 8.** *Each cycle of $G_\cap$ can be removed without altering the properties of $\langle G_1, G_2 \rangle$ by replacing one of its edges with the gadget in Fig. 2.b.*

We now prove the equivalence. Suppose that $\langle G_1, G_2 \rangle$ admits a SEFE $\langle \Gamma_1, \Gamma_2 \rangle$. We show how to construct a c-planar drawing with embedded pipes Γ of $\langle \mathcal{C}(G, \mathcal{T}), \Gamma_A \rangle$. Without loss of generality, assume that vertex v_{out} is embedded on the outer face of $\langle \Gamma_1, \Gamma_2 \rangle$. Note that the paths in $G_\cap$ corresponding to the segments delimiting the pipes representing an edge of G_A incident to a cluster $\mu \in \mathcal{T}$ appear in $\langle \Gamma_1, \Gamma_2 \rangle$ in the same clockwise circular order as the corresponding pipes appear around the disk representing μ in Γ_A. This is due to the fact that the frame gadget is a triconnected planar graph whose unique planar embedding is the one obtained from Γ_A. Note that in $\langle \Gamma_1, \Gamma_2 \rangle$ all the vertices in V appear either in the interior or on the boundary of disk cycles or of pipe cycles. This is due to the fact that removing all the vertices on the boundary of such cycles leaves a connected subgraph of $G_\cup$ and that there exists a unique face of H to which all the vertices belonging to such cycles are incident.

The proof is based on the fact that any SEFE of $\langle G_1, G_2 \rangle$ has the following properties.

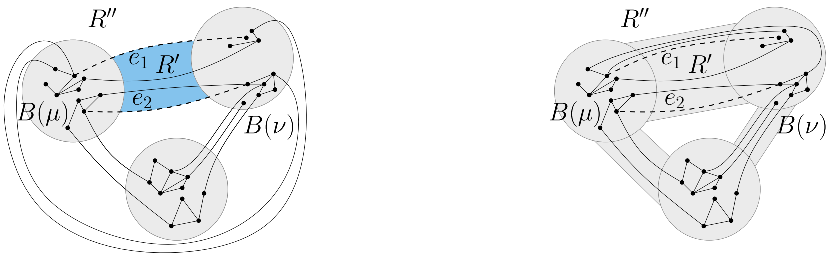
1. For each cluster $\mu \in \mathcal{T}$, the central vertex of star C_μ lies in the interior of the disk cycle of μ, and hence all the vertices and edges of the components c_i of μ lie in the interior of such a cycle, since G_2 is connected.
2. For each two clusters $\mu, \nu \in \mathcal{T}$, the vertices of the components of μ and of the those of ν lie in the interior of different cycles of G_1, since all the components of each cluster μ are connected by paths in G_1 to the leaves of a star $A_{\mu,\xi}$, where ξ is a cluster adjacent to μ. Also, all the leaves of these stars lie in the interior of a cycle of G_1 delimited by edges of $G_\cap$ and by edges $(a'_{\mu,\xi_i}, a''_{\mu,\xi_i})$, for all the clusters ξ_i adjacent to μ.

3. For each inter-cluster edge e connecting a vertex v of a component c_i of μ to a cluster ν, edge $(v, a_\mu(e))$ in G_1 crosses edge $(u_{\mu,\nu}, v_{\mu,\nu})$. This is due to the previous two points and the fact that the leaves of $A_{\mu,\nu}$ lie outside the disk cycle of μ. We can assume that each of these edges crosses edge $(u_{\mu,\nu}, v_{\mu,\nu})$ exactly once, as otherwise we could redraw them to fulfill this requirement.
4. For two adjacent clusters $\mu, \nu \in \mathcal{T}$, the order in which the edges in G_1 incident to the leaves of $A_{\mu,\nu}$ cross $(u_{\mu,\nu}, v_{\mu,\nu})$ from $u_{\mu,\nu}$ to $v_{\mu,\nu}$ is the reverse of the order in which the edges in G_1 incident to the leaves of $A_{\nu,\mu}$ cross $(u_{\nu,\mu}, v_{\nu,\mu})$ from $u_{\nu,\mu}$ to $v_{\nu,\mu}$, where the identification between an edge incident to a leaf $a_\mu(e)$ of $A_{\mu,\nu}$ and an edge incident to a leaf $a_\nu(e)$ of $A_{\nu,\mu}$ is based on the inter-cluster edge e they correspond to. In fact, the order in which the edges in G_1 incident to the leaves of $A_{\mu,\nu}$ cross $(u_{\mu,\nu}, v_{\mu,\nu})$ is transmitted to the leaves of $B_{\mu,\nu}$ via edges in G_2, then it is transmitted to the leaves of $B_{\nu,\mu}$ via edges in G_1, and finally to the leaves of $A_{\nu,\mu}$ via edges in G_2. Note that all the leaves of these stars lie in the interior of the pipe cycle corresponding to edge (μ, ν) of G_A.

We describe how to obtain a c-planar drawing with embedded pipes Γ of $\langle \mathcal{C}(G, \mathcal{T}), \Gamma_A \rangle$ from $\langle \Gamma_1, \Gamma_2 \rangle$. For each $\mu \in \mathcal{T}$, draw region $R(\mu)$ as the simple closed region whose boundary coincides with the drawing in Γ_2 of the disk cycle of μ. Each component c_i of a cluster μ has the same drawing in Γ as c_i in $\langle \Gamma_1, \Gamma_2 \rangle$. For each inter-cluster edge $e = (x, y)$ with $x \in \mu$ and $y \in \nu$, the portion of e in the interior of $R(\mu)$ (of $R(\nu)$) coincides with the drawing of edge $(x, a_\mu(e))$ (of edge $(y, a_\nu(e))$) between x (between y) and the intersection point of this edge with edge $(u_{\mu,\nu}, v_{\mu,\nu})$ (with edge $(u_{\nu,\mu}, v_{\nu,\mu})$). To complete the drawing of all the inter-cluster edges between μ and ν in the interior of the pipe representing edge (μ, ν) of G_A, connect the intersection points between the corresponding edges in G_1 and edges $(u_{\mu,\nu}, v_{\mu,\nu})$ and $(u_{\nu,\mu}, v_{\nu,\mu})$ by means of a set of non-intersecting curves. This is possible since the order in which the edges in G_1 incident to the leaves of $A_{\mu,\nu}$ cross $(u_{\mu,\nu}, v_{\mu,\nu})$ from $u_{\mu,\nu}$ to $v_{\mu,\nu}$ is the reverse of the order in which the edges in G_1 incident to the leaves of $A_{\nu,\mu}$ cross $(u_{\nu,\mu}, v_{\nu,\mu})$ from $u_{\nu,\mu}$ to $v_{\nu,\mu}$. This implies that Γ is a c-planar drawing of $\mathcal{C}(G, \mathcal{T})$. The fact that Γ can be continuously deformed into a c-planar drawing with embedded pipes of $\langle \mathcal{C}(G, \mathcal{T}), \Gamma_A \rangle$ is due to the fact that the paths in $G_\cap$ corresponding to the segments delimiting the pipes incident to each cluster $\mu \in \mathcal{T}$ appear in $\langle \Gamma_1, \Gamma_2 \rangle$ in the same clockwise order as the corresponding pipes appear around the disk representing μ in Γ_A.

For the other direction, the goal is to construct a SEFE $\langle \Gamma_1, \Gamma_2 \rangle$ of $\langle G_1, G_2 \rangle$ that satisfies all the properties describe above starting from a c-planar drawing with pipes Γ of $\langle \mathcal{C}(G, \mathcal{T}), \Gamma_A \rangle$. For each cluster $\mu \in \mathcal{T}$, draw the disk cycle of μ as the boundary of the disk of μ in Γ_A. Also, for each edge (μ, ν) of G_A, draw the corresponding pipe cycle as the boundary of the pipe of edge (μ, ν) in Γ_A. For each cluster $\mu \in \mathcal{T}$, each component c_i of μ has the same drawing in $\langle \Gamma_1, \Gamma_2 \rangle$ as c_i in Γ. For each edge (μ, ν) of G_A, the stars $A_{\mu,\nu}$, $B_{\mu,\nu}$, $A_{\nu,\mu}$, and $B_{\nu,\mu}$ are drawn in $\langle \Gamma_1, \Gamma_2 \rangle$ so that the order of their leaves is the same or the reverse of the order in which the inter-cluster edges between μ and ν traverse the boundary of the disk of μ in Γ. Note that this order is the reverse of the order in which these edges traverse the boundary of the disk of ν in Γ. This allows to draw all the edges in G_1 and in G_2 that are incident to such leaves without introducing crossings between edges of the same graph. The drawing of star C_μ, for each cluster $\mu \in \mathcal{T}$, and of the edges in G_2 incident to its leaves can be easily obtained to respect the circular order of the inter-cluster edges incident to each of the components of μ. This concludes the proof of the lemma. ◀

By Lemma 7 and Theorem 1 we have the following main result.



(a) **(b)**

Figure 3 (a) A c-planar drawing with pipes Γ', with regions R' (blue) and R'' delimited by $B(\mu)$ and $B(\nu)$, and by e_1 and e_2 (dashed), where region R' does not contain any vertex of $G \setminus (\mu \cup \nu)$. (b) A c-planar drawing with pipes Γ^* corresponding to Γ' in which inter-cluster edges are inside pipes.

▶ **Theorem 9.** C-Planarity with Embedded Pipes *can be solved in $O(n^2)$ time for instances $\langle \mathcal{C}(G,\mathcal{T}), \Gamma_A \rangle$ such that for each cluster $\mu \in \mathcal{T}$ and for each edge (μ,ν) in G_A either (CASE 1) cluster μ contains at most one originating multi-edge component from μ to ν or (CASE 2) cluster μ contains at most two multi-edge originating components from μ to ν and does not contain any passing component that is incident to ν.*

5 C-Planarity with Pipes

In this section we introduce and study problem C-Planarity with Pipes. A c-planar drawing Γ of a flat c-graph $\mathcal{C}(G,\mathcal{T})$ is a *c-planar drawing with pipes* if, for any two clusters $\mu, \nu \in \mathcal{T}$ that are adjacent in G_A and for any two inter-cluster edges e_1 and e_2 that are incident to both μ and ν, one of the two regions delimited by $B(\mu)$, by $B(\nu)$, by e_1, and by e_2 does not contain any vertex of $G \setminus (\mu \cup \nu)$; see Fig. 3.

Problem C-Planarity with Pipes asks for the existence of a c-planar drawing with pipes of a given flat c-graph. We first prove that this problem is a generalization of C-Planarity with Embedded Pipes.

▶ **Lemma 10.** C-Planarity with Embedded Pipes *reduces in linear time to* C-Planarity with Pipes. *The reduction does not increase the number of multi-edge components in any cluster.*

We now present an FPT algorithm for C-Planarity with Pipes with two parameters, namely the maximum number K of multi-edge components in a cluster and the number c of clusters with at least two multi-edge components. Our result builds on a characterization of C-Planarity of flat c-graphs in terms of a new constrained embedding problem.

5.1 A Characterization of Flat C-Planarity

We start with some definitions. Let $\mathcal{C}(G,\mathcal{T})$ be a flat c-graph. A *components tree* X_μ of a cluster $\mu \in \mathcal{T}$ is a rooted tree in which every internal vertex is a multi-edge component c of μ and in which every leaf $x_\mu(e)$ corresponds to an inter-cluster edge e incident to one of such components. A *neighbor-clusters tree* Y_μ of μ is a rooted tree in which there exists an internal vertex ν for each cluster ν adjacent to μ, plus a set of additional internal vertices, and every leaf $y_\mu(e)$ corresponds to an inter-cluster edge e incident to μ. Let Γ be a c-planar drawing of $\mathcal{C}(G,\mathcal{T})$, let X_μ be a components tree of μ rooted at a multi-edge component ρ_μ, and let Y_μ be a neighbor-clusters tree of μ rooted at a cluster ξ_μ, such that there exists an

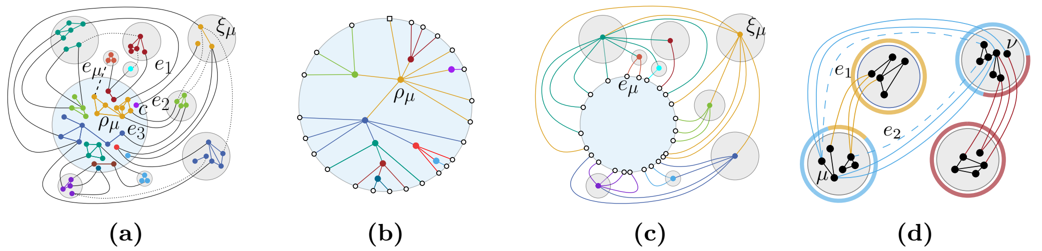


Figure 4 (a) A c-planar drawing Γ focused on cluster μ. Edges incident to μ are solid. Component c is nested into component ρ_μ. Trees (b) X_μ and (c) Y_μ such that Γ is consistent with X_μ and Y_μ. (d) A c-planar drawing that is not a c-planar drawing with pipes, even if the inter-cluster edges incident to the same cluster are consecutive (see the annuli around clusters), due to the presence of trivial block (μ, ν).

inter-cluster edge e_μ incident to both ρ_μ and ξ_μ. Let $\mathcal{O}_\mu$ be the clockwise linear order in which the edges incident to μ traverse $B(\mu)$ in Γ, starting from and ending at e_μ. Drawing Γ is *consistent with* X_μ if, for each vertex $c \in X_\mu$, the leaves of the subtree of X_μ rooted at c are consecutive in the restriction of $\mathcal{O}_\mu$ to the inter-cluster edges incident to multi-edge components of μ. Also, Γ is *consistent with* Y_μ if, for each vertex $\nu \in Y_\mu$, the leaves of the subtree of Y_μ rooted at ν are consecutive in $\mathcal{O}_\mu$. Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets containing a components tree X_μ and a neighbor-clusters tree Y_μ, respectively, for each $\mu \in \mathcal{T}$. Drawing Γ is *consistent with* $\langle \mathcal{X}, \mathcal{Y} \rangle$ if, for each $\mu \in \mathcal{T}$, it is consistent with both X_μ and Y_μ.

Given a flat c-graph $\mathcal{C}(G, \mathcal{T})$, together with two sets $\mathcal{X}$ and $\mathcal{Y}$ of components trees and of neighbor-clusters trees, respectively, for all the clusters in $\mathcal{T}$, problem Inclusion-Constrained C-Planarity asks whether a c-planar drawing of $\mathcal{C}(G, \mathcal{T})$ exists that is consistent with $\langle \mathcal{X}, \mathcal{Y} \rangle$.

▶ **Theorem 11.** *A flat c-graph $\mathcal{C}(G, \mathcal{T})$ is c-planar if and only if there exist two sets $\mathcal{X}$ and $\mathcal{Y}$ of components trees and of neighbor-clusters trees, respectively, for all the clusters in $\mathcal{T}$, such that $\langle \mathcal{C}(G, \mathcal{T}), \mathcal{X}, \mathcal{Y} \rangle$ is a positive instance of* Inclusion-Constrained C-Planarity.

Proof Sketch. The "only if part" trivially follows from the definition of Inclusion-Constrained C-Planarity. For the "if part", let Γ be any c-planar drawing of $\mathcal{C}(G, \mathcal{T})$ and let μ be a cluster in $\mathcal{T}$. Suppose that μ contains at least a multi-edge component ρ_μ, as otherwise X_μ is the empty tree and Γ is consistent with it. Let e_μ be any inter-cluster edge incident to ρ_μ and to a cluster ξ_μ. Let $\mathcal{O}_\mu$ be the clockwise linear order of the edges incident to μ starting from e_μ and ending at e_μ. Since Γ is c-planar, no two pairs of edges incident to two different components of μ (two different clusters adjacent to μ) alternate in $\mathcal{O}_\mu$. Hence, order $\mathcal{O}_\mu$ defines a hierarchical inclusion of the components of μ and of the clusters adjacent to μ with respect to ρ_μ and to ξ_μ, respectively, which can be described by means of two trees X_μ and Y_μ; see Fig. 4.a–4.c. Clearly, Γ is consistent with such trees. ◀

In the following theorem, whose proof is deferred to the full version of the paper [1], we show that the Inclusion-Constrained C-Planarity problem can be solved efficiently.

▶ **Theorem 12.** Inclusion-Constrained C-Planarity *can be solved in quadratic time.*

In the following we prove that, for each cluster μ of a c-graph $\mathcal{C}(G, \mathcal{T})$, there exists a neighbor-clusters tree Y_μ such that every c-planar drawing with pipes of $\mathcal{C}(G, \mathcal{T})$ is consistent with Y_μ. Hence, an FPT algorithm for C-Planarity with Pipes can be based on generating, for each cluster, all the possible components trees and its unique neighbor-clusters tree, and on testing these instances of Inclusion-Constrained C-Planarity by Theorem 12.

5.2 Neighbor-clusters Trees in C-Planar Drawings with Pipes

In the following theorem we give a characterization of the c-graphs that are positive instances of C-Planarity with Pipes based on the possible orders of inter-cluster edges around each cluster in any c-planar drawing. We first consider only c-graphs whose clusters-adjacency graph G_A has no trivial blocks; however, we prove later that this is not a restriction.

▶ **Theorem 13.** *Let $\mathcal{C}(G,\mathcal{T})$ be a flat c-graph such that G_A has no trivial block. Then, $\mathcal{C}(G,\mathcal{T})$ is a positive instance of* C-Planarity with Pipes *if and only if $\mathcal{C}(G,\mathcal{T})$ admits a c-planar drawing Γ in which, for each cluster $\mu \in \mathcal{T}$, the inter-cluster edges between μ and any cluster ν adjacent to μ in G_A are consecutive in the order in which the inter-cluster edges incident to μ cross $B(\mu)$ in Γ.*

Proof Sketch. The "only if part" descends from the definition of a c-planar drawing with pipes. We prove the "if part"; see Fig. 4.d. Let Γ be a c-planar drawing of $\mathcal{C}(G,\mathcal{T})$ satisfying the conditions of the theorem and consider two clusters $\mu, \nu \in \mathcal{T}$ with two inter-cluster edges e_1 and e_2 incident to μ and ν. If both the regions delimited by e_1, e_2, $B(\mu)$, and $B(\nu)$ contain vertices in $G \setminus (\mu \cup \nu)$, then all the clusters in one of the regions are only connected to μ and all the clusters in the other region are only connected to ν, due to the conditions of the lemma. Hence, (μ, ν) is a trivial block of G_A, a contradiction. ◀

We exploit Theorem 13 to construct a neighbor-clusters tree Y°_μ of each cluster $\mu \in \mathcal{T}$ such that any c-planar drawing with pipes of $\mathcal{C}(G,\mathcal{T})$ is consistent with Y°_μ. Tree Y°_μ is rooted at a vertex ω_μ. There exists a child ν of ω_μ for each cluster ν adjacent to μ, having a leaf $y_\mu(e)$ for each inter-cluster edge e incident to μ and to ν. We call Y°_μ the *pipe-neighbor-clusters tree* of μ. Theorem 13 and the construction of Y°_μ, for each cluster $\mu \in \mathcal{T}$, imply the following.

▶ **Corollary 14.** *Let $\mathcal{C}(G,\mathcal{T})$ be a c-graph whose clusters-adjacency graph has no trivial blocks. Then, $\mathcal{C}(G,\mathcal{T})$ admits a c-planar drawing with pipes if and only if $\mathcal{C}(G,\mathcal{T})$ admits a c-planar drawing Γ in which, for each $\mu \in \mathcal{T}$, drawing Γ is consistent with Y°_μ.*

By Corollary 14 we can reduce C-Planarity with Pipes for a c-graph whose clusters-adjacency graph G_A has no trivial blocks to Inclusion-Constrained C-Planarity, where the role played by the neighbor-clusters trees is taken by the pipe-neighbor-clusters trees. In the full paper [1] we explain how to overcome the requirement that G_A has no trivial block.

▶ **Lemma 15.** *Let $\mathcal{C}(G,\mathcal{T})$ be an instance of* C-Planarity with Pipes *in which G_A contains trivial blocks. It is possible to construct in linear time an equivalent instance $\mathcal{C}^*(G^*,\mathcal{T}^*)$ of* C-Planarity with Pipes *in which G^*_A has no trivial block. Further, $K_* = K$ and $c_* = c$, where K (K_*) is the maximum number of multi-edge components in a cluster of $\mathcal{C}(G,\mathcal{T})$ (of $\mathcal{C}^*(G^*,\mathcal{T}^*)$) and c (c_*) is the number of clusters of $\mathcal{C}(G,\mathcal{T})$ (of $\mathcal{C}^*(G^*,\mathcal{T}^*)$) with at least two multi-edge components.*

5.3 An FPT Algorithm for C-Planarity with Pipes

In the following we prove the main result of the section.

▶ **Theorem 16.** C-Planarity with Pipes *can be tested in $O(K^{c(K-2)}) \cdot O(n^2)$ time, where K is the maximum number of multi-edge components in a cluster and c is the number of clusters with at least two multi-edge components.*

Proof Sketch. Let $\mathcal{C}(G,\mathcal{T})$ be a c-graph, which can be assumed to have no trivial block by Lemma 15. Construct the set $\mathcal{Y}$ containing the unique pipe-neighbor-clusters tree Y°_μ of each cluster $\mu \in \mathcal{T}$. Then, construct all the possible sets $\mathcal{X}$ of components trees, over all clusters in $\mathcal{T}$. For each pair $\langle \mathcal{X}, \mathcal{Y} \rangle$, apply Theorem 12 to test whether $\langle \mathcal{C}(G, \mathcal{T}), \mathcal{X}, \mathcal{Y} \rangle$ is a positive instance of Inclusion-Constrained C-Planarity. By Theorem 13 and Corollary 14, c-graph $\mathcal{C}(G, \mathcal{T})$ is a positive instance if and only if at least one of such tests succeeds. ◀

We observe two notable corollaries of Theorem 16 (for the second, see Lemma 10).

▶ **Corollary 17.** Strip Planarity *can be tested in* $O(K^{s(K-2)}) \cdot O(n^2)$ *time, where* K *is the maximum number of multi-edge components in a strip and* s *is the number of strips containing at least two multi-edge components.*

▶ **Corollary 18.** C-Planarity with Embedded Pipes *can be tested in* $K^{c(K-2)} \cdot O(n^2)$ *time, where* K *is the maximum number of multi-edge components in a cluster and* c *is the number of clusters with at least two multi-edge components.*

6 Conclusions and Open Problems

In this paper we studied the problem of constructing c-planar drawings with pipes of flat c-graphs. We presented algorithms to test the existence of such drawings when the number of certain components is small, in different scenarios, namely when the clusters-adjacency graph is a path (Strip Planarity), when it has a fixed embedding (C-Planarity with Embedded Pipes), and when it has no restrictions (C-Planarity with Pipes).

Several questions are left open. We find particularly interesting to determine whether there exist combinatorial properties of the nesting of the components allowing us to reduce the number of possible components trees, analogous to the ones we could prove for the pipe-neighbor-clusters trees. We remark that the introduction of the components trees makes the running time of our FPT algorithms independent of the size of each component.

References

1 Patrizio Angelini and Giordano Da Lozzo. Clustered Planarity with Pipes. *ArXiv e-prints*, 2016. arXiv:1609.09679.

2 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Strip planarity testing for embedded planar graphs. *Algorithmica*, 2016. To appear.

3 Paola Bertolazzi, Giuseppe Di Battista, Carlo Mannino, and Roberto Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998. doi:10.1137/S0097539794279626.

4 Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous embedding of planar graphs. In *Handbook of Graph Drawing and Visualization*. CRC Press, 2013.

5 Thomas Bläsius and Ignaz Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. *Theor. Comput. Sci.*, 609:306–315, 2016. doi:10.1016/j.tcs.2015.10.011.

6 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Trans. Algorithms*, 12(2):16, 2016. doi:10.1145/2738054.

7 K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. of Computer and System Sciences*, 13(3):335–379, 1976.

8 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *SODA'15*, pages 1655–1670. SIAM, 2015. doi:10.1137/1.9781611973730.

9 Pier Francesco Cortese, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Maurizio Pizzonia. C-planarity of c-connected clustered graphs. *J. Graph Algorithms Appl.*, 12(2):225–262, 2008.

10 Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. On embedding a cycle in a plane graph. *Discrete Mathematics*, 309(7):1856–1869, 2009. doi:10.1016/j.disc.2007.12.090.

11 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

12 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for clustered graphs. In *Algorithms – ESA'95, Third Annual European Symposium, Proc.*, pages 213–226, 1995. doi:10.1007/3-540-60313-1_145.

13 Radoslav Fulek. Toward the Hanani–Tutte Theorem for Clustered Graphs. *ArXiv e-prints*, 2014. arXiv:1410.3022.

14 Vít Jelínek, Eva Jelínková, Jan Kratochvíl, and Bernard Lidický. Clustered planarity: Embedded clustered graphs with two-component clusters. In *GD'08*, pages 121–132, 2008. doi:10.1007/978-3-642-00219-9_13.

15 Marcus Schaefer. Toward a theory of planarity: Hanani-tutte and planarity variants. *J. of Graph Algorithms and Applications*, 17(4):367–440, 2013.

L_1 Geodesic Farthest Neighbors in a Simple Polygon and Related Problems[*]

Sang Won Bae

Department of Computer Science, Kyonggi University, Suwon, South Korea
swbae@kgu.ac.kr

Abstract

In this paper, we investigate the L_1 geodesic farthest neighbors in a simple polygon P, and address several fundamental problems related to farthest neighbors. Given a subset $S \subseteq P$, an L_1 geodesic farthest neighbor of $p \in P$ from S is one that maximizes the length of L_1 shortest path from p in P. Our list of problems include: computing the diameter, radius, center, farthest-neighbor Voronoi diagram, and two-center of S under the L_1 geodesic distance. We show that all these problems can be solved in linear or near-linear time based on our new observations on farthest neighbors and extreme points. Among them, the key observation shows that there are at most four extreme points of any compact subset $S \subseteq P$ with respect to the L_1 geodesic distance after removing redundancy.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases simple polygon, L_1 geodesic distance, farthest neighbor, farthest-neighbor Voronoi diagram, k-center

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.14

1 Introduction

The geometry of points in a simple polygon P has been one of the most attractive research subjects in computational geometry since the 1980s. As a metric space, P is often associated with a distance function d induced by shortest paths that stay inside P. Indeed, there are several ways to define a shortest path between two points in P, depending on which underlying metric is adopted to determine the length of a segment in P. Most common are the Euclidean and the L_1 metrics that define the Euclidean and the L_1 shortest paths, respectively, in P. The length of a shortest path between two points $p, q \in P$ is called the (Euclidean or L_1) geodesic distance $d(p, q)$.

In this paper, we are interested in fundamental problems related to geodesic farthest neighbors in P. Given a set S of points in P, a farthest neighbor of $p \in P$ from S is one that maximizes the geodesic distance $d(p, q)$ from p to every $q \in S$. Specifically, our list of problems include those of computing the following:

- The *farthest-neighbor Voronoi diagram* of S.
- The *diameter* of S: $\mathrm{diam}(S) := \max_{q\in S} \max_{q'\in S} d(q, q')$.
- The *radius* of S: $\mathrm{rad}(S) := \min_{p\in P} \max_{q\in S} d(p, q)$.
- A *center* of S: a point $c \in P$ such that $\max_{q\in S} d(c, q) = \mathrm{rad}(S)$.
- A *two-center* of S: a pair of points $c_1, c_2 \in P$ that minimizes $\max_{q\in S} \min\{d(c_1, q), d(c_2, q)\}$.

[*] This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057220).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 14; pp. 14:1–14:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the Euclidean case, where d denotes the Euclidean geodesic distance, these problems have been intensively studied. Aronov et al. [2] presented an $O((n+m)\log(n+m))$-time algorithm that computes the farthest-neighbor Voronoi diagram of S if S consists of m points and P is an n-gon. Very recently, Oh et al. [11] showed that the diagram can be computed faster in $O(n \log\log n + m\log(n+m))$ time, or in $O(n\log\log n)$ time when S is the set of vertices of P. Note that computing the diameter, radius, and center of S is reduced from the farthest-neighbor Voronoi diagram in linear time. On the other hand, in a special case where $S = P$, it is known that we can compute them in linear $O(n)$ time [9, 1]. The problem of computing a two-center of S under the Euclidean geodesic distance was recently addressed by Oh et al. [12] and Oh et al. [10], resulting in two algorithms that run in $O(n^2 \log^2 n)$ time when $S = P$ and in $O(m^2(m+n)\log^3(m+n))$ time when S is a set of m points in P.

The problems in the L_1 geodesic distance have attained less interest compared to those in the Euclidean case. This is probably because most of results for the Euclidean counterpart automatically hold for the L_1 geodesic distance. Note that the Euclidean shortest paths in P are also L_1 shortest paths, and the algorithm of Aronov et al. [2] can be implemented for computing the L_1 geodesic farthest-neighbor Voronoi diagram. However, it is not clear whether the approach by Oh et al. [11] can be extended to compute the L_1 diagram. Bae et al. [3] exhibited some geometric observations on the L_1 geodesic distance that are different from the Euclidean one, and exploited them to devise linear-time algorithms that compute the diameter, radius, and center of a simple polygon P, i.e., the special case where $S = P$. Prior to this work, no algorithm for the two-center of S in the L_1 geodesic distance was known in the literature.

In this paper, we reveal that farthest neighbors in the L_1 geodesic distance behave quite different from – indeed much nicer than – gthose in the Euclidean geodesic distance. Based on our new observations, we show that all the problems listed above in the L_1 geodesic distance can be computed in linear or near-linear time:

- $O(n + m\log n)$ time when S is a set of m points in P, or
- $O(n)$ time when either $S = P$ or S equals the set of vertices of P.

It is worth noting that our algorithms runs in time linear to each of n and m, while the $O(m\log n)$ term was unavoidable for evaluation of the geodesic distance $d(p, q)$. Note that, in particular, our algorithms for the farthest-neighbor Voronoi diagram and the two-center are faster than the currently best algorithms for those in the Euclidean case: roughly by a factor $\log\log n$ for the farthest-neighbor Voronoi diagram [11], and by a factor of n or of m^2 for the two-center problem [12, 10]. All these algorithmic results are based on a key observation that for any compact subset $S \subseteq P$ of P, there are at most four extreme points of S in general. Note that in the Euclidean case, there can be linearly many extreme points.

This phenomenon can be understood as an extension of the relation between the L_1 plane and the Euclidean plane. In the plane associated with the L_1 metric, there are at most four extreme points of S in the four directions corresponding to the four segments of the L_1 metric balls, while in the plane associated with the Euclidean metric, every point of S lying on the boundary of its convex hull is considered to be extreme. An immediate implication is that the farthest-neighbor Voronoi diagram in the L_1 metric consists of at most four nonempty regions and thus has $O(1)$ complexity, while this is not the case for the Euclidean metric. Similarly, an L_1 (or rectilinear) two-center of m points in the plane can be computed in $O(m)$ time [5], while the best known algorithm that computes a Euclidean two-center in the plane runs in $O(m\log^2 m(\log\log m)^2)$ deterministic time [4]. Our results thus provide a series of analogies on farthest neighbors in the L_1 plane into those in the metric space (P, d), where P is a simple polygon and d is the L_1 geodesic distance in P.

2 Preliminaries

For any subset $A \subset \mathbb{R}^2$, we denote by ∂A and $\mathrm{int}A$ the boundary and the interior of A, respectively. For $p, q \in \mathbb{R}^2$, denote by $\overline{pq}$ the line segment with endpoints p and q. For any path π in $\mathbb{R}^2$, let $|\pi|$ be the length of π under the L_1 metric, or simply the L_1 *length.* Note that $|\overline{pq}|$ equals the L_1 distance between p and q.

The following is a basic observation on the L_1 length of paths in $\mathbb{R}^2$. A path is called *monotone* if any vertical or horizontal line intersects it in at most one connected component.

▶ **Lemma 1.** *For any path π between $p, q \in \mathbb{R}^2$, $|\pi| = |\overline{pq}|$ if and only if π is monotone.*

Let P be a simple polygon with n vertices. We regard P as a compact set in $\mathbb{R}^2$, so its boundary ∂P is contained in P. An L_1 *shortest path* between p and q is a path joining p and q that lies in P and minimizes its L_1 length. The L_1 *geodesic distance* $d(p, q)$ is the L_1 length of an L_1 shortest path between p and q. For any $p, q \in P$, let $\Pi(p, q)$ be the set of all L_1 shortest paths from p to q.

Analogously, a path lying in P minimizing its *Euclidean* length is called the *Euclidean shortest path.* It is well known that there is always a unique Euclidean shortest path between any two points in a simple polygon [7]. We let $\pi_2(p, q)$ be the unique Euclidean shortest path from $p \in P$ to $q \in P$. The following states a crucial relation between Euclidean and L_1 shortest paths in a simple polygon.

▶ **Lemma 2** (Hershberger and Snoeyink [8])**.** *For any two points $p, q \in P$, the Euclidean shortest path $\pi_2(p, q)$ is also an L_1 shortest path between p and q. That is, $\pi_2(p, q) \in \Pi(p, q)$.*

Lemma 2 enables us to exploit several structures for Euclidean shortest paths such as Guibas et al. [7] and Guibas and Hershberger [6].

Another important concept regarding the shortest paths in P is the relative convexity. A subset $A \subset P$ is called *relative convex* if $\pi_2(p, q) \subset A$ for any $p, q \in A$. For any subset $A \subset P$, the *relative convex hull* $\mathrm{rconv}(A)$ of A is the smallest relative convex set including A. If A is the set of m points in P, then its relative convex hull forms a weakly simple polygon in P with $O(m + n)$ vertices. Touissant [13] presented an $O((n + m) \log(n + m))$-time algorithm that computes $\mathrm{rconv}(A)$, and Guibas and Hershberger [6] improved it to $O(n+m \log(n+m))$.

Throughout the paper, unless otherwise stated, a shortest path and the geodesic distance always refer to an L_1 shortest path and the L_1 geodesic distance d.

3 Properties of L_1 Shortest Paths

In this section, we observe several useful properties of L_1 shortest paths in P.

We define a *chord* of P to be a maximal segment contained in P. For any $z \in P$, let h_z^- and h_z^+ be the left and right endpoints, respectively, of the horizontal chord through z, while v_z^- and v_z^+ denote the lower and upper endpoints, respectively, of the vertical chord through z. Note that the horizontal or vertical chord may intersect the boundary ∂P of P in several connected components by definition. We also consider the four segments $\overline{zh_z^-}$, $\overline{zh_z^+}$, $\overline{zv_z^-}$ and $\overline{zv_z^+}$, called the *leftward, rightward, downward,* and *upward half-chords* from z, respectively.

Let $z \in P$ be fixed and $p \in P$ be any point. We say that $\pi \in \Pi(p, z)$ *chooses* a half-chord from z if π intersects it at a point other than z. Then, Lemma 1 implies that every $\pi \in \Pi(p, z)$ chooses at most one half-chord from z or none of the four. We then observe the following.

▶ **Lemma 3.** *For any $p, z \in P$, there are no two shortest paths $\pi, \pi' \in \Pi(p, z)$ such that π chooses a half-chord from z and π' chooses its opposite half-chord from z.*

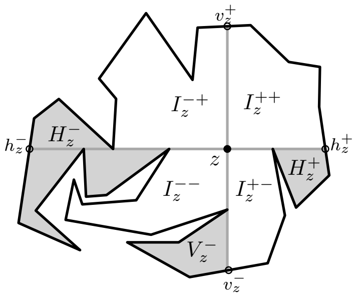


Figure 1 Partition around z.

This implies that any $p \in P$ avoids at least two half-chords that are not opposite by shortest paths to z. We thus consider the four regions of P according to the pair of excluded half-chords. More precisely, for any $\sigma_1, \sigma_2 \in \{+, -\}$, let $P_z^{\sigma_1\sigma_2} \subset P$ be the set of points $p \in P$ such that no shortest path $\pi \in \Pi(p, z)$ chooses $\overline{zh_z^{\overline{\sigma_1}}}$ or $\overline{zv_z^{\overline{\sigma_2}}}$, where $\overline{+} = -$ and $\overline{-} = +$. Lemma 3 guarantees that $P = P_z^{--} \cup P_z^{-+} \cup P_z^{+-} \cup P_z^{++}$ for any $z \in P$, while these four regions are not disjoint. Also, note that $P_z^{--} \cap P_z^{++} = \{z\}$ and $P_z^{-+} \cap P_z^{+-} = \{z\}$.

In order to gain a comprehensive understanding on the four regions $P_z^{\sigma_1\sigma_2}$, we consider the following eight subsets of P around z: Define $H_z^{\sigma_1} := P_z^{\sigma_1 -} \cap P_z^{\sigma_1 +}$, $V_z^{\sigma_2} := P_z^{-\sigma_2} \cap P_z^{+\sigma_2}$, and $I_z^{\sigma_1\sigma_2} := P_z^{\sigma_1\sigma_2} \setminus (H_z^{\sigma_1} \cup V_z^{\sigma_2})$. Observe that H_z^-, for example, is the set of points $p \in P$ such that no shortest path in $\Pi(p, z)$ chooses the downward, upward, or rightward half-chord from z, and I_z^{--} is the set of points $p \in P$ that admit two shortest paths $\pi, \pi' \in \Pi(p, z)$ such that π chooses the leftward half-chord and π' chooses the downward half-chord from z. See Figure 1 for an illustration. Note that these eight subsets $H_z^{\sigma_1}$, $V_z^{\sigma_2}$, and $I_z^{\sigma_1\sigma_2}$ form a partition of P around z. In most cases where the horizontal and vertical chords through z intersects ∂P only at their endpoints, we have $H_z^{\sigma_1} = \overline{zh_z^{\sigma_1}}$ and $V_z^{\sigma_2} = \overline{zv_z^{\sigma_2}}$. However, this is not always the case.

To be more precise, consider the complement $\mathcal{C}_z := P \setminus (\overline{h_z^- h_z^+} \cup \overline{v_z^- v_z^+})$, which in general consists of several connected components. Such a component $C \subseteq \mathcal{C}_z$ is said to be *adjacent* to a half-chord from z if its boundary ∂C intersects the half-chord at a point other than z. Note that any component of $\mathcal{C}_z$ is adjacent to at least one and at most two half-chords from z. The following describes how H_z^-, H_z^+, V_z^-, and V_z^+ are formed.

▶ **Lemma 4.** *Let $z \in P$ and $\sigma_1, \sigma_2 \in \{+, -\}$. Then, $H_z^{\sigma_1}$ is equal to the union of $\overline{zh_z^{\sigma_1}}$ and the components of $\mathcal{C}_z$ that are adjacent to $\overline{zh_z^{\sigma_1}}$ but to none of the others. Analogously, $V_z^{\sigma_2}$ is equal to the union of $\overline{zv_z^{\sigma_2}}$ and the components of $\mathcal{C}_z$ that are adjacent to $\overline{zv_z^{\sigma_2}}$ only.*

Thus, any component C of $\mathcal{C}_z$ that is adjacent to exactly one half-chord is included into the corresponding subset $H_z^{\sigma_1}$ or $V_z^{\sigma_2}$ for some $\sigma_1, \sigma_2 \in \{+, -\}$. On the other hand, if a component C of $\mathcal{C}_z$ is adjacent to two half-chords from z, then the boundary of C must contain z. Thus, there are at most four such components of $\mathcal{C}_z$, and each of them forms $I_z^{\sigma_1\sigma_2}$ for some $\sigma_1, \sigma_2 \in \{+, -\}$. Lemma 4 and the above discussion imply the following corollary.

▶ **Corollary 5.** *Suppose that $H_z^{\sigma_1} \setminus \overline{zh_z^{\sigma_1}} \neq \emptyset$ for $\sigma_1 \in \{+, -\}$. Then, either $z \in \{v_z^-, v_z^+\}$ or there exists a vertex u of P lying on $\overline{zh_z^-}$ such that for all $p \in H_z^{\sigma_1} \setminus \overline{zh_z^{\sigma_1}}$ any shortest path $\pi \in \Pi(p, z)$ passes through u. An analogous claim also holds for the set $V_z^{\sigma_2}$ with $\sigma_2 \in \{+, -\}$.*

We then prove the following properties of L_1 shortest paths in terms of the partition around a point $z \in P$.

▶ **Lemma 6.** *Let $p, q, z \in P$. If $p, q \in A$, then every $\pi \in \Pi(p, q)$ is contained in A, where A is equal to one of the following sets: $H_z^{\sigma_1}$, $V_z^{\sigma_2}$, $I_z^{\sigma_1\sigma_2}$, $I_z^{\sigma_1\sigma_2} \cup \partial I_z^{\sigma_1\sigma_2}$, $I_z^{\sigma_1\sigma_2} \cup H_z^{\sigma_1}$, $I_z^{\sigma_1\sigma_2} \cup V_z^{\sigma_2}$, and $P_z^{\sigma_1\sigma_2}$ for any $\sigma_1, \sigma_2 \in \{+, -\}$.*

▶ **Lemma 7.** *Let $p, q, z \in P$ be any three points. Then, $d(p, q) = d(p, z) + d(z, q)$ if and only if $p \in P_z^{\sigma_1\sigma_2}$ and $q \in P_z^{\overline{\sigma_1}\,\overline{\sigma_2}}$ for some $\sigma_1, \sigma_2 \in \{+, -\}$.*

4 L_1 Geodesic Farthest Neighbors and Extreme Points

Let $S \subseteq P$ be a nonempty, compact subset of P. We are interested in farthest neighbors of each $p \in P$ from S. For each $p \in P$, let $\Phi_S(p) := \max_{q \in S} d(p, q)$. This is well defined since S is a compact set. We call such a $q \in S$ with $d(p, q) = \Phi_S(p)$ an L_1 *geodesic farthest neighbor of* p *from* S, or shortly a *farthest neighbor of* p when there is no confusion. There can be several farthest neighbors of $p \in P$ from S. We denote by $F_S(p)$ the set of all farthest neighbors of p from S. In order pick a representative among them, we impose a total order $\prec$ on S, such as the lexicographical order. We then define $f_S(p) \in F_S(p)$ to be the least with respect to the order $\prec$ among the farthest neighbors of p in $F_S(p)$. We call $q \in S$ an *(L_1 geodesic) extreme point* of S if $q = f_S(p)$ for some $p \in P$.

There are two fundamental quantities defined by farthest neighbors in P: the *(L_1 geodesic) diameter* $\operatorname{diam}(S) := \max_{q \in S} \Phi_S(q)$ and the *(L_1 geodesic) radius* $\operatorname{rad}(S) := \min_{p \in P} \Phi_S(p)$ of S. The diameter and radius of S are well defined since P and S are compact sets. A pair of points $q, q' \in S$ is called *diametral* if $d(q, q') = \operatorname{diam}(S)$, while a point $c \in P$ is called an *(L_1 geodesic) center* of S if $\Phi_S(c) = \operatorname{rad}(S)$. Let $\operatorname{cen}(S)$ be the set of all centers $c \in P$ of S.

In this section, we fully reveal the behavior of the L_1 farthest neighbors and extreme points of any compact set S in P, and finally prove the following theorem.

▶ **Theorem 8.** *In a simple polygon P, there are at most four extreme points of any compact subset $S \subseteq P$ with respect to the L_1 geodesic distance.*

In order to prove Theorem 8, we consider farthest neighbors constrained in regions. For $\sigma_1, \sigma_2 \in \{+, -\}$, define $f_S^{\sigma_1\sigma_2}(p)$ to be the farthest neighbor of p from $S \cap P_p^{\sigma_1\sigma_2}$ that is the least with respect to $\prec$. In the case where $S \cap P_p^{\sigma_1\sigma_2} = \emptyset$, $f_S^{\sigma_1\sigma_2}(p)$ is undefined. Then observe that $f_S(p)$ is the farthest one that is the least with respect to $\prec$ among the four candidates $f_S^{++}(p)$, $f_S^{-+}(p)$, $f_S^{--}(p)$, and $f_S^{+-}(p)$.

We first gather some useful properties of farthest neighbors.

▶ **Lemma 9.** *Given any $p \in P$, suppose that $f_S^{\sigma_1\sigma_2}(p) \in F_S(p)$ for $\sigma_1, \sigma_2 \in \{+, -\}$. Then, $f_S^{\sigma_1\sigma_2}(p) \in F_S(p')$ for any $p' \in P_p^{\overline{\sigma_1}\,\overline{\sigma_2}}$, and $f_S(p') = f_S^{\sigma_1\sigma_2}(p)$ for any $p' \in I_p^{\overline{\sigma_1}\,\overline{\sigma_2}}$. Moreover, if $f_S(p) = f_S^{\sigma_1\sigma_2}(p)$, then $f_S(p') = f_S^{\sigma_1\sigma_2}(p)$ for any $p' \in P_p^{\overline{\sigma_1}\,\overline{\sigma_2}}$.*

▶ **Lemma 10.** *For any $p \in P$, let $z \in \pi$ be a point on a shortest path $\pi \in \Pi(p, f_S(p))$ Then, for any $\sigma_1, \sigma_2 \in \{+, -\}$ with $p \in P_z^{\sigma_1\sigma_2}$ and $f_S(p) \in P_z^{\overline{\sigma_1}\,\overline{\sigma_2}}$, it holds that $f_S(p) = f_S^{\overline{\sigma_1}\,\overline{\sigma_2}}(z)$.*

Note that such $\sigma_1, \sigma_2 \in \{+, -\}$ with $p \in P_z^{\sigma_1\sigma_2}$ and $f_S(p) \in P_z^{\overline{\sigma_1}\,\overline{\sigma_2}}$ in Lemma 10 always exist by Lemma 7 since z is a point on a shortest path from p to $f_S(p)$, so $d(p, f_S(p)) = d(p, z) + d(z, f_S(p))$.

4.1 Proof of Theorem 8

Now, we give a proof of Theorem 8. The case where S consists of at most one point is trivial, so we assume that S consists of more than one point. For a center $c \in \operatorname{cen}(S)$ of S, we consider the set $F_S(c)$ of its farthest neighbors. Since c is a center and S consists of at least two points, we have $|F_S(c)| \geq 2$ and $d(c, \chi) = \Phi_S(c) = \operatorname{rad}(S)$ for any $\chi \in F_S(c)$.

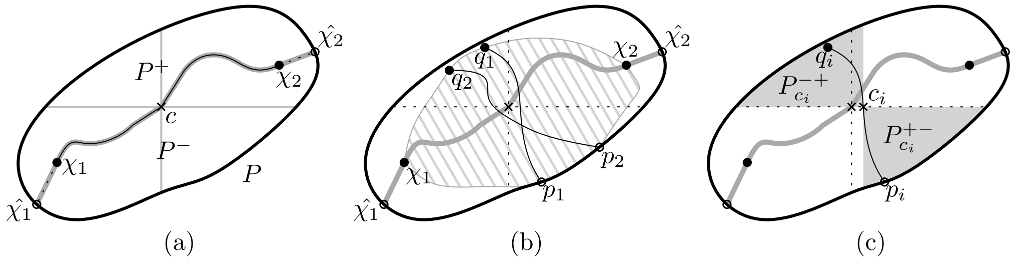


Figure 2 (a) The path $\hat{\pi}$ through $c \in \text{cen}(S)$ partitions P into P^+ and P^-. (b) Illustration to Claims 13 and 14. Shaded region represents $\text{rconv}(S \cup \{p_1, p_2\})$. (c) Illustration to Claim 15.

▶ **Lemma 11.** *For any $c \in \text{cen}(S)$, there exist $\chi_1, \chi_2 \in F_S(c)$ satisfying the following: (i) $d(\chi_1, \chi_2) = d(\chi_1, c) + d(c, \chi_2)$, and (ii) $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$.*

From now on, we fix any two farthest neighbors $\chi_1, \chi_2 \in F_S(c)$ of c with the property of Lemma 11. Note that χ_1 and χ_2 are extreme points of S. Since $d(\chi_1, \chi_2) = d(\chi_1, c) + d(c, \chi_2)$, we have $\chi_1 \in P_c^{\sigma_1 \sigma_2}$ and $\chi_2 \in P_c^{\overline{\sigma_1}\,\overline{\sigma_2}}$ for some $\sigma_1, \sigma_2 \in \{+, -\}$ by Lemma 7. Without loss of generality, we assume that $\sigma_1 = \sigma_2 = -$, so $\chi_1 \in P_c^{--}$ and $\chi_2 \in P_c^{++}$.

Let $\pi := \pi_2(\chi_1, c) \cup \pi_2(c, \chi_2)$ be a path from χ_1 to χ_2. Since $d(\chi_1, \chi_2) = d(\chi_1, c) + d(c, \chi_2)$, π is a shortest path from χ_1 to χ_2, that is, $\pi \in \Pi(\chi_1, \chi_2)$. Then, by Lemma 10, we have $\chi_1 = f_S^{--}(c)$ and $\chi_2 = f_S^{++}(c)$, as $c \in \pi$. This further implies that $f_S(p) = \chi_2$ for any $p \in I_c^{--}$, and $f_S(p) = \chi_1$ for any $p \in I_c^{++}$ by Lemma 9 since $\chi_1, \chi_2 \in F_S(c)$.

We will need the following lemma, which rephrases the Ordering Lemma by Aronov et al. [2]. Note that every extreme point of S appears on the boundary of the relative convex hull $\text{rconv}(S)$ of S.

▶ **Lemma 12** (Aronov et al. [2]). *Suppose that there are three distinct extreme points χ_1, χ_2, χ_3 of S in the counter-clockwise order along $\partial\text{rconv}(S)$. Let $p_i \in \partial P$ be a point on the boundary of P such that $f_S(p_i) = \chi_i$ for each $i \in \{1, 2, 3\}$. Then, p_1, p_2, p_3 appear in this order along ∂P in the counter-clockwise direction.*

Consider the extension of the last segment of $\pi_2(c, \chi_i)$ for each $i \in \{1, 2\}$ until it hits the first boundary point $\hat{\chi}_i \in \partial P$. Let $\hat{\pi}$ be the shortest path from $\hat{\chi}_1$ to $\hat{\chi}_2$ obtained by these extensions from π; that is, $\hat{\pi} = \pi_2(\hat{\chi}_1, c) \cup \pi_2(c, \hat{\chi}_2)$. Note that $f_S(\hat{\chi}_1) = \chi_2$ and $f_S(\hat{\chi}_2) = \chi_1$ by Lemma 9. The path $\hat{\pi}$ partitions P into two parts P^- and P^+, where ∂P^- consists of $\hat{\pi}$ and the chain along ∂P from $\hat{\chi}_1$ to $\hat{\chi}_2$ in the counter-clockwise direction, and ∂P^+ consists of $\hat{\pi}$ and the chain along ∂P from $\hat{\chi}_2$ to $\hat{\chi}_1$ in the counter-clockwise direction. See Figure 2(a) for an illustration. In the following, we show that there are at most one more extreme point of S, other than χ_1 and χ_2, in each of P^- and P^+. Recall that an extreme point of S is $q \in S$ such that $q = f_S(p)$ for some $p \in P$.

Suppose to the contrary that there are two extreme points q_1, q_2 of S such that $q_1, q_2 \in P^+$ and the four points χ_1, χ_2, q_1, q_2 are all distinct. Then there exist two boundary points $p_1, p_2 \in \partial P$ such that $f_S(p_1) = q_1$ and $f_S(p_2) = q_2$. Such boundary points p_1, p_2 are guaranteed to exist by Lemma 9. Our proof will be done by a contradiction based on the following four claims.

▶ **Claim 13.** *Both p_1 and p_2 lie in I_c^{+-}.*

In the following, we assume that the four points χ_2, q_1, q_2, χ_1 appear in this order along $\partial\text{rconv}(S)$ in the counter-clockwise direction. Then, Lemma 12 implies the following.

▶ **Claim 14.** *The four points p_1, p_2, q_1, q_2 appear in this order along* $\partial\mathrm{rconv}(S \cup \{p_1, p_2\})$*.*

See Figure 2(b) for an illustration to the above two claims. Note that Claim 13 implies that $p_1, p_2 \in P^-$ as $I_c^{+-} \subset P^-$, so any shortest path from p_i to q_i crosses $\hat{\pi}$. On the other hand, Claim 14 implies that $\pi_2(p_1, q_1)$ and $\pi_2(p_2, q_2)$ cross each other.

Let $\beta := \partial I_c^{+-} \setminus \partial P$. Note that β is a subset of the union of the rightward half-chord $\overline{ch_c^+}$ and the downward half-chord $\overline{cv_c^-}$ from c. By Claim 13, the paths $\pi_2(p_1, q_1)$ and $\pi_2(p_2, q_2)$ must cross over β as $p_1, p_2 \in I_c^{+-}$ and $q_1, q_2 \in P^+$. For each $i \in \{1, 2\}$, let c_i be the first intersection point of $\pi_2(p_i, q_i) \cap \beta$ when walking from p_i to q_i along $\pi_2(p_i, q_i)$. We then observe the following.

▶ **Claim 15.** *For $i \in \{1, 2\}$, we have $p_i \in P_{c_i}^{+-}$ and $q_i \in P_{c_i}^{-+}$*

See Figure 2(c) for an illustration to Claim 15. Our last claim to prove Theorem 8 is the following.

▶ **Claim 16.** *There exists $c' \in P$ such that $p_1, p_2 \in P_{c'}^{+-}$ and $q_1, q_2 \in P_{c'}^{-+}$.*

Now, we are ready to achieve the final contradiction. Let $c' \in P$ be such a point described in Claim 16. Then, we have $d(p_i, q_i) = d(p_i, c') + d(c', q_i)$ for $i \in \{1, 2\}$ by Lemma 7. Since $f_S(p_i) = q_i$ and $q_i \in P_{c'}^{-+}$, we have $f_S^{-+}(c') = q_i$ for each $i \in \{1, 2\}$ by Lemma 10. This leads to a contradiction since $f_S^{-+}(c')$ is uniquely determined by definition.

Consequently, there are no two disticnt extreme points $q_1, q_2 \in S$ of S such that $q_1, q_2 \in P^+$, implying that there is at most one extreme point of S in P^+ or P^-. This completes the proof of Theorem 8. ◀

5 L_1 Geodesic Center

In this section, we investigate the set $\mathrm{cen}(S)$ of L_1 geodesic centers of S in P. Recall that an L_1 geodesic center c of S minimizes $\Phi_S(c')$ over all $c' \in P$, so $\Phi_S(c) = \mathrm{rad}(S)$. Another remarkable consequence from the discussions in the previous section is the following.

▶ **Lemma 17.** *For any nonempty compact subset $S \subseteq P$, there is a diametral pair (χ_1, χ_2) of S such that $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$.*

The above lemma and its proof indeed show the following.

▶ **Corollary 18.** *For any nonempty compact subset $S \subseteq P$, it holds that* $\mathrm{rad}(S) = \mathrm{diam}(S)/2$.

Bae et al. [3] considered a special case where $S = P$, and proved that $\mathrm{diam}(P) = 2\mathrm{rad}(P)$ by using a Helly-type theorem: any family of L_1 geodesic balls has Helly number at most two. It is worth noting that we generalize it to any compact subset S of P with a relatively direct argument in terms of extreme points of S.

For $p \in P$ and $r \in \mathbb{R}$, let $\mathrm{B}_p(r) := \{x \in P \mid d(x, p) \leq r\}$ be the L_1 *geodesic ball* at p with radius r. Bae et al. [3] also exhibited several basic properties of the L_1 geodesic balls; among them is that $\mathrm{B}_p(r)$ is relative convex for any $p \in P$ and $r \in \mathbb{R}$.

We fully characterize the set $\mathrm{cen}(S)$ of all centers of S by using those known results.

▶ **Lemma 19.** *For any nonempty compact subset $S \subseteq P$,* $\mathrm{cen}(S)$ *is equal to the intersection of at most four geodesic balls $\bigcap_{\chi \in X} \mathrm{B}_\chi(\mathrm{rad}(S))$, where X is the set of extreme points of S.*

In the following, we assume that S consists of at least two points. Then, the set X of extreme points of S consists of at least two and at most four points. Let (χ_1, χ_2) be a diametral pair such that $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$. Such a diametral pair exists by Lemma 17. Since $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$, both χ_1 and χ_2 are extreme points of S, that is, $\chi_1, \chi_2 \in X$. By Corollary 18, we know that $\mathrm{diam}(S) = 2\mathrm{rad}(S)$. Thus, $\mathrm{B}_{\chi_1}(\mathrm{rad}(S))$ and $\mathrm{B}_{\chi_2}(\mathrm{rad}(S))$ intersect only in their boundaries. Let $B := \mathrm{B}_{\chi_1}(\mathrm{rad}(S)) \cap \mathrm{B}_{\chi_2}(\mathrm{rad}(S))$. Since any L_1 geodesic ball is relative convex, as shown in [3], we observe that $B \cap \partial P$ is either $\emptyset$, a single point, or two points. Again by the relative convexity, B already forms a line segment of slope 1 or -1, since the boundary of any L_1 geodesic ball in the interior of P consists of line segments of slope 1 or -1. By Lemma 19, it holds that $\mathrm{cen}(S) \subseteq B$. This implies the the following.

▶ **Corollary 20.** *The set* $\mathrm{cen}(S)$ *of centers of any nonempty compact subset* $S \subseteq P$ *forms a line segment of slope* 1 *or* -1*, unless it is a point.*

Let $c_1, c_2 \in P$ be the endpoints of the segment $\mathrm{cen}(S)$. By Lemma 19, we know that $\mathrm{cen}(S) = B \cap \bigcup_{\chi \in X \setminus \{\chi_1, \chi_2\}} \mathrm{B}_\chi(\mathrm{rad}(S))$. Thus, if $|X| \geq 3$, then a third extreme $\chi_3 \in X$ with $\chi_3 \neq \chi_1, \chi_2$ determines an endpoint c_1 or c_2 of $\mathrm{cen}(S)$ as the intersection $B \cap \partial\mathrm{B}_{\chi_3}(\mathrm{rad}(S))$. More precisely, we observe the following.

▶ **Corollary 21.** *Suppose that* $X = \{\chi_1, \chi_2, \ldots, \chi_k\}$ *with* $3 \leq k \leq 4$*, and* (χ_1, χ_2) *is a diametral pair of* S *with* $f_S(\chi_1) = \chi_2$ *and* $f_S(\chi_2) = \chi_1$*. Then, for each* $3 \leq i \leq k$*,* $\partial\mathrm{B}_{\chi_1}(\mathrm{rad}(S)) \cap \partial\mathrm{B}_{\chi_2}(\mathrm{rad}(S)) \cap \partial\mathrm{B}_{\chi_i}(\mathrm{rad}(S))$ *determines an endpoint of* $\mathrm{cen}(S)$*.*

6 L_1 Geodesic Farthest-Neighbor Voronoi Diagram

We then turn our attention to the L_1 geodesic farthest-neighbor Voronoi diagram. Given a set S of sites in P, its L_1 geodesic farthest-neighbor Voronoi diagram $\mathsf{FVD}(S)$ is a partition of P into regions according to the farthest-neighbor relation between P and S. A common degenerate case of Voronoi diagrams occurs when a point $p \in P$ has four or more equidistant sites in S. There are two popular approaches in the literature to resolve such a degenerate case: assume a general position or impose a total order $\prec$ on S. We take the latter as done so far to give a precise definition of $\mathsf{FVD}(S)$.

The L_1 *geodesic farthest-neighbor Voronoi region* $FR(q, S)$ for each $q \in S$ is defined to be

$$FR(q, S) := \{p \in P \mid f_S(p) = q\}.$$

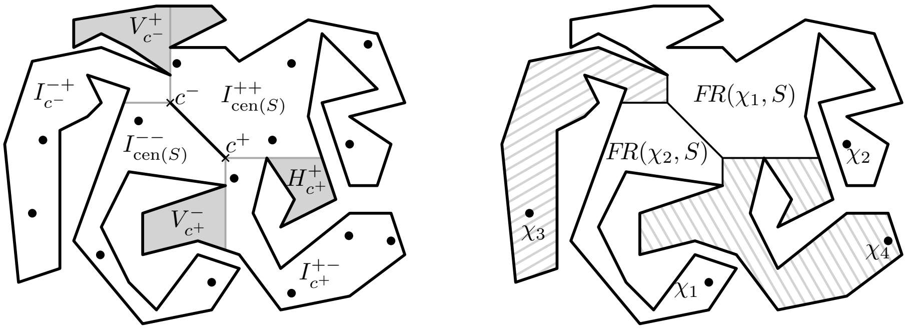
Then, the L_1 *geodesic farthest-neighbor Voronoi diagram* $\mathsf{FVD}(S)$ is defined to be

$$\mathsf{FVD}(S) := \bigcup_{q \in S} \partial FR(q, S) \setminus \partial P,$$

the union of the boundaries of each farthest-neighbor Voronoi region, except ∂P.

By definition, the Voronoi region $FR(q, S)$ for $q \in S$ is nonempty if and only if q is an extreme point of S. By Theorem 8, this implies that $\mathsf{FVD}(S)$ coincides with the diagram of at most four points in S. This enables us to define the Voronoi diagram $\mathsf{FVD}(S)$ for any nonempty compact subset $S \subseteq P$, even if S consists of an infinite number of points.

Let $X \subseteq S$ be the set of extreme points of S. Lemma 17 guarantees the existence of a diametral pair (χ_1, χ_2) of S with $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$, so $\chi_1, \chi_2 \in X$. We first observe the following property of such a diametral pair.



(a) (b)

Figure 3 (a) The partition around the center $\mathrm{cen}(S)$. Points in S are depicted as dots. (b) Illustration to $\mathsf{FVD}(S)$. Shaded regions depict $FR(\chi_3, S)$ and $FR(\chi_4, S)$, respectively. In this example, S has four extreme points $\chi_1, \chi_2, \chi_3, \chi_4$ and we assume that $\chi_3 \prec \chi_1 \prec \chi_4 \prec \chi_2$.

▶ **Lemma 22.** *Let (χ_1, χ_2) be any diametral pair of S with $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$. Then, there exist $\sigma_1, \sigma_2 \in \{+, -\}$ such that $\chi_1 = f_S^{\sigma_1\sigma_2}(c)$ and $\chi_2 = f_S^{\overline{\sigma_1}\,\overline{\sigma_2}}(c)$ for all $c \in \mathrm{cen}(S)$. Moreover, if $\mathrm{cen}(S)$ forms a line segment of positive length, then $\sigma_1 = \sigma_2$ when $\mathrm{cen}(S)$ is of slope -1, or $\sigma_1 = \overline{\sigma_2}$, otherwise.*

Note that if there are two distinct such pairs (χ_1, χ_2) and (χ_1', χ_2'), then Lemma 22 implies that the four points $\chi_1, \chi_2, \chi_1', \chi_2'$ must be all distinct. Since $|X| \leq 4$ by Theorem 8, this implies that there are at most two such pairs.

We then observe the following.

▶ **Lemma 23.** *Suppose that $\mathrm{cen}(S)$ forms a line segment of slope $(\sigma 1)$ for $\sigma \in \{+, -\}$ or a point. Let c^- and c^+ be the left and right endpoints of $\mathrm{cen}(S)$. Then, the following hold:*

- $I_c^{-\sigma} \subseteq FR(f_S^{+\overline{\sigma}}(c), S)$ *and* $I_c^{+\overline{\sigma}} \subseteq FR(f_S^{-\sigma}(c), S)$ *for any* $c \in \mathrm{cen}(S)$.
- $I_{c^-}^{-\overline{\sigma}} \subseteq FR(f_S^{+\sigma}(c^-), S)$ *and* $I_{c^+}^{+\sigma} \subseteq FR(f_S^{-\overline{\sigma}}(c^+), S)$.
- $H_{c^-}^{-} \setminus \{c^-\} \subseteq FR(f_S^{+\sigma_2}(c^-), S)$ *and* $V_{c^-}^{\overline{\sigma}} \setminus \{c^-\} \subseteq FR(f_S^{\sigma_1\sigma}(c^-), S)$ *where* $\sigma_1, \sigma_2 \in \{+, -\}$.
- $H_{c^+}^{+} \setminus \{c^+\} \subseteq FR(f_S^{-\sigma_2'}(c^+), S)$ *and* $V_{c^+}^{\sigma} \setminus \{c^+\} \subseteq FR(f_S^{\sigma_1'\overline{\sigma}}(c^+), S)$ *where* $\sigma_1', \sigma_2' \in \{+, -\}$.

Lemma 23 fully describes the farthest-neighbor Voronoi diagram $\mathsf{FVD}(S)$, according to the shape of $\mathrm{cen}(S)$. Assume without loss of generality that $\mathrm{cen}(S)$ forms a line segment of slope -1 or a point. Let $I_{\mathrm{cen}(S)}^{--} := \bigcup_{c \in \mathrm{cen}(S)} I_c^{--}$ and $I_{\mathrm{cen}(S)}^{++} := \bigcup_{c \in \mathrm{cen}(S)} I_c^{++}$. Then, observe that the eight subsets $I_{\mathrm{cen}(S)}^{--}$, $V_{c^+}^{-}$, $I_{c^+}^{+-}$, $H_{c^+}^{+}$, $I_{\mathrm{cen}(S)}^{++}$, $V_{c^-}^{+}$, $I_{c^-}^{-+}$, and $H_{c^-}^{-}$ form a partition of P around $\mathrm{cen}(S)$. See Figure 3 for an illustration. Lemma 23 describes to which Voronoi region each of these eight subsets of P belongs. Note that each of the four subsets $I_{\mathrm{cen}(S)}^{--}$, $I_{c^+}^{+-}$, $I_{\mathrm{cen}(S)}^{++}$, and $I_{c^-}^{-+}$ may be empty, when c^- or c^+ lies on ∂P. If $\mathrm{cen}(S) = \{c\}$ consists of a single point, then we have $c^+ = c^- = c$, $I_{\mathrm{cen}(S)}^{--} = I_c^{--}$, and $I_{\mathrm{cen}(S)}^{++} = I_c^{++}$. Otherwise, $\mathrm{cen}(S)$ forms a line segment of positive length. Then, since $\mathrm{cen}(S)$ is of slope -1, by Lemma 22, we have $\chi_1, \chi_2 \in X$ such that $\chi_1 = f_S^{--}(c)$ and $\chi_2 = f_S^{++}(c)$ for any $c \in \mathrm{cen}(S)$. Lemma 23 tells us that $I_{\mathrm{cen}(S)}^{++} \subseteq FR(\chi_1, S)$ and $I_{\mathrm{cen}(S)}^{--} \subseteq FR(\chi_2, S)$. On the other hand, if $I_{c^{\sigma'}}^{\sigma'\overline{\sigma'}} \neq \emptyset$ for any $\sigma' \in \{+, -\}$, then the endpoint $c^{\sigma'}$ is not a boundary point in ∂P. In particular, if $f_S^{\overline{\sigma'}\sigma'}(c^{\sigma'}) \notin \{\chi_1, \chi_2\}$, then we have a third extreme point $\chi_3 = f_S^{\overline{\sigma'}\sigma'}(c^{\sigma'})$ as described in Corollary 21.

Since the boundaries of any two of the eight subsets around $\mathrm{cen}(S)$ always intersect in a subset of a half-chord from c^- or from c^+, we conclude the following.

▶ **Theorem 24.** *For any compact subset $S \subseteq P$ with $|S| \geq 2$, its L_1 geodesic farthest-neighbor Voronoi diagram* $\mathsf{FVD}(S)$ *consists of* $\mathrm{cen}(S)$ *and a subset of the following four segments: two segments that are subsets of half-chords from each endpoint of* $\mathrm{cen}(S)$*.*

7 Algorithms

Now, we are ready to describe our algorithms that compute the extreme points X of S, the diameter, radius, center of S and the farthest-neighbor Voronoi diagram $\mathsf{FVD}(S)$. We keep the generality by setting S to be any nonempty compact subset of P, while an operation that computes $f_S(p)$ for any $p \in P$ is supposed to be processed in at most T time as a black box.

We first describe how to compute the set X of extreme points of S. Pick any $q_0 \in S$. Let $q_i := f_S(q_{i-1})$ for $i \geq 0$, and compute q_i until we have $q_{k+1} = q_{k-1}$ for some $k \geq 2$. By Theorem 8, this ends up with $k \leq 4$. If $k = 4$, then let $\chi_i := q_i$ for each $i \in \{1, 2, 3, 4\}$, and we are done as $X = \{\chi_1, \chi_2, \chi_3, \chi_4\}$ by Theorem 8. Otherwise, we let $\chi_1 := q_{k-1}$ and $\chi_2 := q_k$. Note that $f_S(\chi_1) = \chi_2$ and $f_S(\chi_2) = \chi_1$. Let $r := d(\chi_1, \chi_2)/2$. Then, we compute $\mathrm{B}_{\chi_1}(r)$ and $\mathrm{B}_{\chi_2}(r)$, and their intersection $\mathrm{B}_{\chi_1}(r) \cap \mathrm{B}_{\chi_2}(r)$. Since $r = d(\chi_1, \chi_2)/2$, $\mathrm{B}_{\chi_1}(r) \cap \mathrm{B}_{\chi_2}(r)$ forms a line segment $\overline{z^-z^+}$ of slope 1 or -1, where z^- is to the left of z^+, possibly being a point $z^- = z^+$. Without loss of generality, assume that $\overline{z^-z^+}$ is of slope -1. For each $\sigma \in \{+, -\}$, let $p^\sigma \in P$ be any point in $I_{z^\sigma}^{\sigma\overline{\sigma}}$ if $I_{z^\sigma}^{\sigma\overline{\sigma}}$ is nonempty, or let $p^\sigma := z^\sigma$, otherwise, if $I_{z^\sigma}^{\sigma\overline{\sigma}} = \emptyset$. Let $\chi_3 := f_S(p^-)$ and $\chi_4 := f_S(p^+)$. Then, we have $X = \{\chi_1, \chi_2, \chi_3, \chi_4\}$.

▶ **Lemma 25.** *Let $S \subseteq P$ be a given compact subset, and suppose that $f_S(p)$ for any $p \in P$ can be computed in T time. The above algorithm correctly computes the set X of extreme points of S in $O(n + T)$ time.*

The diameter, radius, center, and farthest-neighbor Voronoi diagram of S can be computed in the same time bound.

▶ **Lemma 26.** *Let $S \subseteq P$ be a given compact subset, and suppose that the set of extreme points of S is known. Then, the following can be computed in $O(n)$ time:* $\mathrm{diam}(S)$*,* $\mathrm{rad}(S)$*,* $\mathrm{cen}(S)$*, and* $\mathsf{FVD}(S)$*.*

Proof. Let X be the set of extreme points of S. Note that $\mathrm{diam}(S) = \max_{\chi,\chi' \in X} d(\chi, \chi')$. Thus, $\mathrm{diam}(S)$ and a diametral pair can be computed in additional $O(n)$ time [7], as $|X| \leq 4$ by Theorem 8. By Corollary 18, we have $\mathrm{rad}(S) = \mathrm{diam}(S)/2$. The set $\mathrm{cen}(S)$ can be computed by intersecting at most four geodesic balls $\mathrm{B}_\chi(\mathrm{rad}(S))$ for $\chi \in X$ by Lemma 19. This can be done in additional $O(n)$ time by computing the shortest path maps [7]. After computing $\mathrm{cen}(S)$, the farthest-neighbor Voronoi diagram $\mathsf{FVD}(S)$ can be found by considering the eight subsets around $\mathrm{cen}(S)$ by Lemma 23. As $\mathsf{FVD}(S)$ consists of at most five segments, it can be found in additional $O(n)$ time. ◀

Now, we describe the subprocedure that computes $f_S(p)$ for any $p \in P$. Here, we assume that S is a finite set of m points.

▶ **Lemma 27.** *Let S be a set of m points in P. Then, $f_S(p)$ for any $p \in P$ can be computed in $O(n + m \log n)$ time. If the order of $S \cap \partial\mathrm{rconv}(S)$ along $\partial\mathrm{rconv}(S)$ is provided, then $O(n + m)$ time is sufficient.*

Proof. As a preprocessing, we build in $O(n)$ time the data structure of Guibas and Hershberger [6] that evaluates $d(p, q)$ for any $p, q \in P$ in $O(\log n)$ time. Given any $p \in P$, we compute $d(p, q)$ for all $q \in S$, and gather the set $F_S(p)$ of farthest neighbors of p in $O(m \log n)$

time. And pick the least element in $F_S(p)$ with respect to the total order $\prec$, and report it as $f_S(p)$. This takes $O(n + m \log n)$ time.

If the order of $S \cap \partial\text{rconv}(S)$ along $\partial\text{rconv}(S)$ is known, then we can apply the fast matrix search technique of Hershberger and Suri [9]. This takes $O(n + m)$ time. ◀

Another interesting case is when $S = P$. Since $\text{rconv}(P) = P$, in this case, we know the order of points $P \cap \partial\text{rconv}(P) = \partial P$. Moreover, since $f_P(p)$ is always a vertex of P, we have the following corollary.

▶ **Corollary 28.** *For any $p \in P$, $f_P(p)$ can be computed in $O(n)$ time.*

Combining all these results, we obtain the following theorems.

▶ **Theorem 29.** *Let P be a simple n-gon and S be a set of m points in P. Then, the set of L_1 geodesic extreme points of S, $\text{diam}(S)$, $\text{rad}(S)$, $\text{cen}(S)$, and $\mathsf{FVD}(S)$ can be computed in $O(n + m \log n)$ time. If the order of $S \cap \partial\text{rconv}(S)$ along $\partial\text{rconv}(S)$ is provided, then $O(n + m)$ time is sufficient.*

▶ **Theorem 30.** *Let P be a simple n-gon. Then, the set of L_1 geodesic extreme points of P, $\text{diam}(P)$, $\text{rad}(P)$, $\text{cen}(P)$, and $\mathsf{FVD}(P)$ can be computed in $O(n)$ time.*

8 L_1 Geodesic Two-Center

In this section, we address the two-center problem for any compact subset $S \subseteq P$ under the L_1 geodesic distance. The L_1 *geodesic two-center problem* asks a pair of points $c_1, c_2 \in P$ that minimize $\max_{q \in S} \min\{d(q, c_1), d(q, c_2)\}$. Such a pair (c_1, c_2) is called an L_1 *geodesic two-center* of S in P, or shortly a *two-center* of S. Let $\text{rad}_2(S) := \max_{q \in S} \min\{d(q, c_1), d(q, c_2)\}$ be the optimal objective value for the problem, called the *two-radius* or 2-*radius* of S. A two-center (c_1, c_2) induces a bipartition (S_1, S_2) of S such that $S_1 = S \cap \text{B}_{c_1}(\text{rad}_2(S))$ and $S_2 = S \setminus S_1$. Conversely, a bipartition (S_1, S_2) of S is called *optimal* if $\text{rad}_2(S) = \max\{\text{rad}(S_1), \text{rad}(S_2)\}$. Note that in general we have $\max\{\text{rad}(S_1), \text{rad}(S_2)\} \geq \text{rad}_2(S)$ if $S_1 \cup S_2 = S$. Given an optimal bipartition (S_1, S_2) of S, observe that any $c_1 \in \text{cen}(S_1)$ and $c_2 \in \text{cen}(S_2)$ form a two-center (c_1, c_2) of S. Thus, the two-center problem is equivalent to finding an optimal bipartition of S.

Another closely related problem is the *minmax-diameter bipartition problem* that asks a bipartition (S_1, S_2) of S such that $\max\{\text{diam}(S_1), \text{diam}(S_2)\}$ is minimized. Thus, this problem is to compute the 2-*diameter* $\text{diam}_2(S)$ of S defined to be the minimum value of $\max\{\text{diam}(S_1), \text{diam}(S_2)\}$ over all possible bipartitions (S_1, S_2) of S. In the L_1 geodesic case, the two-center problem is equivalent to the minmax-diameter bipartition problem.

▶ **Lemma 31.** *For any compact subset $S \subseteq P$, it holds that $\text{rad}_2(S) = \text{diam}_2(S)/2$.*

Thus, if (S_1, S_2) is the optimal solution to the minmax-diameter bipartition problem, then it is an optimal bipartition for the two-center problem.

In the following, we let X be the set of extreme points of S.

▶ **Lemma 32.** *There exists an optimal bipartition (S_1, S_2) of S such that for each $\chi \in X$, $\chi \in S_1$ if and only if $f_S(\chi) \in S_2$.*

The following is our key lemma.

▶ **Lemma 33.** *There exists an optimal bipartition (S_1^*, S_2^*) of S such that*

$$S_1^* = S \cap \bigcup_{\chi \in X \cap S_2^*} FR(\chi, S) \quad \textit{and} \quad S_2^* = S \cap \bigcup_{\chi \in X \cap S_1^*} FR(\chi, S).$$

Our algorithm that computes a two-center of S is described as follows: First, compute the set X of extreme points of S, and the farthest-neighbor Voronoi diagram $\mathsf{FVD}(S)$. For each bipartition (X_1, X_2) of X that satisfies the property of Lemma 32, let $S_1 := S \cap \bigcup_{\chi \in X_2} FR(\chi, S)$ and $S_2 := S \cap \bigcup_{\chi \in X_1} FR(\chi, S)$. Then, compute $\mathrm{diam}(S_1)$ and $\mathrm{diam}(S_2)$, and keep the minimum of $\max\{\mathrm{diam}(S_1), \mathrm{diam}(S_2)\}$ for all such bipartitions of X.

Let (S_1^*, S_2^*) be the bipartition of S with a minimum value of $\max\{\mathrm{diam}(S_1^*), \mathrm{diam}(S_2^*)\}$. Then, (S_1^*, S_2^*) is an optimal bipartition and $\mathrm{diam}_2(S) = \max\{\mathrm{diam}(S_1^*), \mathrm{diam}(S_2^*)\}$ by Lemmas 31 and 33. A two-center (c_1, c_2) of S can be found by choosing any $c_1 \in \mathrm{cen}(S_1^*)$ and any $c_2 \in \mathrm{cen}(S_2^*)$.

The above algorithm works properly when S is a finite set of points in P.

▶ **Theorem 34.** *Let S be a set of m points in a simple n-gon P. Then, an L_1 geodesic two-center of S can be computed in $O(n + m \log n)$ time.*

Another interesting special case is when $S = P$.

▶ **Theorem 35.** *An L_1 geodesic two-center of a simple n-gon can be computed in $O(n)$ time.*

References

1 H.-K. Ahn, L. Barba, P. Bose, J.-L. De Carufel, M. Korman, and E. Oh. A linear-time algorithm for the geodesic center of a simple polygon. In *Proc. 31st Symp. Comput. Geom. (SoCG 2015)*, volume 34 of *LIPIcs*, pages 209–223, 2015.

2 B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete Comput. Geom.*, 9:217–255, 1993.

3 S.W. Bae, M. Korman, Y. Okamoto, and H. Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. *Comput. Geom.*, 48(6):495–505, 2015.

4 T. M. Chan. More planar two-center algorithms. *Comput. Geom.*, 13(3):189–198, 1999.

5 Z. Drezner. On the rectangular p-center problem. *Naval Research Logistics (NRL)*, 34(2):229–234, 1987.

6 L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.

7 L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

8 J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom.*, 4(2):63–97, 1994.

9 J. Hershberger and S. Suri. Matrix searching with the shortest path metric. *SIAM J. Comput.*, 26(6):1612–1634, 1997.

10 E. Oh, S.W. Bae, and H.-K. Ahn. Computing a geodesic two-center of points in a simple polygon. In *Proc. 12th Latin American Theoretical Info. Symp. (LATIN 2016)*, volume 9644 of *LNCS*, pages 646–658, 2016.

11 E. Oh, L. Barba, and H.-K. Ahn. The farthest-point geodesic Voronoi diagram of points on the boundary of a simple polygon. In *Proc. 32nd Symp. Comput. Geom. (SoCG 2016)*, LIPIcs, pages 56:1–56:15, 2016.

12 E. Oh, J.-L. De Carufel, and H.-K. Ahn. The 2-center problem in a simple polygon. In *Proc. 26th Int'l Symp. Algo. Comput. (ISAAC 2015)*, volume 9472 of *LNCS*, pages 307–317, 2015.

13 G. T. Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle*, 3:9–42, 1989.

Approximate Clustering via Metric Partitioning*

Sayan Bandyapadhyay[1] **and Kasturi Varadarajan**[2]

1 **Department of Computer Science, University of Iowa, Iowa City, USA**
sayan-bandyapadhyay@uiowa.edu

2 **Department of Computer Science, University of Iowa, Iowa City, USA**
kasturi-varadarajan@uiowa.edu

Abstract

In this paper we consider two metric covering/clustering problems – *Minimum Cost Covering Problem* (MCC) and k-clustering. In the MCC problem, we are given two point sets X (clients) and Y (servers), and a metric on $X \cup Y$. We would like to cover the clients by balls centered at the servers. The objective function to minimize is the sum of the α-th power of the radii of the balls. Here $\alpha \geq 1$ is a parameter of the problem (but not of a problem instance). MCC is closely related to the k-clustering problem. The main difference between k-clustering and MCC is that in k-clustering one needs to select k balls to cover the clients.

For any $\varepsilon > 0$, we describe quasi-polynomial time $(1+\varepsilon)$ approximation algorithms for both of the problems. However, in case of k-clustering the algorithm uses $(1+\varepsilon)k$ balls. Prior to our work, a 3^α and a c^α approximation were achieved by polynomial-time algorithms for MCC and k-clustering, respectively, where $c > 1$ is an absolute constant. These two problems are thus interesting examples of metric covering/clustering problems that admit $(1+\varepsilon)$-approximation (using $(1+\varepsilon)k$ balls in case of k-clustering), if one is willing to settle for quasi-polynomial time. In contrast, for the variant of MCC where α is part of the input, we show under standard assumptions that no polynomial time algorithm can achieve an approximation factor better than $O(\log |X|)$ for $\alpha \geq \log |X|$.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Approximation Algorithms, Clustering, Covering, Probabilistic Partitions

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.15

1 Introduction

We consider two metric covering/clustering problems. In the first problem, we are given two point sets X (clients) and Y (servers), and a metric d on $X \cup Y$. For $z \in X \cup Y$ and $r \geq 0$, the ball $B(z,r)$ centered at z and having radius $r \geq 0$ is the set $\{y \in X \cup Y | d(z,y) \leq r\}$. A *cover* for a subset $P \subseteq X$ is a set of balls, each centered at a point of Y, whose union contains P. The *cost* of a set $\mathcal{B} = \{B_1, \ldots, B_k\}$ of balls, denoted by $\text{cost}(\mathcal{B})$, is $\sum_{i=1}^{k} r(B_i)^\alpha$, where $r(B_i)$ is the radius of B_i, and $\alpha \geq 1$ is a parameter of the problem (but not of a problem instance). The goal is to compute a minimum cost *cover* for the clients X. We refer to this problem as the *Minimum Cost Covering Problem* (MCC).

In the second problem, we are given a set X of n points, a metric d on X, and a positive

[*] This material is based upon work supported by the National Science Foundation under Grant CCF-1318996.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 15; pp. 15:1–15:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

integer k. Unlike in the case of MCC, here each ball is centered at a point in X.[1] The *cost* $\mathrm{cost}(\mathcal{B})$ of a set $\mathcal{B}$ of balls is defined exactly in the same way as in the case of MCC. The goal is to find a set $\mathcal{B}$ of k balls whose union contains all the points in X and $\mathrm{cost}(\mathcal{B})$ is minimized. We refer to this problem as k-clustering.

Inspired by applications in wireless networks, MCC has been well studied [22]. One can consider the points in Y as the potential locations of mobile towers and the points in X as the locations of customers. A tower can be configured in a way so that it can serve the customers lying within a certain distance. But the service cost increases with the distance served. The goal is to serve all the customers minimizing the total cost. For modelling the energy needed for wireless transmission, it is common to consider the value of α to be at least 1.

For the MCC problem with $\alpha = 1$, a primal-dual algorithm of Charikar and Panigrahy [10] leads to an approximation guarantee of 3; their result generalizes to $\alpha \geq 1$, with an approximation guarantee of 3^{α}. The problem is known to be NP-hard for $\alpha > 1$, even when X and Y are points in the Euclidean plane [2]. The case $\alpha = 1$ has received particular attention. The first PTAS for the Euclidean plane was designed by Lev-Tov and Peleg [22]. Later, Gibson *et. al* [17] have designed a polynomial time exact algorithm for this problem when X and Y are points in the plane, and the underlying distance function d is either the l_1 or l_∞ metric. For the l_2 metric they also get an exact algorithm if one assumes two candidate solutions can be compared efficiently; without this assumption, they get a $(1+\varepsilon)$ approximation. Their algorithm is based on a separator theorem that, for any optimal solution, proves the existence of a balanced separator that intersects with at most 12 balls in the solution. In a different work they have also extended the exact algorithm to arbitrary metric spaces [16]. The running time is quasi-polynomial if the aspect ratio of the metric (ratio of maximum to minimum interpoint distance) is bounded by a polynomial in the number of points. When the aspect ratio is not polynomially bounded, they obtain a $(1+\varepsilon)$ approximation in quasi-polynomial time. Their algorithms are based on a partitioning of the metric space that intersects a small number of balls in the optimal cover.

When $\alpha > 1$, the structure that holds for $\alpha = 1$ breaks down. It is no longer the case, even in the Euclidean plane, that there is a good separator (or partition) that intersects a small number of balls in an optimal solution. In the case $\alpha = 2$ and the Euclidean plane, the objective function models the total area of the served region, which arises in many practical applications. Hence this particular version has been studied in a series of works. Chuzhoy developed an unpublished 9-factor approximation algorithm for this version. Freund and Rawitz [15] present this algorithm and give a primal fitting interpretation of the approximation factor. Bilo *et. al* [9] have extended the techniques of Lev-Tov and Peleg [22] to get a PTAS that works for any $\alpha \geq 1$ and for any fixed dimensional Euclidean space. The PTAS is based on a sophisticated use of the *shifting strategy* which is a popular technique in computational geometry for solving problems in $\mathbb{R}^d$ [13, 19]. For general metrics, however, the best known approximation guarantee for $\alpha > 1$ remains the already mentioned 3^{α} [10].

The k-clustering problem has applications in many fields including Data Mining, Machine Learning and Image Processing. Over the years it has been studied extensively from both theoretical and practical perspectives [9, 10, 12, 16, 17, 23]. The problem can be seen as a variant of MCC where $Y = X$ and at most k balls can be chosen to cover the points in X. As one might think, the constraint on the number of balls that can be used in k-clustering

[1] Our results do generalize to the problem where we distinguish between clients and servers as in the MCC.

makes it relatively harder than MCC. Thus all the hardness results for MCC also hold for k-clustering. For $\alpha = 1$, Charikar and Panigrahy [10] present a polynomial time algorithm with an approximation guarantee of about 3.504. Gibson *et. al* [16, 17] obtain the same results for k-clustering with $\alpha = 1$ as the ones described for MCC, both in $\mathbb{R}^d$ and arbitrary metrics. Recently, Salavatipour and Behsaz [8] have obtained a polynomial time exact algorithm for $\alpha = 1$ and metrics of unweighted graphs, if we assume that no singleton clusters are allowed. However, in case of $\alpha > 1$ the best known approximation factor (in polynomial time) for general metrics is c^α, for some absolute constant $c > 1$; this follows from the analysis of Charikar and Panigrahy [10], who explicitly study only the case $\alpha = 1$. In fact, no better polynomial time approximation is known even for the Euclidean plane. We note that though the polynomial time algorithm in [9] yields a $(1 + \varepsilon)$ approximation for k-clustering in any fixed dimensional Euclidean space and for $\alpha \geq 1$, it can use $(1 + \varepsilon)k$ balls.

In addition to k-clustering many other clustering problems (k-means, k-center, k-median etc.) have been well studied [5, 11, 24, 18].

In this paper we address the following interesting question. Can the techniques employed by [9] for fixed dimensional Euclidean spaces be generalized to give $(1 + \varepsilon)$ approximation for MCC and k-clustering in any metric space? Our motivation for studying the problems in a metric context is partly that it includes two geometric contexts: (a) high dimensional Euclidean spaces; and (b) shortest path distance metric in the presence of polyhedral obstacles in $\mathbb{R}^2$ or $\mathbb{R}^3$.

1.1 Our Results and Techniques

In this paper we consider the metric MCC and k-clustering with $\alpha \geq 1$. For any $\epsilon > 0$, we design a $(1 + \epsilon)$-factor approximation algorithm for MCC that runs in quasi-polynomial time, that is, in $2^{(\log mn/\varepsilon)^c}$ time, where $c > 0$ is a constant, $m = |Y|$, and $n = |X|$. We also have designed a similar algorithm for k-clustering that uses at most $(1 + \varepsilon)k$ balls and yields a solution whose cost is at most $(1 + \varepsilon)$ times the cost of an optimal k-clustering solution. The time complexity of the latter algorithm is also quasi-polynomial. As already noted, somewhat stronger guarantees are already known for the case $\alpha = 1$ of these problems [16], but the structural properties that hold for $\alpha = 1$ make it rather special.

The results in this paper should be compared with the polynomial time algorithms [10] that guarantee 3^α approximation for MCC and c^α approximation for k-clustering. The MCC and k-clustering are thus interesting examples of metric covering/clustering problems that admit $(1 + \varepsilon)$-approximation (using $(1 + \varepsilon)k$ balls in case of k-clustering), if one is willing to settle for quasi-polynomial time. From this perspective, our results are surprising, as most of the problems in general metrics are APX-hard. The MCC and k-clustering are also examples where the techniques used in fixed dimensional Euclidean spaces generalize nicely to metric spaces. This is in contrast to the *facility location problem* [3].

The algorithms that we have designed for both of the problems use similar techniques that exploit the following key property of optimal covers: there are only a "small" number of balls whose radius is "large". We can therefore afford to guess these balls by an explicit enumeration. However, there can be a "large" number of balls with "small" radius. To help 'find' these, we partition the metric space into blocks (or subsets) with at most half the original diameter, and recurse on each block. We have to pay a price for this recursion in the approximation guarantee. This price depends on the number of blocks in the partition that a small radius ball can intersect. (This is not an issue in the case $\alpha = 1$, where each ball that is not guessed intersects precisely one of the blocks [16].)

We are led to the following problem: is there a way to probabilistically partition a metric space into blocks of at most half the diameter, so that for any ball with "small" radius, the expected number of blocks that intersect the ball can be nicely bounded? The celebrated partitioning algorithms of Bartal [6] and Fakcharoenphol, Rao, and Talwar [14] guarantee that the probability that such a ball is intersected by two or more blocks is nicely bounded. However, their bounds on the probability that a small ball is intersected do not directly imply a good bound on the expected number of blocks intersected by a small ball. Indeed, if one employs the partitioning algorithm of [14], the expected number of blocks intersected by a small ball can be quite "large" . Fortunately, the desired bound on the expectation can be shown to hold for the algorithm of Bartal [6], even though he did not study the expectation itself. We use a similar partitioning scheme and derive the expectation bound in Section 2, using an analysis that closely tracks previous work [1, 7, 20]. While the bound on the expectation is easily derived from previous work, our work is the first to study and fruitfully apply this bound.

The algorithms for MCC and k-clustering, which use the partitioning scheme of Section 2, are described in Section 3 and 4, respectively. In Section 5, we consider the approximability of a variant of the MCC where we allow α to be part of the input. For $\alpha \geq \log |X|$, we show, under standard complexity theoretic assumptions, that no polynomial (or quasi-polynomial) time algorithm for MCC can achieve an approximation factor better than $O(\log |X|)$. This partly explains the dependence on α of the running time of our algorithms.

2 The Partitioning Scheme

Let Z be a point set with an associated metric d, let $P \subseteq Z$ be a point set with at least 2 points, and $n \geq |P|$ be a parameter. For $Q \subseteq Z$, denote the maximum interpoint distance (or diameter) of Q by $\text{diam}(Q)$. Consider any partition of P into subsets (or blocks) $\{P_1, P_2, \ldots, P_t\}$, where $2 \leq t \leq |P|$. Abusing notation, we will also view $\{P_1, P_2, \ldots, P_t\}$ as a sequence of blocks. We say that P_i non-terminally (resp. terminally) intersects a ball B if P_i intersects B and it is not (resp. it is) the last set in the sequence $P_1, P_2, \ldots, P_t$ that intersects B. We would like to find a partition $\{P_1, P_2, \ldots, P_t\}$ of P that ensures the following properties:

1. For each $1 \leq i \leq t$, $\text{diam}(P_i) \leq \text{diam}(P)/2$.
2. For any ball B (centered at some point in Z) of radius $r \leq \frac{\text{diam}(P)}{16 \log n}$, the expected size of the set $\{i | P_i \cap B \neq \emptyset\}$ is at most $1 + c\frac{r}{\text{diam}(P)} \log n$, where $c > 0$ is a constant. In other words, the expected number of blocks in the partition that intersect B is at most $1 + c\frac{r}{\text{diam}(P)} \log n$.
3. For any ball B (centered at some point in Z) of radius $r \leq \frac{\text{diam}(P)}{16 \log n}$, the expected number of blocks in the partition that non-terminally intersect B is at most $c\frac{r}{\text{diam}(P)} \log n$, where $c > 0$ is a constant.

We note that the second property follows from the third, as the number of blocks that intersect ball B is at most one more than the number of blocks that non-terminally intersect B. We design a probabilistic partitioning algorithm that finds a partition with the desired properties. We refer the reader to the full version of the paper for the algorithm and its analysis [4]. We conclude by summarizing the result.

▶ **Theorem 1.** *Let Z be a point set with an associated metric d, let $P \subseteq Z$ be a point set with at least 2 points, and $n \geq |P|$ be a parameter. There is a polynomial-time probabilistic algorithm RAND-PARTITION(P) that partitions P into blocks $\{P_1, P_2, \ldots, P_t\}$ and has the following guarantees:*

1. *For each $1 \leq i \leq t$, diam$(P_i) \leq$ diam$(P)/2$.*

2. *There is a constant $c > 0$ so that for any ball B (centered at some point in Z) of radius $r \le \frac{diam(P)}{16 \log n}$, the expected size of the set $\{i | P_i \cap B \neq \emptyset\}$ is at most $1 + c\frac{r}{diam(P)} \log n$ and the expected number of blocks that non-terminally intersect B is at most $c\frac{r}{diam(P)} \log n$.*

3 Algorithm for MCC

We now describe our $(1+\epsilon)$-factor approximation algorithm for the MCC problem. Recall that we are given a set X of clients, a set Y of servers, and a metric d on $X \cup Y$. We wish to compute a cover for X with minimum cost. Let $m = |Y|$ and $n = |X|$.

For $P \subseteq X$, let $\text{opt}(P)$ denote some optimal cover for P. Denote by $\text{cost}(B)$ the cost of a ball B (the α-th power of B's radius) and by $\text{cost}(\mathcal{B})$ the cost $\sum_{B \in \mathcal{B}} \text{cost}(B)$ of a set $\mathcal{B}$ of balls.

To compute a cover for P, our algorithm first guesses the set $\mathcal{Q} \subseteq \text{opt}(P)$ consisting of all the large balls in $\text{opt}(P)$. As we note in the structure lemma below, we may assume that the number of large balls in $\text{opt}(P)$ is small. We then use the algorithm of Theorem 1 to partition P into $\{P_1, P_2, \ldots, P_t\}$. For each $1 \le i \le t$, we recursively compute a cover for the set $P'_i \subseteq P_i$ of points not covered by $\mathcal{Q}$.

To obtain an approximation guarantee for this algorithm, we use the guarantees of Theorem 1. With this overview, we proceed to the structure lemma and a complete description of the algorithm.

3.1 A Structure Lemma

It is not hard to show that for any $\gamma \ge 1$ and $P \subseteq X$ such that $\text{diam}(P)$ is at least a constant factor of $\text{diam}(X \cup Y)$, $\text{opt}(P)$ contains at most $(c/\gamma)^\alpha$ balls of radius at least $\text{diam}(P)/\gamma$. Here c is some absolute constant. The following structural lemma extends this fact.

▶ **Lemma 2.** *Let $P \subseteq X$, $0 < \lambda < 1$ and $\gamma \ge 1$, and suppose that opt(P) does not contain any ball of radius greater than or equal to $2\alpha \cdot diam(P)/\lambda$. Then the number of balls in opt(P) of radius greater than or equal to $diam(P)/\gamma$ is at most $c(\lambda, \gamma) := (9\alpha\gamma/\lambda)^\alpha$.*

Proof. Suppose that $\text{opt}(P)$ does not contain any ball of radius greater than or equal to $2\alpha \cdot \text{diam}(P)/\lambda$. Note that each ball in $\text{opt}(P)$ intersects P and has radius at most $2\alpha \cdot \text{diam}(P)/\lambda$. Thus the point set $\{z \in X \cup Y \mid z \in B \text{ for some } B \in \text{opt}(P)\}$ has diameter at most $\text{diam}(P) + 8\alpha \cdot \text{diam}(P)/\lambda \le 9\alpha \cdot \text{diam}(P)/\lambda$. It follows that there is a ball centered at a point in Y, with radius at most $9\alpha \cdot \text{diam}(P)/\lambda$ that contains P.

Let t denote the number of balls in $\text{opt}(P)$ of radius greater than or equal to $\text{diam}(P)/\gamma$. By optimality of $\text{opt}(P)$, we have $t \cdot (\text{diam}(P)/\gamma)^\alpha \le (9\alpha \cdot \text{diam}(P)/\lambda)^\alpha$. Thus $t \le (9\alpha\gamma/\lambda)^\alpha$. ◀

3.2 The Algorithm

We may assume that the minimum distance between two points in X is 1. Let $L = 1 + \log(\text{diam}(X))$. As we want a $(1+\varepsilon)$-approximation, we fix a parameter $\lambda = \varepsilon/2L$. Let $\gamma = \frac{c \log n}{\lambda}$, where c is the constant in Theorem 1. Denote $\mathcal{D}$ to be the set of balls such that each ball is centered at a point of $y \in Y$ and has radius $r = d(x, y)$ for some $x \in X$. We note that for any $P \subseteq X$, any ball in $\text{opt}(P)$ must belong to this set. Note that $|\mathcal{D}| \le mn$. Recall that $c(\lambda, \gamma) = (9\alpha\gamma/\lambda)^\alpha$.

With this terminology, the procedure POINT-COVER(P) described as Algorithm 1 returns a cover of $P \subseteq X$. If $|P|$ is smaller than some constant, then the procedure returns an

Algorithm 1 POINT-COVER(P)

```
Require: A subset $P \subseteq X$.
Ensure: A cover of the points in $P$.
if $|P|$ is smaller than some constant $\kappa$ then
    return a minimum solution by checking all covers with at most $\kappa$ balls.
sol $\leftarrow$ the best cover with one ball
cost $\leftarrow$ cost($sol$)
Let $\{P_1, \dots, P_\tau\}$ be the set of nonempty subsets returned by RAND-PARTITION($P$)
Let $\mathcal{B}$ be the set of balls in $\mathcal{D}$ having radius greater than $\frac{\mathrm{diam}(P)}{\gamma}$
for each $\mathcal{Q} \subseteq \mathcal{B}$ of size at most $c(\lambda, \gamma)$ do
    for $i = 1$ to $\tau$ do
        Let $P'_i = \{p \in P_i \mid p \notin \bigcup_{B \in \mathcal{Q}} B\}$
    $\mathcal{Q}' \leftarrow \mathcal{Q} \cup \bigcup_{\mathrm{i}=1}^{\tau}$ POINT-COVER($P'_i$)
    if cost($\mathcal{Q}'$) < cost then
        cost $\leftarrow$ cost($\mathcal{Q}'$)
        sol $\leftarrow \mathcal{Q}'$
return sol
```

optimal solution by searching all covers with a constant number of balls. In the general case, one candidate solution is the best single ball solution. For the other candidate solutions, the procedure first computes a partition $\{P_1, \dots, P_\tau\}$ of P, using the RAND-PARTITION(P) procedure. Here RAND-PARTITION(P) is called with $Z = X \cup Y$ and $n = |X| \geq |P|$. Then it iterates over all possible subsets of $\mathcal{D}$ of size at most $c(\lambda, \gamma)$ containing balls of radius greater than diam$(P)/\gamma$. For each such subset $\mathcal{Q}$ and $1 \leq i \leq \tau$, it computes the set $P'_i \subseteq P_i$ of points not covered by $\mathcal{Q}$. It then makes recursive calls and generates the candidate solution $\mathcal{Q} \cup \bigcup_{\mathrm{i}=1}^{\tau}$ POINT-COVER(P'_i). Note that all the candidate solutions are actually valid covers for P. Among these candidate solutions the algorithm returns the best solution.

Our overall algorithm for MCC calls the procedure POINT-COVER(X) to get a cover of X.

3.3 Approximation Guarantee

For $P \subseteq X$, let level(P) denote the smallest non-negative integer i such that diam$(P) < 2^i$. As the minimum interpoint distance in X is 1, level$(P) = 0$ if and only if $|P| \leq 1$. Note that level$(X) \leq L$.

The following lemma bounds the quality of the approximation of our algorithm.

▶ **Lemma 3.** *POINT-COVER(P) returns a solution whose expected cost is at most $(1 + \lambda)^l$ cost(opt(P)), where l = level(P).*

Proof. We prove this lemma using induction on l. If $l = 0$, then $|P| \leq 1$ and POINT-COVER(P) returns an optimal solution, whose cost is cost(opt(P)). Thus assume that $l \geq 1$ and the statement is true for subsets having level at most $l - 1$. Let $P \subseteq X$ be a point set with level$(P) = l$. If $|P|$ is smaller than the constant threshold κ, POINT-COVER(P) returns an optimal solution. So we may assume that $|P|$ is larger than this threshold. We have two cases.

Case 1: There is some ball in opt(P) whose radius is at least $2\alpha \cdot \mathrm{diam}(P)/\lambda$. Let B denote such a ball and $r(B) \geq 2\alpha \cdot \mathrm{diam}(P)/\lambda$ be its radius. Since $(1 + \lambda/2\alpha) r(B) \geq r(B) + \mathrm{diam}(P)$, the concentric ball of radius $(1+\lambda/2\alpha)r(B)$ contains P. It follows that there is a cover for P that consists of a single ball and has cost at most

$$(1+\lambda/2\alpha)^\alpha r(B)^\alpha \leq (1+\lambda)\text{cost}(\text{opt}(P)) \leq (1+\lambda)^l\text{cost}(\text{opt}(P)).$$

Case 2: There is no ball in $\text{opt}(P)$ whose radius is at least $2\alpha\cdot\text{diam}(P)/\lambda$. Let $\mathcal{Q}_0 \subseteq \text{opt}(P)$ contain those balls of radius at least $\text{diam}(P)/\gamma$. It follows from Lemma 2 that $|\mathcal{Q}_0| \leq c(\lambda,\gamma)$. Thus the algorithm considers a $\mathcal{Q}$ with $\mathcal{Q} = \mathcal{Q}_0$. Fix this iteration. Also fix the partition $\{P_1,\dots,P_\tau\}$ of P computed by RAND-PARTITION(P). RAND-PARTITION ensures that $\text{diam}(P_i) \leq \text{diam}(P)/2$ for $1 \leq i \leq \tau$. Thus $\text{diam}(P_i') \leq \text{diam}(P)/2$ and the level of each P_i' is at most $l-1$. Hence by induction the expected value of cost(POINT-COVER(P_i')) is at most $(1+\lambda)^{l-1}\text{cost}(\text{opt}(P_i'))$.

Let $\mathcal{S}' = \text{opt}(P) \setminus \mathcal{Q}_0$. We argue below that the expected value of $\sum_{i=1}^{\tau} \text{cost}(\text{opt}(P_i'))$ is at most $(1+\lambda)\text{cost}(\mathcal{S}')$. Assuming this, we have

$$\begin{aligned} E[\text{cost}(\mathcal{Q}_0 \cup \bigcup_{i=1}^{\tau} \text{POINT-COVER}(P_i'))] &\leq \text{cost}(\mathcal{Q}_0) + (1+\lambda)^{l-1}E[\sum_{i=1}^{\tau}\text{cost}(\text{opt}(P_i'))] \\ &\leq \text{cost}(\mathcal{Q}_0) + (1+\lambda)^l\text{cost}(\mathcal{S}') \\ &\leq (1+\lambda)^l\text{cost}(\text{opt}(P)). \end{aligned}$$

Thus POINT-COVER(P) returns a solution whose expected cost is at most $(1+\lambda)^l\text{cost}(\text{opt}(P))$, as desired.

We now argue that the expected value of $\sum_{i=1}^{\tau}\text{cost}(\text{opt}(P_i'))$ is at most $(1+\lambda)\text{cost}(\mathcal{S}')$. Let $\mathcal{B}_i$ consist of those balls in $\mathcal{S}'$ that intersect P_i. For $B \in \mathcal{S}'$, let $\mu(B)$ denote the number of blocks in the partition $\{P_1,\dots,P_\tau\}$ that B intersects. Because $\mathcal{B}_i$ is a cover for P_i', we have $\text{cost}(\text{opt}(P_i')) \leq \text{cost}(\mathcal{B}_i)$. Thus

$$\sum_{i=1}^{\tau}\text{cost}(\text{opt}(P_i')) \leq \sum_{i=1}^{\tau}\text{cost}(\mathcal{B}_i) = \sum_{B\in\mathcal{S}'}\mu(B)\text{cost}(B).$$

By definition of $\mathcal{Q}_0$, any ball $B \in \mathcal{S}' = \text{opt}(P)\setminus\mathcal{Q}_0$ has radius at most $\frac{\text{diam}(P)}{\gamma} = \frac{\lambda\cdot\text{diam}(P)}{c\log n}$, where c is the constant in Theorem 1. We may assume that $c \geq 16$ and hence $\frac{\lambda\cdot\text{diam}(P)}{c\log n} \leq \frac{\text{diam}(P)}{16\log n}$. Theorem 1 now implies that

$$E[\mu(B)] \leq 1 + \frac{c\cdot r(B)\log n}{\text{diam}(P)} \leq 1 + \frac{c\log n}{\text{diam}(P)}\cdot\frac{\lambda\cdot\text{diam}(P)}{c\log n} = 1+\lambda.$$

Thus the expected value of $\sum_{i=1}^{\tau}\text{cost}(\text{opt}(P_i'))$ is at most

$$\sum_{B\in\mathcal{S}'}E[\mu(B)]\text{cost}(B) \leq (1+\lambda)\sum_{B\in\mathcal{S}'}\text{cost}(B) = (1+\lambda)\text{cost}(\mathcal{S}'),$$

as claimed. ◀

We conclude that the expected cost of the cover returned by POINT-COVER(X) is at most $(1+\lambda)^L\text{cost}(\text{opt}(X)) \leq (1+\varepsilon)\text{cost}(\text{opt}(X))$, since $\lambda = \varepsilon/2L$.

Now consider the time complexity of the algorithm. POINT-COVER(P) makes $(mn)^{O(c(\lambda,\gamma))}$ direct recursive calls on subsets of diameter at most $\text{diam}(P)/2$. Thus the overall time complexity of POINT-COVER(X) can be bounded by $(mn)^{O(c(\lambda,\gamma)L)}$. Plugging in $\lambda = \varepsilon/2L$, $\gamma = c\log n/\lambda$, and $c(\lambda,\gamma) = (9\alpha\gamma/\lambda)^\alpha$, we conclude

▶ **Theorem 4.** *There is an algorithm for MCC that runs in time* $(mn)^{O(\frac{\alpha L^2 \mathrm{log} n}{\epsilon^2})^{\alpha} L}$ *and returns a cover whose expected cost is at most* $(1+\varepsilon)$ *times the optimal. Here* L *is 1 plus the logarithm of the aspect ratio of* X*, that is, the ratio of the maximum and minimum interpoint distances in the client set* X*.*

Using relatively standard techniques, which we omit here, we can pre-process the input to ensure that the ratio of the maximum and minimum interpoint distances in X is upper bounded by a polynomial in $\frac{mn}{\varepsilon}$. However, this affects the optimal solution by a factor of at most $(1+\epsilon)$. After this pre-processing, we have $L = O(\log \frac{mn}{\varepsilon})$. Using the algorithm in Theorem 4 after the pre-processing, we obtain a $(1+\epsilon)$ approximation with the quasi-polynomial running time $O(2^{\log^{O(1)} mn})$. Here the $O(1)$ hides a constant that depends on α and ε.

4 Algorithm for k-clustering

Recall that in k-clustering we are given a set X of points, a metric d on X, and a positive integer k. Let $|X| = n$. For $P \subseteq X$ and integer $\kappa \geq 0$, let $\mathrm{opt}(P,\kappa)$ denote an optimal solution of κ-clustering for P (using balls whose center can be any point in X). We reuse the notions of $\mathrm{level}(P)$, $\mathrm{cost}(B)$ and $\mathrm{cost}(\mathcal{B})$ from Section 3, for a point set P, a ball B, and a set $\mathcal{B}$ of balls, respectively. Denote $\mathcal{D}$ to be the set of balls such that each ball is centered at a point of $y \in X$ and has radius $r = d(x,y)$ for some $x \in X$. We note that for any $P \subseteq X$, any ball in $\mathrm{opt}(P,\kappa)$ must belong to this set. Note that $|\mathcal{D}| \leq n^2$.

To start with we prove a structure lemma for k-clustering.

▶ **Lemma 5.** *Let* $P \subseteq X$*,* κ *be a positive integer, and* $\gamma \geq 1$*. Then the number of balls in* $opt(P,\kappa)$ *of radius greater than or equal to* $diam(P)/\gamma$ *is at most* $c(\gamma) := \gamma^{\alpha}$*.*

Proof. Note that any ball centered at a point in P and having radius $\mathrm{diam}(P)$ contains all the points of P. Now by definition of $\mathrm{diam}(P)$ and $\mathcal{D}$, there is a point $x \in P$ such that the ball $B(x, \mathrm{diam}(P)) \in \mathcal{D}$. Hence $\mathrm{opt}(P,\kappa) \leq \mathrm{diam}(P)^{\alpha}$.

Let t denote the number of balls in $\mathrm{opt}(P,\kappa)$ of radius greater than or equal to $\mathrm{diam}(P)/\gamma$. By optimality of $\mathrm{opt}(P,\kappa)$, we have $t \cdot (\mathrm{diam}(P)/\gamma)^{\alpha} \leq \mathrm{diam}(P)^{\alpha}$. Thus $t \leq \gamma^{\alpha}$. ◀

Like in the case of MCC, we assume that the minimum distance between two points in X is 1. Let $L = 1 + \log(\mathrm{diam}(X))$. We fix a parameter $\lambda = \varepsilon/6L$. Let $\gamma = \frac{c \log n}{\lambda}$, where c is the constant in Theorem 1.

We design a procedure CLUSTERING(P,κ) (see Algorithm 2) that given a subset P of X and an integer κ, returns a set of at most $(1+3\lambda)^l \kappa$ balls whose union contains P, where $l = \mathrm{level}(P)$. We overview this procedure, focussing on the differences from the procedure POINT-COVER() used to solve the MCC problem. In CLUSTERING(P,κ), RAND-PARTITION(P) is called with $Z = X$ and $n = |X| \geq |P|$. We require two properties of the partition $\{P_1, \ldots, P_\tau\}$ of P computed by RAND-PARTITION(P). Let $\mathcal{Q}_0$ be the set containing the large balls of $\mathrm{opt}(P,\kappa)$, that is, those with radius at least $\mathrm{diam}(P)/\gamma$. Let $\mathcal{S}' = \mathrm{opt}(P,\kappa) \setminus \mathcal{Q}_0$ denote the set of small balls, and let $\mathcal{S}'_i \subseteq \mathcal{S}'$ consist of those balls that contain at least one point in P_i that is not covered by $\mathcal{Q}_0$. We would like (a) $\sum_{i=1}^{\tau} \mathrm{cost}(\mathcal{S}'_i) \leq (1+3\lambda)\mathrm{cost}(\mathcal{S}')$, and (b) $\sum_{i=1}^{\tau} |\mathcal{S}'_i| \leq (1+3\lambda)|\mathcal{S}'|$. Theorem 1 ensures that each of (a) and (b) holds in expectation. However, we would like both (a) and (b) to hold simultaneously, not just in expectation. For this reason, we try $\Theta(\log n)$ independent random partitions in Line 6, ensuring that with high probability, properties (a) and (b) hold for at least one of them.

Algorithm 2 CLUSTERING(P, κ)

```
Require: A subset $P \subseteq X$, an integer $\kappa$.
Ensure: A set of balls whose union contains the points in $P$.
 1: if $|P|$ is smaller than some constant $\beta$ then
 2:     return a minimum solution by checking all solutions with at most $\min\{\kappa, \beta\}$ balls.
 3: sol $\leftarrow$ the best solution with one ball
 4: cost $\leftarrow$ cost$(sol)$
 5: $l \leftarrow$ level$(P)$
 6: for all $2\log_{3/2} n$ iterations do
 7:     Let $\{P_1, \ldots, P_\tau\}$ be the set of nonempty subsets returned by RAND-PARTITION$(P)$
 8:     Let $\mathcal{B}$ be the set of balls in $\mathcal{D}$ having radius greater than $\frac{\text{diam}(P)}{\gamma}$
 9:     for each $\mathcal{Q} \subseteq \mathcal{B}$ of size at most $c(\gamma)$ do
10:         for $i = 1$ to $\tau$ do
11:             Let $P'_i = \{p \in P_i \mid p \notin \bigcup_{B \in \mathcal{Q}} B\}$
12:         for each $1 \leq i \leq \tau$ and $0 \leq \kappa_1 \leq (1{+}3\lambda)\kappa$ do
13:             cluster$(P'_i, \kappa_1) \leftarrow$ CLUSTERING$(P'_i, \kappa_1)$
14:         for $i = 0$ to $\tau - 1$ do
15:             $R_i \leftarrow \bigcup_{j=i+1}^{\tau} P'_j$
16:         for $\kappa_1 = 0$ to $(1{+}3\lambda)\kappa$ do
17:             cluster$(R_{\tau-1}, \kappa_1) \leftarrow$ cluster$(P'_\tau, \kappa_1)$
18:         for all $i = \tau - 2$ to $0$ and $0 \leq \kappa_1 \leq (1{+}3\lambda)\kappa$ do
19:             $\kappa'_{\min} \leftarrow \arg\min_{\kappa' : 0 \leq \kappa' \leq \kappa_1}$ cost(cluster$(P'_{i+1}, \kappa') \cup$ cluster$(R_{i+1}, \kappa_1 - \kappa')$)
20:             cluster$(R_i, \kappa_1) \leftarrow$ cluster$(P'_{i+1}, \kappa'_{\min}) \cup$ cluster$(R_{i+1}, \kappa_1 - \kappa'_{\min})$
21:         $\mathcal{Q}' \leftarrow \mathcal{Q} \cup$ cluster$(R_0, (1 + 3\lambda) \cdot (\kappa - |\mathcal{Q}|))$
22:         if $|\mathcal{Q}'| \leq (1{+}3\lambda)^l \kappa$ and cost$(\mathcal{Q}') <$ cost then
23:             cost $\leftarrow$ cost$(\mathcal{Q}')$
24:             sol $\leftarrow \mathcal{Q}'$
25: return sol
```

Now let us fix one of these $\Theta(\log n)$ trials where we got a partition $\{P_1, \ldots, P_\tau\}$ satisfying properties (a) and (b), and also fix an iteration in Line 9 where we have $\mathcal{Q} = \mathcal{Q}_0$. Let $P'_i \subseteq P_i$ be the points not covered by $\mathcal{Q}_0$. For each $1 \leq i \leq \tau$ and $0 \leq \kappa_1 \leq (1 + 3\lambda)\kappa$, we set cluster$(P'_i, \kappa_1)$ to be the cover obtained by recursively invoking CLUSTERING(P'_i, κ_1) (as in Line 13).

Let us call a tuple $(\kappa_1, \kappa_2, \ldots, \kappa_\tau)$ of integers *valid* if $0 \leq \kappa_i \leq (1 + 3\lambda)(\kappa - |\mathcal{Q}_0|)$ and $\sum_{i=1}^{\tau} \kappa_i \leq (1 + 3\lambda)(\kappa - |\mathcal{Q}_0|)$. We would like to minimize $\sum_{i=1}^{\tau}$ cost(cluster(P'_i, κ_i)) over all valid tuples $(\kappa_1, \kappa_2, \ldots, \kappa_\tau)$. As there are too many valid tuples to allow explicit enumeration, we solve this optimization problem in Lines 14–21 via a dynamic programming approach.

This completes our overview. Our overall algorithm for k-clustering calls the procedure CLUSTERING(X, k). Next we give the approximation bound on the cost of the solution returned by CLUSTERING(P, κ).

▶ **Lemma 6.** *For any $P \subseteq X$ and an integer $\kappa \geq 1$, CLUSTERING(P, κ) returns a solution consisting of at most $(1 + 3\lambda)^l \kappa$ balls and with probability at least $1 - \frac{|P|-1}{n^2}$, the cost of the solution is at most $(1 + 3\lambda)^l$ cost$(opt(P, \kappa))$, where $l =$ level(P).*

We refer the reader to the full version of the paper [4] for the proof of Lemma 6. Overall, it is similar to the proof of Lemma 3, and the key differences have already been anticipated in our overview.

Since $\lambda = \varepsilon/6L$, $(1+3\lambda)^L \le 1+\varepsilon$. Thus we conclude that with probability at least $1-\frac{1}{n}$, CLUSTERING(X,k) returns a solution with at most $(1+\varepsilon)k$ balls whose cost is at most $(1+\varepsilon)\text{cost}(\text{opt}(X,k))$.

Now consider the time complexity of the algorithm. CLUSTERING(P,κ) makes $n^{O(c(\gamma))}$ direct recursive calls on subsets of diameter at most diam$(P)/2$. Thus the overall time complexity of CLUSTERING(X,k) can be bounded by $n^{O(c(\gamma)L)}$. Plugging in $\lambda = \varepsilon/6L$, $\gamma = c\log n/\lambda$, and $c(\gamma) = \gamma^\alpha$, we conclude

▶ **Theorem 7.** *There is a randomized algorithm for k-clustering that runs in time $n^{O((\frac{L\log n}{\epsilon})^\alpha L)}$ and with probability at least $1-\frac{1}{n}$ returns a solution with at most $(1+\varepsilon)k$ balls whose cost is at most $(1+\varepsilon)$ times the optimal. Here L is 1 plus the logarithm of the aspect ratio of X, that is, the ratio of the maximum and minimum interpoint distances in the set X.*

5 Inapproximability Result

In this section we present an inapproximability result which complements the result in Section 3. In particular here we consider the case when α is not a constant. The heart of this result is a reduction from the dominating set problem. Given a graph $G=(V,E)$, a dominating set for G is a subset V' of V such that for any vertex $v \in V \setminus V'$, v is connected to at least one vertex of V' by an edge in E. The dominating set problem is defined as follows.

Dominating Set Problem (DSP)
INSTANCE: Graph $G=(V,E)$, positive integer $k \le |V|$.
QUESTION: Is there a dominating set for G of size at most k?

The following inapproximability result is proved by Kann [21].

▶ **Theorem 8.** *There is a constant $c > 0$ such that there is no polynomial-time $c\log|V|$-factor approximation algorithm for DSP assuming $\mathcal{P} \ne \mathcal{NP}$.*

The following theorem shows an inapproximability bound for MCC when $\alpha \ge \log|X|$.

▶ **Theorem 9.** *For $\alpha \ge \log|X|$, no polynomial time algorithm for MCC can achieve an approximation factor better than $c\log|X|$ assuming $\mathcal{P} \ne \mathcal{NP}$.*

Proof. To prove this theorem we show a reduction from DSP. Given an instance $(G=(V,E),k)$ of DSP we construct an instance of MCC. The instance of MCC consists of two sets of points X (clients) and Y (servers), and a metric d defined on $X\cup Y$. Let $V=\{v_1,v_2,\dots,v_n\}$, where $n=|V|$. For each $v_i \in V$, Y contains a point y_i and X contains a point x_i. For any point $p \in X\cup Y$, $d(p,p)=0$. For $i,j\in[n]$, $d(x_i,y_j)$ is 1 if $i=j$ or the edge $(v_i,v_j)\in E$, and $d(x_i,y_j)$ is 3 otherwise. For $i,j\in[n]$ such that $i\ne j$, we set $d(x_i,x_j)=d(y_i,y_j)=2$.

Consider two nonadjacent vertices v_i and v_j. For any $x_t \in X$ such that $t\ne i,j$, $d(x_i,x_t)+d(x_t,y_j)\ge 3$. Similarly, for any $y_t\in Y$ such that $t\ne i,j$, $d(x_i,y_t)+d(y_t,y_j)\ge 3$. Thus d defines a metric. Next we will prove that G has a dominating set of size at most k iff the cost of covering the points in X using the balls around the points in Y is at most k.

Suppose G has a dominating set J of size at most k. For each vertex $v_j\in J$, build a radius 1 ball around y_j. We return this set of balls $\mathcal{B}$ as the solution of MCC. Now consider any point $x_i\in X$. If $v_i\in J$, then x_i is covered by the ball around y_i. Otherwise, there must be a vertex $v_j\in J$ such that $(v_i,v_j)\in E$. Then $d(x_i,y_j)$ is 1 and x_i is covered by the ball around y_j. Hence $\mathcal{B}$ is a valid solution of MCC with cost at most k.

Now suppose there is a solution $\mathcal{B}$ of MCC with cost at most k. If $k > |X|$, then V is a dominating set for G of size $|X| < k$. If $k \leq |X|$, our claim is that the radius of each ball in $\mathcal{B}$ is 1. Suppose one of the balls B has a radius more than 1. Then the way the instance of MCC is created the radius should be at least 3. Hence $k \geq 3^\alpha \geq 3^{\log |X|} > |X|$, which is a contradiction. Now consider the set of vertices J corresponding to the centers of balls in $\mathcal{B}$. It is not hard to see that J is a dominating set for G of size at most k.

Let OPT be the cost of any optimal solution of MCC for the instance (X, Y, d). Then by the properties of this reduction the size of any minimum dominating set for G is OPT. Thus if there is an approximation algorithm for MCC that gives a solution with cost $(c \log |X|)\cdot$OPT, then using the reduction we can produce a dominating set of size $(c \log |V|)\cdot$OPT. Then from Theorem 8 it follows that $\mathcal{P} = \mathcal{NP}$. This completes the proof of our theorem. ◀

6 Conclusions

One generalization of the MCC problem that has been studied [10, 9] includes fixed costs for opening the servers. As input, we are given two point sets X (clients) and Y (servers), a metric on $Z = X \cup Y$, and a *facility cost* $f_y \geq 0$ for each server $y \in Y$. The goal is to find a subset $Y' \subseteq Y$, and a set of balls $\{B_y \; | y \in Y' \text{ and } B_y \text{ is centered at } y\}$ that covers X, so as to minimize $\sum_{y \in Y'} (f_y + r(B_y)^\alpha)$. It is not hard to see that our approach generalizes in a straightforward way to give a $(1 + \varepsilon)$ approximation to this problem using quasi-polynomial running time. To keep the exposition clear, we have focussed on the MCC rather than this generalization.

The main open problem that emerges from our work is whether there one can obtain a $(1 + \varepsilon)$-approximation for the k-clustering problem in quasi-polynomial time.

Acknowledgements. We would like to thank an anonymous reviewer of an earlier version of this paper for suggestions that improved the guarantees and simplified the proof of Theorem 1. We also thank other reviewers for their feedback and pointers to the literature.

References

1 Ittai Abraham, Yair Bartal, and Ofer Neimany. Advances in metric embedding theory. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 271–286. ACM, 2006.

2 Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S. B. Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the 22nd ACM Symposium on Computational Geometry, Sedona, Arizona, USA, June 5-7, 2006*, pages 449–458, 2006. doi:10.1145/1137856.1137922.

3 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC'98, pages 106–113, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276718.

4 Sayan Bandyapadhyay and Kasturi R. Varadarajan. Approximate clustering via metric partitioning. *CoRR*, abs/1507.02222, 2015. URL: http://arxiv.org/abs/1507.02222.

5 Sayan Bandyapadhyay and Kasturi R. Varadarajan. On variants of k-means clustering. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, pages 14:1–14:15, 2016. doi:10.4230/LIPIcs.SoCG.2016.14.

6 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS'96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193. IEEE Computer Society, 1996. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4141, doi:10.1109/SFCS.1996.548477.

7 Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *Algorithms–ESA 2004*, pages 89–97. Springer, 2004.

8 Babak Behsaz and Mohammad R. Salavatipour. On minimum sum of radii and diameters clustering. In *Algorithm Theory – SWAT 2012 – 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, pages 71–82, 2012. doi:10.1007/978-3-642-31155-0_7.

9 Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Algorithms – ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, pages 460–471, 2005. doi:10.1007/11561071_42.

10 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Syst. Sci.*, 68(2):417–441, 2004. doi:10.1016/j.jcss.2003.07.014.

11 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. In *FOCS, to appear*, 2016.

12 Srinivas Doddi, Madhav V. Marathe, S. S. Ravi, David Scot Taylor, and Peter Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. *Nord. J. Comput.*, 7(3):185–203, 2000.

13 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.

14 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/j.jcss.2004.04.011.

15 Ari Freund and Dror Rawitz. Combinatorial interpretations of dual fitting and primal fitting. In *Approximation and Online Algorithms, First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003, Revised Papers*, pages 137–150, 2003. doi:10.1007/978-3-540-24592-6_11.

16 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57(3):484–498, 2010. doi:10.1007/s00453-009-9282-7.

17 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On clustering to minimize the sum of radii. *SIAM J. Comput.*, 41(1):47–60, 2012. doi:10.1137/100798144.

18 Sariel Har-Peled. Geometric approximation algorithms, 2011.

19 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

20 Lior Kamma, Robert Krauthgamer, and Huy L Nguyên. Cutting corners cheaply, or how to remove steiner points. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1040. SIAM, 2014.

21 Viggo Kann. On the approximability of np-complete optimization problems. *PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm*, 1992. URL: http://www.csc.kth.se/~viggo/papers/phdthesis.pdf.

22 Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47(4):489–501, 2005. doi:10.1016/j.comnet.2004.08.012.

23 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.

24 Mohammad Salavatipour Zachary Friggstad, Mohsen Rezapour. Local search yields a ptas for k-means in doubling metrics. In *FOCS, to appear*, 2016.

Hard Communication Channels for Steganography

Sebastian Berndt[1] and Maciej Liśkiewicz[2]

1 **University of Lübeck, Lübeck, Germany**
berndt@tcs.uni-luebeck.de

2 **University of Lübeck, Lübeck, Germany**
liskiewi@tcs.uni-luebeck.de

Abstract

This paper considers steganography – the concept of hiding the presence of secret messages in legal communications – in the computational setting and its relation to cryptography. Very recently the first (non-polynomial time) steganographic protocol has been shown which, for any communication channel, is provably secure, reliable, and has nearly optimal bandwidth. The security is unconditional, i.e. it does not rely on any unproven complexity-theoretic assumption. This disproves the claim that the existence of one-way functions and access to a communication channel oracle are both necessary and sufficient conditions for the existence of secure steganography in the sense that secure and reliable steganography exists independently of the existence of one-way functions. In this paper, we prove that this equivalence also does not hold in the more realistic setting, where the stegosystem is polynomial time bounded. We prove this by constructing (a) a channel for which secure steganography exists if and only if one-way functions exist and (b) another channel such that secure steganography implies that *no* one-way functions exist. We therefore show that security-preserving reductions between cryptography and steganography need to be treated very carefully.

1998 ACM Subject Classification E.3 Data Encryption

Keywords and phrases provable secure steganography, cryptographic assumptions, pseudorandom functions, one-way functions, signature schemes

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.16

1 Introduction

Digital steganography has recently received substantial interest in modern computer science since it allows secret communication without revealing its presence. Currently, using freely available steganographic software, one party is able to spread secret messages over widely accessible services, such as photo-sharing websites, camouflaging the presence of the messages in legal communications. Although the uploads and views by other users can be recorded and analyzed it is fairly difficult to distinguish the altered documents containing a secret message from those of millions of the other ordinary documents. For more details on applied steganography see the textbook [16] or the current survey [38] and the literature therein. For applications of steganography in other areas, like covert computation, broadcasting, or anonymous communication see e.g. [6, 7, 14, 18, 24, 35].

A common computational model for secret-key steganography, also used in this paper, was introduced by Hopper, Langford, and von Ahn [21, 22, 23]. Independently, Katzenbeisser and Petitcolas [25] provided a similar formulation. In this setting, a *stegosystem* is defined as a pair of probabilistic algorithms, called encoder and decoder, which share a secret-key. The aim of the encoder (often called Alice or the steganographer) is to hide a secret message in a document and to send it to the decoder (Bob) via a public *channel* $\mathcal{C}$, which is completely

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 16; pp. 16:1–16:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

monitored by an *adversary* (Warden or steganalyst). The channel is modeled by a *cover-document* sampler that can be queried adaptively, in a black-box manner and the adversary's task is to distinguish those from altered ones called *stego-documents.*

To hide a secret message m, the encoder can take sample cover-documents, based on past communication, and manipulate them to embed m. The decoder, receiving stego-documents, should be able to decode the hidden message correctly. The stegosystem is called *reliable* if the decoder succeeds with high probability. The adversary is a probabilistic algorithm with access to additional knowledge about the channel. A stegosystem is *secure* if no adversary of *polynomial* time complexity is able to distinguish with significant probability between cover- and stego-documents generated by the stegosystem's encoder. This implies in general that the distributions of cover-documents and stego-documents have to be fairly close in a complexity-theoretic sense. The *insecurity* of a stegosystem is the advantage of the best adversary to distinguish between cover- and stego-documents. Thus, a stegosystem is secure if its insecurity is sufficiently small, i.e. negligible in the security parameter κ defining the length of the shared secret-key.

The security and reliability are necessary attributes of any reasonable stegosystem. Additionally, the system should be efficient in terms of the transmission *rate* (payload), i.e. the number of bits transmitted per single stego-document should be as high as possible. The stegosystems used in practice (not necessary *provable* secure in the computational model) typically achieve a rate of $\sqrt{n}$ [26], where $n := n(\kappa)$ denotes the length of a single document that is polynomial in κ. A longstanding conjecture, the *Square Root Law of Steganographic Capacity* [15, 27] says that a rate of the form $(1 - \varepsilon)\sqrt{n}$ is always achievable in the information-theoretic setting.

Importantly, in the definition of the computational model Hopper, Langford and von Ahn [21, 22, 23] do not bound the running time of the stegosystem, while the time complexity of the adversary is required to be bounded by a polynomial. For this setting we have shown very recently the strongest possible result; namely, that there exists a *universal* stegosystem which for *any* channel is secure, reliable and achieves almost optimal rate. Recall, that a system is called *universal*[1] if the encoding method does not rely on knowledge of the distribution for the channel $\mathcal{C}$ except that its min-entropy is sufficiently large.

▶ **Theorem 1** ([4], Informal). *There exists a universal (non-polynomial time) stegosystem $\mathcal{S}$ that is unconditionally secure and reliable. Moreover $\mathcal{S}$ is rate-efficient.*

This disproves the widely circulated result claimed in [21, 23] that the existence of one-way functions and access to a communication channel oracle are both necessary and sufficient conditions for the existence of secure steganography (see e.g. the textbook [16] for a discussion). In fact, secure and reliable (non-polynomial) steganography exists independently of the existence of one-way functions.

In this paper we investigate a more reasonable setting in which the stegosystem's running time is bounded by a polynomial and study provably secure steganography and its relation to cryptography. We prove that, despite strong connections, polynomial time steganography is *not* cryptography. More precisely we show that, similarly as in the case of non-polynomial time steganography, the equivalence between the existence of one-way functions and the existence of secure, reliable, and rate-efficient (polynomial time) steganography does not hold.

[1] In the literature universal stegosystems are also called ”black-box”.

1.1 Previous Works

As we discuss in [4], a commonly heard argument for the premise that steganography is cryptography goes as follows: Let m and m' be two different secret messages and s and s' be stego-documents which embed m, resp. m'. If the distributions of s and s' are indistinguishable from the distribution of the cover-documents, then by the triangle-inequality, the distributions of s and s' are also indistinguishable. Hence, a secure stegosystem is also a secure cryptosystem.

While the argument concerning the triangle-inequality is true, the argument ignores the channel oracle. If the channel documents are e.g. natural digital pictures, the cryptosystem simulating the stegosystem needs access to samples of those documents. But an efficient sampler for this channel seems highly unlikely. Thus, this reasoning is wrong and in fact we show in [4] that (non-polynomial time) steganography exists independently of the existence of one-way functions. Below we discuss known results in this direction.

In contrast to the non-polynomial case, *universal* steganography is very limited when requiring polynomial running time. In [10], Dedić et al. proved that for every stegosystem $\mathcal{S}$ with security parameter κ (describing the length of the secret key) which hides $\lambda := \lambda(\kappa)$ bits, takes $q := q(\kappa)$ samples per stego-document and runs in time $p := p(\kappa)$ there exists a channel $\mathcal{C}(\kappa)$ of min-entropy $\mathsf{pol}(\kappa)$ such that

$$\mathbf{InSec}(\kappa) + \mathbf{UnRel}(\kappa) \;\geq\; \frac{1}{2} - \frac{e \cdot q}{2^{\lambda}} - \Psi(p, \kappa) - o(1). \tag{1}$$

Here, $\mathbf{InSec}(\kappa)$ denotes the insecurity (against polynomial time bounded wardens) and $\mathbf{UnRel}(\kappa)$ the unreliability of $\mathcal{S}$ on $\mathcal{C}(\kappa)$, and Ψ describes a term caused by the insecurity of the pseudorandom function used in the construction of $\mathcal{C}(\kappa)$. From this result we get that if restricted to polynomial time steganography, Theorem 1 does not hold unless one-way functions do not exist:

▶ **Theorem 2** ([10], Informal statement). *Assuming one-way functions exist there exists no secure and reliable universal* polynomial time *stegosystem of rate* $\omega(\log \kappa)$.

Interestingly, the logarithmic bound on the bandwidth above is sharp. Due to Hopper et al. [23] and Dedić et al. [10] we know that the existence of one-way functions implies the existence of a secure and reliable universal (polynomial time) stegosystem of rate $\mathcal{O}(\log \kappa)$.

Theorem 2 shows a very important property, interesting in itself: when requiring polynomial time, the applicability of universal steganography is very limited. Due to this reason it makes sense to consider the security of a stegosystem $\mathcal{S}$ only for a specific channel or for channels of a specific family, and do not to require its security for all possible channels. This is also a common approach in practical steganography where a system has to satisfy security properties for a specific channel, like e.g. natural images in JPEG-format, but its security for texts, audio signals, TCP/IP transmission packages, etc. is irrelevant. For this setting the relationship between steganography and cryptography remains unsolved. Particularly, it is not known whether for any channel $\mathcal{C}(\kappa)$ there exists a secure, reliable, and rate-efficient (polynomial time) stegosystem for $\mathcal{C}(\kappa)$. The question remains open both for unconditional security and under some unproven assumptions like the existence of one-way functions.

Note that the lower bound (1) above does not allow to answer this question. To prove their result, Dedić et al. [10] show that for every (polynomial time) stegosystem $\mathcal{S}$ there exists a channel $\mathcal{C}(\kappa)$ that satisfies inequality (1). However, every channel $\mathcal{C}(\kappa)$ of [10] has a secure, reliable and rate-efficient (polynomial time) stegosystem (for a proof see e.g. [31]). Also the following lower bound provided by Hopper et al. [23] does not suffice to solve this problem. They show that for any function $q(\kappa)$ bounded by a polynomial in κ there exists a channel $\mathcal{C}(\kappa)$ such that for every (polynomial time) stegosystem $\mathcal{S}$ of query complexity $q(\kappa)$ which hides $\lambda(\kappa)$ bits per document it is true

$$\mathbf{InSec}(\kappa) + \mathbf{UnRel}(\kappa) \ \geq \ 1 - q/2^{\lambda} - 2^{-\kappa}. \tag{2}$$

In case $\lambda(\kappa) \in \omega(\log \kappa)$ the right-hand side of the inequality (2) is big, meaning that $\mathcal{S}$ is insecure or unreliable, but again in in this situation one can construct a (polynomial time) stegosystem $\mathcal{S}'$ of query complexity $q(\kappa) + 1$ that is secure, reliable and rate-efficient on $\mathcal{C}(\kappa)$.

Hence both of these lower bounds prove that every stegosystem that hides $\omega(\log \kappa)$ bits is insecure or unreliable on *some* channel from a channel family $\mathcal{F}$. On the other hand, for all of those channels, one can construct a secure and reliable stegosystem. Hence, the insecurity or unreliability of the stegosystem on those channels comes from the fact that the stegosystem must work for *all* channels in $\mathcal{F}$ and not necessarily from the complexity of a single channel.

1.2 Our Contributions

We prove that polynomial time bounded, provably secure, reliable, and rate-efficient steganography is independent of cryptographic assumptions, such as the existence of one-way functions. This is a consequence of the following results.

▶ **Theorem 3** (Informal). *Assuming one-way functions exist there exists a channel $\mathcal{C}(\kappa)$ such that for $\mathcal{C}(\kappa)$ no secure and reliable polynomial time stegosystem of rate $\omega(\log \kappa)$ is possible.*

The logarithmic bound on the bandwidth above is sharp unless one-way functions do not exist. One can conclude even more, namely that if Theorem 3 holds for rate $\mathcal{O}(\log \kappa)$, no one-way functions exists. More formally, we have the following:

▶ **Corollary 4.** *If proposition (a) is true:*

(a) *Assuming one-way functions exist there exists a channel $\mathcal{C}(\kappa)$ such that for $\mathcal{C}(\kappa)$ no secure and reliable polynomial time stegosystem of rate $\mathcal{O}(\log \kappa)$ is possible;*

then one-way functions do not exist.

To see this, again from [23] and [10] we know that: (b) *If one-way functions exist then for every channel $\mathcal{C}(\kappa)$ there exists secure and reliable polynomial time stegosystem of rate $\mathcal{O}(\log \kappa)$*. Thus, proving the proposition (a) in Corollary 4 would be possible only if one-way functions do not exist – only in this case both of the proposition (a) and (b) are true. Clearly, current research is far from proving anything like proposition (a).

Theorem 3 is the main technical achievement of this paper. We complement our result by showing a channel for which the existence of one-way functions implies the existence of a secure, reliable, and rate-efficient polynomial time stegosystem. Constructions of similar channels are known in the steganography community however, for the sake of correctness and completeness we formulate and prove a suitable result in our paper:

▶ **Theorem 5** (Informal). *There exists a channel $\mathcal{C}(\kappa)$ such that if one-way functions exist then secure, reliable, and rate-efficient polynomial time stegosystem for $\mathcal{C}(\kappa)$ exists.*

The proofs of the theorems are constructive. Interestingly, the channel $\mathcal{C}(\kappa)$ satisfying Theorem 3 is specified by a cryptographic signature scheme protocol that is widely used in practice. While $\mathcal{C}(\kappa)$ per se is artificial, its close relative, the channel of cryptographic signed emails on the internet, is widely used. In this work we prove also that there exist more such hard channels satisfying the conditions of Theorem 3. In fact we show that any channel which can express the signature scheme belongs to this family. Our construction is inspired by the technique used in the work of De et al. [9] which apply this method to show that it is not possible to uniformly generate satisfying assignments to a 3-CNF formula if one is given polynomial many samples of satisfying assignments. The channels satisfying the conditions of Theorem 5 are channels that can be sampled by an algorithm in polynomial time.

1.3 Relevant Work

The running time of universal steganography was improved by Kiayias et al. in [28] by using t-wise independent family of functions instead of a pseudorandom function to choose the corresponding documents from the channel. They also showed that a key length of $(1+o(1))n$ is sufficient to achieve information-theoretic security of $2^{-n/log^{O(1)}(n)}$ for message length n.

Van Le and Kurosawa [29] used arithmetic coding techniques to improve upon the rates of the universal systems proposed in [23] and [10]. In order to achieve this they assume that the system has access to additional knowledge on the channel. Their work thus does not fit into the model introduced by Hopper et al. [23].

Von Ahn and Hopper [36] gave the first complexity-theoretic definitions of *public-key steganography*, where the running time of the stegosystem is polynomial time bounded. Their work was extended by Backes and Cachin [2], who introduced stronger security definitions and presented a universal non-rate-efficient stegosystem for one of their definitions. Hopper [20] then proceeded by proving that every so-called efficiently sampleable channel has a non-rate-efficient stegosystem that achieves the strongest security definition.

Universal stegosystems have also been studied in the information-theoretic setting, where the information-theoretic distance between the distribution of the channel documents and the distribution of the stego-documents must be bounded. The first information-theoretic definitions of steganography were given by Cachin [5]. Wang and Moulin [37] presented a whole framework to study the optimal embedding rate of information-theoretic perfect stegosystems. For more information on this see e.g. [8, 15, 33].

The paper is organized as follows: The next section contains the basic definitions regarding stegosystems, their security and the cryptographic primitives we make use of. The proof of Theorem 3 and its extension can be found in Section 3, while Theorem 5 is proved in Section 4. Finally, we conclude our paper and discuss the future work in Section 5.

2 Preliminaries and Definitions

We say that an algorithm A has *oracle access* to a probability distribution D (denoted as A^D), if A can sample an element d according to D in unit time. The elements are sampled independently. If D is parameterized by $\rho_1, \rho_2, \ldots, \rho_k$, we write $A^{D(\rho_1,\ldots,\rho_k)}$ to describe the situation, where all of the parameters are fixed. If D is allowed to choose the parameter ρ_i itself, this is denoted by a dot, as in $A^{D(\rho_1,\ldots,\rho_{i-1},\cdot,\rho_{i+1},\ldots,\rho_k)}$. More generally, we also use dots in the parameters of an algorithm to indicate that this parameter may be chosen freely.

If one tries to hide the transfer of a secret message via unsuspicious communication, one first needs to define a model for this type of communication. This is done via the notion of a *channel* $\mathcal{C}$ on an alphabet Σ.

▶ **Definition 6.** A *channel* $\mathcal{C}$ on the alphabet Σ is a function taking an $n \in \mathbb{N}$ and a history $h \in (\Sigma^n)^*$ to a probability distribution on Σ^n, denoted by $\mathcal{C}_{h,n}$.

Note that we do not require the distributions $\mathcal{C}_{h,1}, \mathcal{C}_{h,2}, \ldots$ to be polynomial time constructible, as the typical channels in use may be of high complexity, e.g., pictures or poems.

As usual, a communication channel has a certain capacity, that is bounded by the entropy of the channel. The *min-entropy* $\mathcal{H}(D)$ of a probability distribution D is defined as $\mathcal{H}(D) := \min_{d \in \mathrm{supp}(D)}\{-\log D(d)\}$. The *min-entropy* $\mathcal{H}(\mathcal{C}_n)$ of a channel $\mathcal{C}$ with respect to $n \in \mathbb{N}$ is then defined as $\mathcal{H}(\mathcal{C}_n) = \min_h\{\mathcal{H}(\mathcal{C}_{h,n})\}$. The number of bits embeddable into a single document is bounded by $\mathcal{H}(\mathcal{C}_n)$ (see e.g. [22] for a proof).

To give a sound formal treatment, we parameterize the behaviour of all parties by the security parameter κ – the length of the secret key k. We therefore say that a function $f\colon \mathbb{N} \to [0,1]$ is *negligible*, if for every c and all sufficiently large n, it holds that $f(n) < n^{-c}$.

Informally, a stegoencoder SE has access to samples of $\mathcal{C}$ and *embeds* a message m into a sequence of documents $d_1, \ldots, d_\ell$, thereby producing a sequence $d_1^*, \ldots, d_\ell^*$. The goal of SE is that no efficient algorithm can distinguish the distributions of $d_1, \ldots, d_\ell$ and $d_1^*, \ldots, d_\ell^*$.

▶ **Definition 7.** A *stegosystem* $\mathcal{S}$ for the polynomial time constructible *message space* $\{\mathcal{M}_\kappa\}_{\kappa \in \mathbb{N}}$ with *document length* $n\colon \mathbb{N} \to \mathbb{N}$ and *output length* $\ell\colon \mathbb{N} \to \mathbb{N}$ is a pair of probabilistic, polynomial time Turing machines (PPTMs) $[SE, SD]$ with the following functionality upon security parameter κ:

- The *encoding algorithm* SE takes as input a key $k \in \{0,1\}^\kappa$, a message $m \in \mathcal{M}_\kappa$, a history h and a *state information* $s \in \{0,1\}^*$ and produces a document d and state information $s' \in \{0,1\}^*$ by having access to $\mathcal{C}_{h,n}$. By $SE^{\mathcal{C}}(k, m, h)$, we denote the outcome of:

```
Steganographic Encoding $SE^{\mathcal{C}}(k,m,h)$

$s := \lambda$; // initialize the first state as the empty string
for $i = 1, \ldots, \ell$:
    $(d_i, s) \leftarrow SE^{\mathcal{C}_{h,n(\kappa)}}(k, m, h, s)$;
    $h := hd_i$; // concatenate $h$ with the new document
return $d_1, \ldots, d_\ell$
```

  Note that SE is only allowed to get samples for the $i+1$-th document, after it produced the i-th document. For the sake of simplicity, we sometimes write $SE^{\mathcal{C}}(k, m, h)_i$ to denote the i-th document d_i.

- The *decoding algorithm* SD takes as input a key $k \in \{0,1\}^\kappa$ and a sequence of documents $d_1, \ldots, d_\ell$ and outputs a message m'.

The *sampling complexity* $q(\kappa)$ of SE is the number of calls of SE to its sampling oracle. The *transmission rate* $b(\kappa)$ is defined as $b(\kappa) := {}^{\log|\mathrm{supp}(\mathcal{M}_\kappa)|}/_{\ell(\kappa)} \le n(\kappa)$.

The key $k \in \{0,1\}^\kappa$ is shared by SE and SD before the embedding process. Clearly, SD should be able to reconstruct the original message with high probability. We say that $\mathcal{S}$ is *ρ-reliable*, if the maximum probability of an error (i.e. $SD(k, SE^{\mathcal{C}}(k, m, h)) \neq m$) is bounded by $1 - \rho(\kappa)$ for every message m and every history h. If $\mathcal{S}$ is ρ-reliable for a negligible ρ, we call $\mathcal{S}$ *reliable*. In addition to this, SE wants to embed as much information as possible into a document in order to reduce the overhead of the transmission. We say that $\mathcal{S}$ is *rate-efficient*, if there is constant $\alpha > 0$ such that $b(\kappa) \geq \mathcal{H}(\mathcal{C}_{n(\kappa)})^\alpha$ for all κ (we thus embed n^α bits per document with entropy n).

2.1 Security of a Stegosystem

A *warden* W is a PPTM that should decide whether the communication parties use steganography or not. In order to do so, W chooses a history and a message and presents this to a *challenge oracle* CH which, on key k, message m and history h outputs a sequence of $\ell(\kappa)$

documents $d_1, \ldots, d_\ell$. This sequence is either the output of the stegosystem $SE^{\mathcal{C}}(k, m, h)$ for a uniformly chosen key k or the ℓ-fold output $\mathcal{C}_\ell(m, h)$ of the channel with the distribution $\mathcal{C}_\ell(m, h) \sim (\mathcal{C}_{h,n(\kappa)}, \mathcal{C}_{hd_1,n(\kappa)}, \ldots, \mathcal{C}_{hd_1 d_2 \cdots d_{\ell-1},n(\kappa)})$.

The goal of W is now to reconstruct whether the challenge oracle is $SE^{\mathcal{C}}(k, \cdot, \cdot)$ (it outputs 0 or »Stego«) or $\mathcal{C}_\ell(\cdot, \cdot)$ (it outputs 1 or »not Stego«). More precisely, we consider the following experiment for an chosen hiddentext attack (CHA):

```
Steganographic Security $\textsc{cha}_\kappa(W, \mathcal{S}, \mathcal{C})$

1. $k \leftarrow \{0,1\}^\kappa$;
2. $b \leftarrow \{0,1\}$;
3. if $b = 0$ then $CH := SE^{\mathcal{C}}(k, \cdot, \cdot)$ else $CH := \mathcal{C}_\ell(\cdot, \cdot)$;
4. $b' \leftarrow W^{\mathcal{C}_{\cdot,n(\kappa)}, CH(\cdot,\cdot)}(1^\kappa)$; // $W$ chooses $h$ and $m$ for $CH$
5. if $b = b'$ then return 1 else return 0
```

Note that the warden has several liberties: It may choose the history for the channel oracle (the stegosystem can only work with its given history), the history submitted to the challenge oracle and the message submitted to the challenging oracle.

As W is able to chose the message (or the hiddentext), we say that the *chosen hiddentext attack (*CHA*) advantage* $\mathbf{Adv}^{\textsc{cha}}_{W,\mathcal{S},\mathcal{C}}(\kappa)$ of W on the stegosystem $\mathcal{S}$ on channel $\mathcal{C}$ is given as

$$\mathbf{Adv}^{\textsc{cha}}_{W,\mathcal{S},\mathcal{C}}(\kappa) = 2 \cdot \left|\Pr[\textsc{cha}_\kappa(W, \mathcal{S}, \mathcal{C}) = 1] - 1/2\right|,$$

where the probabilities are taken over the random choice of k and the randomness of CH, W and the channel. The *random hiddentext attack (*RHA*) advantage* $\mathbf{Adv}^{\textsc{rha}}_{W,\mathcal{S},\mathcal{C}}(\kappa)$ of a warden W is defined similarly with the difference that the messages given to the challenge oracle CH are chosen randomly instead of adversarially. This is a much weaker security requirement than CHA-security. Finally, for $\textsc{x} \in \{\textsc{cha}, \textsc{rha}\}$, the X-*insecurity* $\mathbf{InSec}^{\textsc{x}}_{\mathcal{S},\mathcal{C}}(q, t, \kappa)$ of a stegosystem $\mathcal{S}$ on the channel $\mathcal{C}$ is defined as

$$\mathbf{InSec}^{\textsc{x}}_{\mathcal{S},\mathcal{C}}(q, t, \kappa) = \max_W \{\mathbf{Adv}^{\textsc{x}}_{W,\mathcal{S},\mathcal{C}}(\kappa)\}.$$

The maximum is taken over all wardens W that make an expected number of $q(\kappa)$ queries and run in expected time $t(\kappa)$. We say that $\mathcal{S}$ is X-ϵ-*secure*, if $\mathbf{InSec}^{\textsc{x}}_{\mathcal{S},\mathcal{C}}(q, t, \kappa) \leq \epsilon(\kappa)$ for all polynomials q and t and X-*secure* if it is X-negl-secure for a negligible function negl.

2.2 Cryptographic Primitives

We recall briefly the definitions of the following three cryptographic primitives and the known relationships between them. For exact definitions see e.g. the literature quoted below.

One-Way Function. A polynomial time computable function $F \colon \{0,1\}^* \to \{0,1\}^*$ is called a *one-way function*, if every algorithm (inverter) upon input $F(x)$ fails to produce an element x' such that $F(x') = F(x)$.

Signature Scheme. A *signature scheme* $\mathcal{SIG}$ consists of a probabilistic *key-generation* algorithm G, that produces a secret key and a public key, a probabilistic *signing* algorithm S, that takes the secret key, a message and produces a signature for the message and a deterministic *verifying* algorithm V, that takes the public key and tests whether a message-signature pair is valid. An attacker either gets random valid message-signature pairs (random-message attack (RMA)) or can produce valid signatures for chosen messages (chosen-message attack (CMA)). Its goal is to produce a fresh message-signature pair.

Symmetric Encryption Scheme. A *symmetric encryption scheme* $\mathcal{SES}$ consists of an *encryption* algorithm *ENC*, which takes a secret key and a plain text and produces a cyphertext. This cyphertext can be decoded by the *decryption* algorithm *DEC* with the help of the same secret key. An attacker is given access to an oracle, which either encrypts a message chosen by the attacker (the *real* message) or gives a totally random cyphertext (the *random* message). The goal of the attacker is to distinguish those cases. We denote the *advantage* of an attacker A to distinguish real messages from random ones (ROR) on a symmetric encryption scheme $\mathcal{SES}$ with key length κ by $\mathbf{Adv}^{\text{ROR}}_{\mathcal{SES},A}(\kappa)$. Also, the probability that the decrypted message does not equal the original message must be negligible.

There is a deep connection between those primitives, as all of them are equivalent to each other. The groundbreaking works [3, 13, 17, 19, 32] imply the following:

▶ **Theorem 8** (informal). *One-Way functions exists* ⇔ *RMA-secure signature schemes exists* ⇔ *CMA-secure signature schemes exists* ⇔ *secure symmetric encryption schemes exist*

In Section 3, we construct an RMA-forger on a special signature scheme $\widehat{\mathcal{SIG}}$, that is "complete" for all signature schemes, i.e., if $\widehat{\mathcal{SIG}}$ is insecure, every signature scheme is insecure. The construction of such a complete signature scheme relies on the following theorem of Levin which states the existence of a complete one-way function $\widehat{F}$:

▶ **Theorem 9** (Levin [30]). *The function* $\widehat{F}$ *is a one-way function iff one-way functions exist.*

Combining Theorem 8 and Theorem 9, we get the following corollary needed to construct the "complete" signature scheme $\widehat{\mathcal{SIG}}$:

▶ **Corollary 10.** *The signature scheme* $\widehat{\mathcal{SIG}}$ *is RMA-secure iff one-way functions exist.*

3 A Channel $\mathcal{C}$ such that Efficient Steganography on $\mathcal{C}$ Does Imply the Non-existence of One-way Functions

The main result of this section, Corollary 14, says that for the widely used channel specified by a signature scheme protocol, secure and efficient steganography implies that one-way functions do not exist. Then we show that our construction can be generalized for more channels. We will only work with RHA-secure stegosystems in this section, as impossibility results upon this weaker notion imply the same results for CHA-secure stegosystems.

Our first technical goal is to formalize the following intuition: A secure and reliable stegosystem for a channel $\mathcal{C}$ must (a) have negligible probability of producing documents outside of $\text{supp}(\mathcal{C}_{h,n})$ and (b) be able to generate new documents out of the sampled documents. These properties have been formulated first in [10] for universal stegosystems.

We start with showing that the probability that the output of a secure stegosystem is not in the support of the channel is small (under the assumption that Warden can efficiently test whether a document belongs to the support of the channel). Before, let us introduce an auxiliary notion of a *membership-testable channel with confidence parameter* ν: We say that $\mathcal{C}$ is membership-testable with confidence parameter ν if there exists a probabilistic polynomial time algorithm, call it Test, which takes a polynomial number $\vec{x} = x_1, x_2, \ldots, x_q$ of documents such that $\mathcal{C}_{x_1 x_2 \ldots x_{i-1}}(x_i) > 0$ for every $i \geq 1$ and a document x and it either returns 1 or 0 such that the probability $\Pr_{\vec{x} \leftarrow \text{supp}(\mathcal{C}_{\varnothing,n})}[\text{Test}(\vec{x}, x) = 1]$ is $\geq 1 - \nu$, if $x \in \text{supp}(\mathcal{C}_{\vec{x},n})$ and $\leq \nu$ otherwise.

▶ **Lemma 11.** *Let* $\mathcal{S} = [SE, SD]$ *be a stegosystem for the message space* $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ *with document length* n *and output length* ℓ *for the channel* $\mathcal{C}$ *such that* $\mathcal{S}$ *is* RHA-ϵ*-secure.*

Furthermore, let $\mathcal{C}$ be membership-testable with parameter ν. Then for all $\kappa \in \mathbb{N}$, $m \in \mathrm{supp}(\mathcal{M}_\kappa)$, histories h, and all $i = 1, \ldots, \ell(\kappa)$, it holds for $d_i = SE^{\mathcal{C}}(k, m, h)_i$ that $\Pr_{k \leftarrow \{0,1\}^\kappa}[d_i \notin \mathrm{supp}(\mathcal{C}_{hd_1d_2\ldots d_{i-1},n(\kappa)})] \leq \epsilon(\kappa) + 2\nu$.

Next, we will prove that, as long as the support of $\mathcal{C}_{h,n}$ is large enough, a reliable stegosystem needs to produce non-seen examples of $\mathrm{supp}(\mathcal{C}_{h,n})$. Intuitively, we need to embed $|\mathrm{supp}(\mathcal{M}_n)| \approx 2^n$ messages (hereby creating at least 2^n different documents) while we only have access to $\mathsf{pol}(n)$ example documents. Note that for a rate-efficient polynomial time stegosystem, the term $\frac{\log|\mathrm{supp}(\mathcal{M}_\kappa)|}{\ell(\kappa)} = b(\kappa)$ is of the form κ^α for a $\alpha > 0$ and thus the term $\frac{q(\kappa)^{\ell(\kappa)}}{|\mathrm{supp}(\mathcal{M}_\kappa)|} = \frac{q(\kappa)^{\ell(\kappa)}}{2^{b(\kappa)\cdot\ell(\kappa)}} = \left(\frac{q(\kappa)}{2^{b(\kappa)}}\right)^{\ell(\kappa)} \approx \left(\frac{\mathsf{pol}(\kappa)}{2^{\kappa^\alpha}}\right)^{\ell(\kappa)}$ is negligible.

▶ **Lemma 12.** *Let $\mathcal{S} = [SE, SD]$ be a ρ-reliable stegosystem for the message space $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ with sample complexity c and output length ℓ for the channel $\mathcal{C}$. Then for every κ, the probability that the encoder SE produces a cover-document, which was not provided by the channel oracle, is at least $1 - \rho(\kappa) - \frac{q(\kappa)^{\ell(\kappa)}}{|\mathrm{supp}(\mathcal{M}_\kappa)|}$.*

The proofs of the lemmas above are similar to those presented in [10], thus we skip them.

We will now combine the two lemmas in order to construct an attacker to a signature scheme. For a signature scheme $\mathcal{SIG} = [G, S, V]$, define the channel $\mathcal{C}_{\mathcal{SIG}}$ with probability distributions $\mathcal{C}_{h,n}$ as follows: If h is the empty history $\varnothing$, the probability distribution $\mathcal{C}_{\varnothing,n}$ is the uniform distribution on all public keys generated by $G(1^n)$. If $(pk, sk) \in \mathrm{supp}(G(1^n))$, the probability distribution $\mathcal{C}_{pk,n}$ is then created by the following experiment:

```
Distribution of $\mathcal{C}_{pk,n}$
1. $m \leftarrow \mathcal{M}_n^{\mathrm{sig}}$; $\sigma \leftarrow S(sk, m)$; return $(m, \sigma)$
```

Furthermore, for every $i \geq 1$ and every series of valid (with respect to (pk, sk)) message-signature pairs $(m_1, \sigma_1), (m_2, \sigma_2) \ldots$ the distribution $\mathcal{C}_{pk(m_1,\sigma_1)(m_2,\sigma_2)\ldots(m_i,\sigma_i),n}$ is also equal to $\mathcal{C}_{pk,n}$. Note that $\mathcal{C}_{\mathcal{SIG}}$ is membership-testable with confidence parameter 0 due to the public key. A similar technique was used by Dwork et al. [12] and later by Ullman [34] in the context of differential privacy [11]. They prove that a certain class of databases exists such that any algorithm for a given set of counting queries is either not differentially private or inaccurate.

▶ **Theorem 13.** *Let $\mathcal{SIG} = [G, S, V]$ be a signature scheme. If there exists a polynomial time stegosystem $\mathcal{S} = [SE, SD]$ for $\mathcal{C}_{\mathcal{SIG}}$ for the message space $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ with rate b, output length ℓ and sampling complexity q such that $\mathcal{S}$ is* RHA-*ϵ-secure and ρ-reliable on $\mathcal{C}_{\mathcal{SIG}}$, then there exists an efficient forger on $\mathcal{SIG}$ with advantage at least $1 - \epsilon(\kappa) - \rho(\kappa) - \frac{q(\kappa)^{\ell(\kappa)}}{|\mathrm{supp}(\mathcal{M}_\kappa)|}$ for every κ.*

Combining Theorem 13 and Corollary 10 with $\widehat{\mathcal{SIG}}$, we obtain the following result that directly implies Theorem 3.

▶ **Corollary 14.** *The existence of a secure, reliable and rate-efficient polynomial time stegosystem on the channel $\mathcal{C}_{\widehat{\mathcal{SIG}}}$ implies that one-way functions do not exist.*

In the rest of this section we show that the proof of Theorem 13 can be generalized to more channels if they can express the signature scheme. Examples for such channels include satisfying assignments of 3-CNF formulas or satisfying assignments of monotone 2-CNF formulas. Our construction is inspired by the work of De et al. [9] who used a similar technique to show that it is not possible to uniformly generate satisfying assignments to a 3-CNF formula if one is given polynomial many samples of satisfying assignments.

Let $\mathcal{SIG} = [G, S, V]$ be a signature scheme and $in(\kappa)$ be an upper bound on the size of every message-signature pair constructed by the signing algorithm S on security parameter κ. Let $\boldsymbol{B}$ be a function class of Boolean functions such that there is a polynomial time invertible Levin reduction $[A, B, C]$ from CIRCUIT-SAT (see e.g. [1] for a formal definition) to $\boldsymbol{B}$. Such a reduction transforms a circuit $\mathfrak{C}$ into a function $f := A(\mathfrak{C})$ and a satisfying assignment β of $\mathfrak{C}$ into a value $x := B(\mathfrak{C}, \beta)$ with $f(x) = 1$. Furthermore, every x' with $f(x') = 1$ can be transformed into a satisfying assignment $\beta' := C(f, x')$ of $\mathfrak{C}$. Moreover let $\gamma\colon A(\text{CIRCUIT-SAT}) \to \{0,1\}^*$ be a polynomial time encoding of the functions generated by the reduction such that $red(\kappa)$ is an upper bound on $|\gamma(A(\mathfrak{C}))|$, if $\mathfrak{C}$ has $in(\kappa)$ input gates. Furthermore, let $\mathcal{C}$ be a channel with probability distributions $\mathcal{C}_{h,\kappa}$ defined as follows. For the empty history $\varnothing$, the distribution $\mathcal{C}_{\varnothing,\kappa}$ is the uniform distribution on $\gamma(A(\{\mathfrak{C} \mid \mathfrak{C} \text{ has } in(\kappa) \text{ input gates}\})) \subseteq \{0,1\}^{red(\kappa)}$. For every history $h_0 = \gamma(A(\mathfrak{C}))$ the probability distribution $\mathcal{C}_{h_0,\kappa}$ is the uniform distribution on documents $x \in \{0,1\}^{in(\kappa)}$ with $A(\mathfrak{C})(x) = 1$. Furthermore, for every $i \geq 1$ and every series of documents $x_1, x_2, \ldots \in \{0,1\}^{in(\kappa)}$ with $A(\mathfrak{C})(x_j) = 1$ for every j, the probability distribution $\mathcal{C}_{h_0 x_1 x_2 \ldots x_i,\kappa}$ is also the uniform distribution on the documents $x \in \{0,1\}^{in(\kappa)}$ with $A(\mathfrak{C})(x) = 1$. Moreover, assume $\mathcal{C}$ is membership-testable with confidence parameter ν.

▶ **Theorem 15.** *Let $\mathcal{SIG}$ be a signature scheme and let $\mathcal{C}$ be a channel as defined above. Assume $\mathcal{S}$ is a polynomial time stegosystem for the message space $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ with transmission rate b, output length ℓ and sampling complexity q for $\mathcal{C}$ such that $\mathcal{S}$ is* RHA-*ϵ-secure and ρ-reliable on $\mathcal{C}$. Then for every κ, there is a polynomial forger for $\mathcal{SIG}(\kappa)$ with advantage at least $1 - \epsilon(\kappa) - 2\nu - \rho(\kappa) - \frac{q(\kappa)^{\ell(\kappa)}}{|\operatorname{supp}(\mathcal{M}_\kappa)|}$.*

4 A Channel $\mathcal{C}$ such that Efficient Steganography on $\mathcal{C}$ Does Imply the Existence of One-way Functions

We will now show a channel $\mathcal{C}$ such that secure and reliable steganography on it implies the existence of one-way functions (this will follow from the theorem below and Theorem 8). The channel is assumed to be *efficiently sampleable*, i.e. such for which a polynomial time algorithm simulating sampling from $\mathcal{C}$ exists. Then a straightforward argument implies the following equivalences between steganography and cryptography.

▶ **Theorem 16.** *Let $\mathcal{C}$ be a channel with $\mathcal{C}_{h,n} = \mathcal{C}_{h',n} := \mathcal{C}_n$ for all histories h, h' and assume $\mathcal{C}$ is efficiently sampleable. If there exists a secure, reliable, and rate-efficient (polynomial time) stegosystem $\mathcal{S} = [SE, SD]$ for the channel $\mathcal{C}$ with message space $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$, then there exists a secure symmetric encryption scheme $\mathcal{SES}$ for the plaintexts $\{\mathcal{M}_n^{\text{plain}}\}_{n\in\mathbb{N}}$ with $\mathcal{M}_n^{\text{plain}} = \mathcal{M}_n$ and cyphertexts $\{\mathcal{M}_n^{\text{cypher}}\}_{n\in\mathbb{N}}$ with $\mathcal{M}_\kappa^{\text{cypher}} = \mathcal{C}_{n(\kappa)}^{\ell(\kappa)}$.*

▶ **Theorem 17.** *Let $\mathcal{SES}$ be a secure symmetric encryption scheme on plaintexts $\{\mathcal{M}_n^{\text{plain}}\}_{n\in\mathbb{N}}$ and cyphertexts $\{\mathcal{M}_n^{\text{cypher}}\}_{n\in\mathbb{N}}$. Let $\mathcal{C}$ be a channel with the documents $\operatorname{supp}(\mathcal{M}_n^{\text{cypher}})$ and $\mathcal{C}_{h,n} = \mathcal{M}_n^{\text{cypher}}$ for every h. There exists a secure, reliable, and rate-efficient (polynomial time) stegosystem $\mathcal{S}$ for $\mathcal{C}$ with message space $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ with $\mathcal{M}_n = \mathcal{M}_n^{\text{plain}}$.*

Thus, reasonable steganography on e.g. the channel $\mathcal{C}_n$ that is the uniform distribution on $\{0,1\}^n$, is equivalent to the existence of one-way functions. This proves Theorem 5.

5 Conclusions and Further Work

We have proved that steganography and cryptography are somehow orthogonal to each other. To show this statement, we constructed a specific channel based upon secure signature schemes and proved that every rate-efficient stegosystem on this channels breaks the security of the signature scheme. By using a universal one-way function due to Levin, we were then able to show that the existence of such a rate-efficient stegosystem implies that one-way functions do not exist. This is a generalization of the result of Dedić et al. [10], who only proved the existence of a *family* of channels $\mathcal{F}$ such that the existence of a rate-efficient stegosystem that works for *every* channel in $\mathcal{F}$ implies the non-existence of one-way functions. We thus proved that there is a channel $\mathcal{C}_1$ such that rate-efficient steganography on $\mathcal{C}_1$ implies the non-existence of one-way functions. On the other hand, we also gave a simple channel $\mathcal{C}_2$ and proved that rate-efficient steganography on $\mathcal{C}_2$ implies the existence of one-way functions.

The existence of those channels thus implies that statements of the form "Steganography is Cryptography" or "Steganography implies Cryptography" are wrong in this universality. Furthermore, it proves that the communication channel is a fundamental object in steganography and can not be ignored. In order to explore the fascinating connection between steganography and cryptography, it would be interesting to broaden our understanding of the influence of the communication channels. The work of Liśkiewicz et al. [31] already showed that knowledge or ignorance about some aspect of the channels has a significant impact on the steganographic setting.

References

1 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

2 Michael Backes and Christian Cachin. Public-key steganography with active attacks. In *Theory of Cryptography Conference (TCC)*, pages 210–226. Springer, 2005.

3 Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science (FOCS)*, pages 394–403. IEEE, 1997.

4 Sebastian Berndt and Maciej Liśkiewicz. Provable secure universal steganography of optimal rate. In *Information Hiding and Multimedia Security (IH&MMSec)*. ACM, 2016.

5 Christian Cachin. An information-theoretic model for steganography. *Information and Computation*, 192(1):41–56, 2004. `doi:10.1016/j.ic.2004.02.003`.

6 Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multi-party computation. In *Foundations of Computer Science (FOCS)*, pages 238–248. IEEE, 2007.

7 Chongwon Cho, Dana Dachman-Soled, and Stanisław Jarecki. Efficient concurrent covert computation of string equality and set intersection. In *Topics in Cryptology-CT-RSA*, pages 164–179. Springer, 2016.

8 Pedro Comesaña and Fernando Pérez-González. On the capacity of stegosystems. In *Multimedia & Security (MMSec)*, pages 15–24. ACM, 2007.

9 Anindya De, Ilias Diakonikolas, and Rocco A. Servedio. Learning from satisfying assignments. In *Symp. on Discrete Algorithms (SODA)*, pages 478–497. ACM-SIAM, 2015.

10 Nenad Dedić, Gene Itkis, Leonid Reyzin, and Scott Russell. Upper and lower bounds on black-box steganography. *Journal of Cryptology*, 22(3):365–394, 2009.

11 Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming (ICALP)*, pages 1–12. Springer, 2006.

12 Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Symp. on Theory of Computing (STOC)*, pages 381–390, 2009.

13 Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

14 Nelly Fazio, Antonio R Nicolosi, and Irippuge Milinda Perera. Broadcast steganography. In *Topics in Cryptology–CT-RSA 2014*, pages 64–84. Springer, 2014.

15 Tomás Filler, Andrew D. Ker, and Jessica J. Fridrich. The square root law of steganographic capacity for markov covers. In *Media Forensics and Security I, part of the IS&T-SPIE Electronic Imaging Symposium*, page 725408, 2009.

16 Jessica Fridrich. *Steganography in digital media: principles, algorithms, and applications.* Cambridge University Press, 2009.

17 Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

18 Vipul Goyal and Abhishek Jain. On the round complexity of covert computation. In *Symp. on Theory of Computing (STOC)*, pages 191–200. ACM, 2010.

19 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

20 Nicholas Hopper. On steganographic chosen covertext security. In *Automata, Languages and Programming (ICALP)*, volume 3580, pages 311–323. Springer, 2005.

21 Nicholas J Hopper. Toward a theory of steganography. Technical report, Technical Report CMU-CS-04-157, Carnegie Mellon Univ., 2004.

22 Nicholas J. Hopper, John Langford, and Luis von Ahn. Provably secure steganography. In *Advances in Cryptology (CRYPTO)*, pages 77–92. Springer, 2002. doi:10.1007/3-540-45708-9_6.

23 Nicholas J. Hopper, Luis von Ahn, and John Langford. Provably secure steganography. *Computers, IEEE Transactions on*, 58(5):662–676, 2009.

24 Sune K Jakobsen and Claudio Orlandi. How to bootstrap anonymous communication. In *Conf. on Innovations in Theoretical Computer Science (ITCS)*, pages 333–344. ACM, 2016.

25 Stefan Katzenbeisser and Fabien A.P. Petitcolas. Defining security in steganographic systems. In *Electronic Imaging 2002*, pages 50–56. SPIE, 2002.

26 Andrew D. Ker, Patrick Bas, Rainer Böhme, Rémi Cogranne, Scott Craver, Tomáš Filler, Jessica Fridrich, and Tomáš Pevnỳ. Moving steganography and steganalysis from the laboratory into the real world. In *Information Hiding and Multimedia Security (IH&MMSec)*, pages 45–58. ACM, 2013.

27 Andrew D. Ker, Tomás Pevný, Jan Kodovský, and Jessica J. Fridrich. The square root law of steganographic capacity. In *Multimedia Security (MMSec)*, pages 107–116. ACM, 2008.

28 Aggelos Kiayias, Alexander Russell, and Narasimha Shashidhar. Key-efficient steganography. In *Information Hiding (IH)*, pages 142–159. Springer, 2012.

29 Tri Van Le and Kaoru Kurosawa. Bandwidth optimal steganography secure against adaptive chosen stegotext attacks. In *Information Hiding (IH)*, pages 297–313. Springer, 2006.

30 Leonid A. Levin. One way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

31 Maciej Liśkiewicz, Rüdiger Reischuk, and Ulrich Wölfel. Security levels in steganography – insecurity does not imply detectability. *Electronic Colloquium on Computational Complexity (ECCC)*, 22(10), 2015.

32 Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

33 Boris Ryabko and Daniil Ryabko. Constructing perfect steganographic systems. *Information and Computation*, 209(9):1223–1230, 2011.

34 Jonathan Ullman. Answering $n^{2+o(1)}$ counting queries with differential privacy is hard. In *Symp. on Theory of Computing (STOC)*, pages 361–370. ACM, 2013.

35 Luis von Ahn, Nicholas Hopper, and John Langford. Covert two-party computation. In *Symposium on Theory of Computing (STOC)*, pages 513–522. ACM, 2005.

36 Luis von Ahn and Nicholas J Hopper. Public-key steganography. In *Advances in Cryptology (EUROCRYPT)*, pages 323–341. Springer, 2004.

37 Ying Wang and Pierre Moulin. Perfectly secure steganography: Capacity, error exponents, and code constructions. *Information Theory, IEEE Transactions on*, 54(6):2706–2722, 2008.

38 Elżbieta Zielińska, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Trends in steganography. *Communications of the ACM*, 57(3):86–95, 2014.

On r-Guarding Thin Orthogonal Polygons[*]

Therese Biedl [1] and Saeed Mehrabi[2]

[1] **Cheriton School of Computer Science, University of Waterloo, Canada**
`biedl@cs.uwaterloo.ca`

[2] **Cheriton School of Computer Science, University of Waterloo, Canada**
`smehrabi@uwaterloo.ca`

Abstract

Guarding a polygon with few guards is an old and well-studied problem in computational geometry. Here we consider the following variant: We assume that the polygon is orthogonal and *thin* in some sense, and we consider a point p to guard a point q if and only if the minimum axis-aligned rectangle spanned by p and q is inside the polygon.

A simple proof shows that this problem is NP-hard on orthogonal polygons with holes, even if the polygon is thin. If there are no holes, then a thin polygon becomes a *tree* polygon in the sense that the so-called dual graph of the polygon is a tree. It was known that finding the minimum set of r-guards is polynomial for tree polygons (and in fact for all orthogonal polygons), but the run-time was $\tilde{O}(n^{17})$. We show here that with a different approach one can find the minimum set of r-guards can be found in tree polygons in linear time, answering a question posed by Biedl et al. (SoCG 2011). Furthermore, the approach is much more general, allowing to specify subsets of points to guard and guards to use, and it generalizes to polygons with h holes or thickness K, becoming fixed-parameter tractable in $h + K$.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Art Gallery Problem, Orthogonal Polygons, r-Guarding, Treewidth, Fixed-parameter Tractable

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.17

1 Introduction

The art gallery problem is one of the oldest problems studied in computational geometry. In the standard art gallery, introduced by Klee in 1973 [21], the objective is to observe a simple polygon P in the plane with the minimum number of point guards, where a point $p \in P$ is seen by a guard if the line segment connecting p to the guard lies entirely inside the polygon. Chvátal [4] proved that $\lfloor n/3 \rfloor$ point guards are always sufficient and sometimes necessary to guard a simple polygon with n vertices. The art gallery problem is known to be NP-hard on arbitrary polygons [18] and orthogonal polygons [24]. Even severely restricting the shape of the polygon does not help: the problem remains NP-hard for simple monotone polygons [17] and for orthogonal tree polygons (defined precisely below) if guards must be at vertices [26]. Further, the art gallery problem is APX-hard on simple polygons [9], but some approximation algorithms have been developed [12, 17]. A number of other types of guards have been studied, especially for orthogonal polygons. See for example guarding with sliding cameras [15, 8], guarding with rectangles [10] or with orthogonally convex polygons [20]. Also, different types of visibility have been studied, especially for orthogonal polygons: guards

[*] This work was supported by Natural Sciences and Engineering Research Council of Canada (NSERC).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 17; pp. 17:1–17:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

could be only seeing along horizontal or vertical lines inside P, or along an orthogonal staircase path inside P [20], or use r-visibility (defined below).

Definitions and Model. Let P be an orthogonal polygon with n vertices. The *pixelation* of P (also called *dent diagram* [6] and related to a *rectangleomino* [1]) is the partition of P obtained by extending a horizontal and a vertical ray inward at every reflex vertex, and expand it until it hits the boundary. Let Ψ be the resulting set of rectangles that we call *pixels* (also called *basic regions* [27]). See Figure 1 for an example. Note that $|\Psi|$ could be quadratic in general. We will sometimes interpret the pixelation as a planar graph, with one vertex at every corner of a pixel and an edge for each side of a pixel. Define the *dual graph* D of a polygon P to be the weak dual graph of the pixelation of P, i.e., D has a vertex for every pixel and two pixels are adjacent in D if and only if they have a common side.

An orthogonal polygon P is called a *thin polygon* if any pixel-corner lies on the boundary of P. It is called a *tree polygon* if its dual graph is a tree. One can easily see that a tree polygon is the same as a thin polygon that has no holes (see also Lemma 9). For most of this paper, polygons are assumed to be thin polygons.

We say that point g *r-guards* a point p if the minimum axis-aligned rectangle $R(g, p)$ containing g and p is a subset of P. The (standard) *rGuarding problem* hence consists of finding a minimum set S of points such that any point in P is r-guarded by a point in S. However, our results work for a broader problem as follows. Let $U \subseteq P$ be the region that we wish to guard. In particular, we could choose to guard only the vertices of P, or only the boundary, or only those parts of the art gallery that truly need to be watched. Let Γ be the set of guards that are allowed to be used (in particular, we could choose to use only vertices as guards). In the standard problem, Γ is the set of all points in P. Biedl et al. [1] introduced *pixel-guards*, where one guard consists of all the points that belong to one pixel (see Figure 1). Our approach allows pixel-guards, so $\Gamma \subset P \cup \Psi$. Now the (U, Γ, P)-*rGuarding problem* consists of finding a minimum set S of guards in Γ such that all of U is r-guarded by some guard in S (or to report that no such set exists).

Restricting the region that needs to be guarded exacerbates some degeneracy-issues for r-guarding. Previous papers were silent about what happens if rectangle $R(g, p)$ (in the definition of r-guarding) is a line segment. For example, in Figure 1, does g guard u_4? Does u_1 guard u_4? This issue can be avoided by assuming that only the interior of pixels must be guarded (as seems to have been done by Keil and Worman [27], e.g. their Lemma 1 is false for point u_4 located in the pixel ψ_{10} in Figure 1, because u_4 sees $q \in P$ but not all points in ψ_{10} do). When the entire polygon needs to be guarded, then this is a reasonable restriction since the guards that see the interior also see the boundary in the limit. But if only a subset of P must be guarded, then we must clarify how degeneracies are to be handled. We say that an axis-aligned rectangle R is *degenerate* if it has area 0 (i.e., is a line segment) and there exists no rectangle R' with positive area and $R \subset R' \subseteq P$. In Figure 1, $R(g, u_4)$ is degenerate while $R(u_1, u_4)$ is not. Our approach is broad enough that it can handle both allowing and disallowing the use of degenerate rectangles when defining r-guarding.

Related Results. The problem of guarding orthogonal polygons using r-guards was introduced by Keil [16] in 1986. He gave an $O(n^2)$-time exact algorithm for the rGuarding problem for horizontally convex orthogonal polygons. The complexity of rGuarding in simple orthogonal polygons was a long-standing open problem until 2007 when Worman and Keil [27] gave a polynomial-time algorithm for it. However, the algorithm by Worman and Keil is quite slow: it runs in $\tilde{O}(n^{17})$-time, where n denotes the number of the vertices of P and $\tilde{O}$

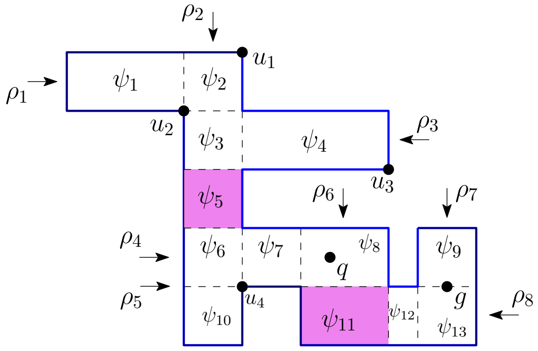


Figure 1 A tree polygon with pixels $\{\psi_1, \ldots, \psi_{13}\}$ and maximal axis-aligned rectangles $\{\rho_1, \ldots, \rho_8\}$; rectangle ρ_5 is degenerate. Pixel-guard ψ_5 guards u_3 via its top-right corner.

hides a poly-logarithmic factor. As such, Lingas et al. [19] gave a linear-time 3-approximation algorithm for rGuarding in simple polygons. Faster exact algorithms are known for a number of special cases of orthogonal polygons [16, 6, 22]. All these algorithms require the polygon to be simple. We are not aware of any results concerning the rGuarding problem for polygons with holes, or if only the vertices or only the boundary need to be guarded or used as guards.

The first results on guarding thin polygons were (to our knowledge) in [1]; they studied guarding pixelations and asked whether this can be done more easily if the dual graph is a tree. However, no better results than applying [27] were found. Later, Tomas [26] showed that indeed guarding tree polygons[1] is NP-hard in the traditional guarding-model (i.e. g guards p if the line segment gp is in P), and if all guards must be at vertices. The complexity of guarding thin polygons in the r-guarding model remained open. Paper [1] was also (apparently) the first paper to consider pixel-guards in place of point-guards.

Our Results. In this paper, we resolve the complexity of the rGuarding problem on thin polygons. We show with a simple reduction from Vertex Cover in planar graphs that this problem is NP-hard on polygons with holes, even if the polygon is thin. As our main result, we show that the rGuarding problem is linear-time solvable on thin polygons without holes.

Comparing our results to the one by Worman and Keil [27], their algorithm works for a broader class of polygons (they do not require thinness), but is slower. Moreover, their approach crucially needs that the polygon is simple, that the entire polygon needs to be guarded, and that any point in the polygon can guard. In contrast to this, our approach generalizes easily to a number of other scenarios. First of all, it is not crucial that the polygon is simple; we can deal with any constant number h of holes. Secondly, we can choose what has to be guarded and what to guard with; we can hence also solve all art gallery variants where only the vertices or only the boundary need to be guarded, or where only guards at the vertices or the boundary are allowed to be used. Finally, the restriction on thinness can be relaxed. We use thinness only to bound the treewidth of the dual graph of the polygon, and as long as the treewidth is bounded the approach works. In particular, if the polygon is K-thin in some sense, and has at most h holes, then for constants h and K our algorithm is still linear, and the rGuarding problem hence is fixed-parameter tractable in $h + K$.

[1] Tomas constructs only simple polygons and hence used the term “thin polygon” for tree polygons.

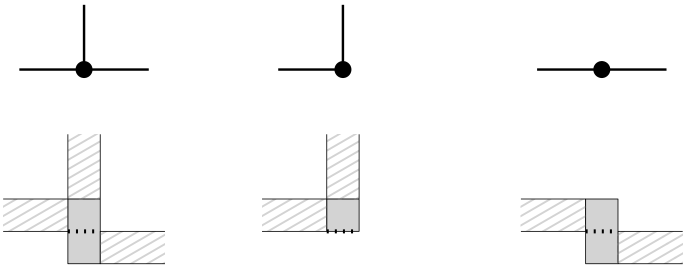


Figure 2 Converting an orthogonal drawing without bends into a polygon for rGuarding. R_e is hatched, R_v is gray, and s_v is dotted.

2 NP-hardness

In this section, we prove that rGuarding is NP-hard in polygons with holes. The reduction is from Vertex Cover in planar graphs with maximum degree 3; it is well-known that this is NP-hard [11]. So let $G = (V, E)$ be a planar graph with maximum degree 3. Let G^s be the graph obtained from G by subdividing every edge twice. It is folklore (see e.g. [23]) that G has a vertex cover of size k if and only if G^s has a vertex cover of size $|E| + k$. G has a planar orthogonal drawing with at most one bend per edge (see e.g. [14]). By placing one subdivision vertex of each edge at such a bend (if any) and placing the other subdivision vertex arbitrarily, we hence obtain a drawing Γ of G^s where every vertex is a point, every edge is a horizontal or vertical line segment, and edges are disjoint except at common endpoints.

We construct a polygon P as a "thickened" version of Γ. After possible scaling, we may assume that Γ resides in an integer grid with consecutive grid-lines at least $2n$ units apart, where $n = |V|$. Replace each horizontal edge e by a rectangle R_e of unit height, spanning between the points corresponding to the ends of e. Similarly replace each vertical edge by a rectangle of unit width. These rectangles will get moved later, but never so far that they would overlap edge-rectangles from other rows or columns.

We replace vertex-points by small gadgets as illustrated in Figure 2. Thus, let v be a vertex of degree 3 in G^s; up to rotation it has incident edges e_1, e_2, e_3 on the left, right and top in Γ. Replace v by two adjacent pixels, one above the other; we denote the resulting gadget by R_v. Then, attach R_{e_3} at the top of the upper pixel, R_{e_1} at the left side of the upper pixel and R_{e_2} at the right side of the lower pixel. Let s_v be the side common to the two pixels of R_v. Rectangles R_{e_1} and R_{e_2} are not quite horizontally aligned, resulting in one of them being offset from the grid-line. However, in total over all vertices in the row, there are at most n offsets, and so edge-rectangles remain disjoint. For any vertex of degree 2, omit the third rectangle and also any pixel that is not needed.

▶ **Observation 1.** *For any vertex v, any point in s_v guards the rectangles R_e of any incident edge $e = (v, w)$, as well as the pixel of w where R_e attaches. Moreover, for any edge $e = (v, w)$, if any point in R_e is r-guarded from a point q, then q belongs to R_e, R_v or R_w.*

Using this observation, the reduction is immediate. Namely, let C be a vertex cover of G_s of size k. For any $v \in C$, place a guard anywhere along s_v. Since C was a vertex cover, this r-guards R_e for all edges, and also R_w for all $w \notin C$ since each pixel of R_w is attached to some R_e. Vice versa, if we have a set S of r-guards, then we can create a set C as follows: For any vertex v, if R_v contains a guard in S, then add v to C. For any edge $e = (v, w)$, if R_e contains a guard in S that is in neither R_v nor R_w, then arbitrarily add one of v, w to C. Clearly $|C| \leq |S|$, and since any rectangle R_e was guarded, any edge in E is covered by C.

Inspection of Figure 2 shows that the constructed polygon is thin. Observe that it has holes, namely, one per face of G. Since rGuarding is clearly in NP, we can conclude:

▶ **Theorem 2.** *The rGuarding problem is* NP*-complete on thin polygons.*

3 Polygons Whose Dual Has Bounded Treewidth

We now show how to solve the rGuarding problem in a tree polygon in linear time. In fact, we show something stronger, and prove that the rGuarding problem can be solved in linear time in any polygon for which the dual graph D has bounded treewidth, and under any restriction on the set U to be guarded and the set Γ that may serve as guards.

The approach is to construct an auxiliary graph H, and argue that solving the rGuarding problem reduces to a graph problem in H. Then we argue that the treewidth of H satisfies $tw(H) \in O(tw(D))$ and that the graph problem is linear-time solvable in bounded treewidth graphs. This auxiliary graph is different from the so-called region-visibility-graph used by Worman and Keil [27] in that it encodes who can guard what, rather than who can be guarded by a common guard.

3.1 Simplifying U and Γ

We first show that we can simplify the points to guard and the point-guards to use such that only a constant number of each occur at each pixel.

▶ **Lemma 3.** *Let $U \subseteq P$ be any (possibly infinite) set of points in P. Then there exists a finite set of points $U' \subseteq U$ such that U' is r-guarded by a set S if and only if U is. Moreover, for any pixel ψ, at most 4 points in U' belong to ψ.*

Proof. We construct the set U' as follows.

- For every pixel ψ, if the interior of ψ intersects U, then add one point from this intersection into U'.
- For every pixel-side e, if neither incident pixel has a point of U in its interior, but the open set e intersects U, then add one point from this intersection to U',
- For every pixel-corner c, if $c \in U$, and if none of the incident pixels or pixel-sides has added a point to U', then add c to U'.

Correctness can be shown easily (see the full version [2]), by arguing that any two points in the strict interior of one pixel ψ are guarded by the same set of guards, and similarly for points in the relative interior of a side of a pixel. ◀

▶ **Lemma 4.** *Let $\Gamma \subseteq P$ be any (possibly infinite) set of points in P. Then there exists a finite set of points $\Gamma' \subseteq \Gamma$ such that for any pixel ψ, at most 4 points in Γ' belong to ψ. Moreover, if some set $S \subseteq \Gamma$ r-guards a set $U \subseteq P$, then there exists a set $S' \subseteq \Gamma'$ with $|S'| \leq |S|$ that also r-guards U.*

Proof. We construct the set Γ' follows:

- For every pixel-corner c, if $c \in \Gamma$ then add c to Γ'.
- For every pixel-side e, if neither endpoint of e is in Γ, but some interior point of e is in Γ, then add one such point to Γ'.
- Finally, for every pixel ψ, if no corner is in Γ and no side has a point in Γ, but the interior of ψ contains points in Γ, then add one such point to Γ'.

Correctness can be shown easily (see the full version [2]). ◀

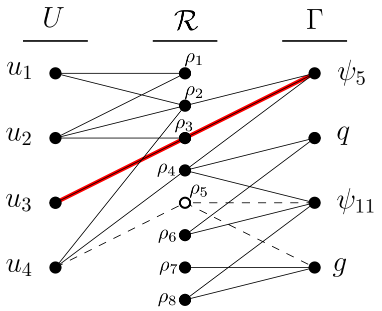


Figure 3 The graph H corresponding to Figure 1 for the chosen U and Γ. The thick red path corresponds to the pixel-guard ψ_5 seeing the point u_3 since both intersect rectangle ρ_3. Rectangle ρ_5 and its incident edges are included in H only if we allow degenerate rectangles.

3.2 Maximal Rectangles and an Auxiliary Graph

Assume we are given a polygon P, a region $U \subseteq P$ to be guarded, and a set Γ of guards allowed to be used. In what follows, we treat any element $\gamma \in \Gamma$ as a set, so either $\gamma = \psi$ is a pixel-guard or $\gamma = \{p\}$ is a point-guard.

As a first step, apply Lemmas 3 and 4 to reduce U and the point-guards in Γ so that they are finite sets, each pixel contains at most 4 points of U, and at most 4 point-guards of Γ.

Let $\mathcal{R}$ be the set of maximal axis-aligned rectangles in P, i.e., $\rho \in \mathcal{R}$ if and only if $\rho \subseteq P$ and there is no axis-aligned rectangle ρ' with $\rho \subset \rho' \subseteq P$. In this definition of $\mathcal{R}$, we use the one that was meant for r-guarding, i.e., we include degenerate rectangles in $\mathcal{R}$ if and only if a degenerate rectangle $R(g, p)$ is sufficient for g to r-guard p. Now define graph H as follows. The vertices of H are $U \cup \mathcal{R} \cup \Gamma$, i.e., we have one vertex for every point that needs guarding, one for every maximal rectangle in P, and one for every potential guard. We define edges of H via containment as follows (see also Figure 3).

(i) There is an edge from a point $u \in U$ to a rectangle $\rho \in \mathcal{R}$ if and only if $u \in \rho$.

(ii) There is an edge from a potential guard $\gamma \in \Gamma$ to a rectangle $\rho \in \mathcal{R}$ if and only if their intersection is non-empty.

▶ **Lemma 5.** *A point $u \in U$ is r-guarded by $\gamma \in \Gamma$ if and only if there exists a path of length 2 from u to γ in H.*

Proof. If u is r-guarded by γ, then there exists some $g \in \gamma$ such that the axis-aligned rectangle R spanned by p and g is inside P. Expand R until it is maximal to obtain $\rho \in \mathcal{R}$. More precisely, if R is non-degenerate, then use as ρ some maximal rectangle that has non-zero area and contains R. If R is degenerate, then obviously degenerate rectangles were allowed for r-guarding, and so expanding R into a maximal line segment within P gives an element ρ of $\mathcal{R}$. Either way $u \in R \subseteq \rho$ and $g \in R \subseteq \rho$ and we have a path $u - \rho - g$ in H.

Vice versa, if there exists such a path, then it must have the form $u - \rho - \gamma$ for some maximal rectangle ρ by construction of H. By definition of the edges, $u \in \rho$ and some point $g \in \gamma$ satisfies $g \in \rho$, which means that the axis-aligned rectangle spanned by u and g is inside $\rho \subseteq P$ and so g (and with it γ) guards u. ◀

So, the rGuarding problem reduces to finding the minimum subset $S \subseteq \Gamma$ such that all $u \in U$ have a path of length 2 to some $\gamma \in S$, or reporting that no such S exists. We call this the *restricted distance-2-dominating set* since this is the distance-2-dominating set [25] with restrictions on who can be chosen and who must be dominated. Therefore, we have:

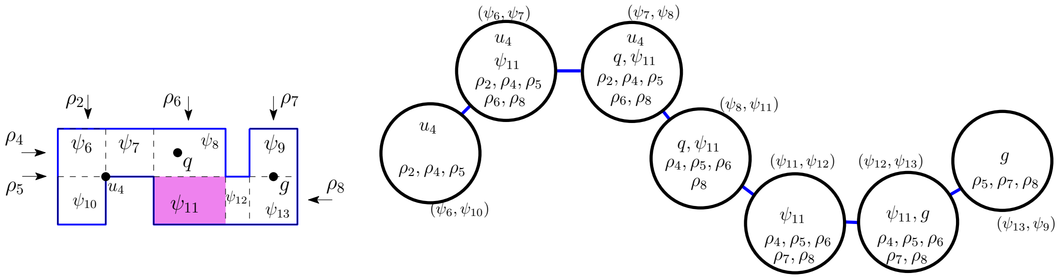


Figure 4 The tree decomposition $\mathcal{T}^H = (I, \mathcal{X}^H)$ of graph H corresponding to a sub-polygon of the one in Figure 1. We label the bags with the edges of the tree they correspond to.

▶ **Lemma 6.** *The (U, Γ, P)-rGuarding problem has a solution of size k if and only if the restricted distance-2-dominating set in H has a solution of size k.*

3.3 Constructing a Tree Decomposition

Recall graph D, the weak dual graph of the pixelation of polygon P. Assume now that the dual graph D has small treewidth, defined as follows. A *tree decomposition* of a graph D consists of a tree I and an assignment $\mathcal{X} : I \to 2^{V(D)}$ of *bags* to the nodes of I such that (a) for any vertex v of D, the bags containing v form a connected subtree of I and (b) for any edge (v, w) of D, some bag contains both v and w. The width of such a decomposition is $\max_{X \in \mathcal{X}} |X| - 1$, and the *treewidth* $tw(D)$ of D is the minimum width over all tree decompositions of D.

Fix a tree decomposition $\mathcal{T} = (I, \mathcal{X})$ of D that has width $tw(D)$. We now construct a tree decomposition of H from $\mathcal{T}$ while increasing the bag-size by a constant factor. Any bag $X \in \mathcal{X}$ consists of vertices of D, i.e., pixels of P. To obtain $\mathcal{T}' = (I, \mathcal{X}')$, modify any bag $X \in \mathcal{X}$ to get X' as follows: For any pixel $\psi \in X$, add to X'

- any point of U that is in ψ,
- any guard of Γ that intersects ψ, and
- any rectangle in $\mathcal{R}$ that intersects ψ.

Finally we may (optionally) delete all pixels from all bags, since these are not vertices of H. We call the final construction $\mathcal{T}^H = (I, \mathcal{X}^H)$. See also Figure 4.

▶ **Lemma 7.** *For any polygon, $\mathcal{T}^H = (I, \mathcal{X}^H)$ is a tree decomposition of H. If P is thin, then the tree decomposition has width $O(tw(D))$.*

Proof. First we argue that for any vertex of H the bags containing it are connected. Crucial for this is that for any pixel ψ, the bags that used to contain ψ in $\mathcal{T}$ are a connected subtree since $\mathcal{T}$ was a tree decomposition. First consider a point p. (We use p for both the point and for the vertex in H representing it.) Vertex p was added to all bags that contained a pixel ψ with $p \in \psi$. There may be multiple such pixels (if p is on the side or the corner of a pixel), but the union of them is a connected subgraph of D. For any connected subgraph, the bags containing vertices of it form a connected subtree. So the bags to which p has been added form a connected subtree of the tree I of the tree decomposition as required.

The connectivity-argument is identical for a point-guard, and similar for pixel-guards and rectangles. Namely, consider a vertex of H representing a pixel-guard γ. This guard was added to all the bags that contained a pixel ψ that intersects γ. Again there may be many such pixels (up to 9), but they are connected via ψ and so the bags to which γ is added are connected. Finally, consider a rectangle $\rho \in \mathcal{R}$ which was added to all bags of pixels intersecting ρ. The pixels that ρ intersects form a connected subset of P (because they are connected along ρ), and hence correspond to a connected subgraph of D. So the bags containing ρ form a connected subtree. Now we must verify that for any edge of H, both endpoints appear in a bag. Let (u, ρ) be an edge from some point u to some rectangle ρ. Let ψ be a pixel containing u. Then $\rho \cap \psi \supseteq \{u\}$ is non-empty and so ρ was added to any bag containing ψ. We also added u to any bag containing ψ, so u and ρ appear in one bag. Now consider some edge (γ, ρ) from a guard γ to some rectangle ρ. This edge exists because some point $g \in \gamma$ belongs to ρ. Again fix some pixel ψ that contains g and observe that any bag that contained ψ has both g and ρ added to it.

It remains to discuss the width of the tree decomposition. Consider a bag X of $\mathcal{T}$ and one pixel ψ in X. Since we reduced U and Γ with Lemma 3 and 4, pixel ψ intersects at most 4 points in U and at most 4 point-guards. It also intersects at most 9 pixel-guards. Finally, one can show that in a thin polygon ψ intersects at most 6 maximal rectangles. (A more general statement will be proved in Lemma 14.) Thus when creating bag X' from bag X we add $O(1)$ new items per pixel and hence $|X'| \in O(|X|)$ and $\mathcal{T}^H$ has width $O(tw(D))$. ◀

3.4 Solving 2-dominating Set

To solve the restricted distance-2-dominating set problem on H, we first show that the problem can be expressed as a monadic second-order logic formula [5]. In particular, a set S is a feasible solution for this problem if and only if

$$S \subseteq \Gamma \quad \wedge \quad \forall u \in U\, \exists \rho \in \mathcal{R}\, \exists \gamma \in S:\ \mathrm{adj}(u, \rho) \wedge \mathrm{adj}(\rho, \gamma)$$

where adj is a logic formula to encode that its two parameters are adjacent in H. Since H has bounded treewidth, we can find the smallest set S that satisfies this or report that no such S exists in linear time using Courcelle's theorem [5]. Here "linear" refers to the number of bags and hides a term that only depends on the treewidth. One can show that a thin polygon has $O(n)$ pixels (we will show something more general in Lemma 13). Therefore graph D has $O(n)$ vertices and hence a tree decomposition with $O(n)$ bags. In consequence the run-time is hence $O(f(tw(D))n)$ for some computable function f.

3.5 Run-time considerations

We briefly discuss here how to do all other steps in linear time, under some reasonable assumptions. The first step is to find the pixels. To do so, we need to compute the vertical decomposition (i.e., the partition obtained by extending only vertical rays from reflex vertices), which can be done in $O(n)$ time [3]. Likewise, compute the horizontal decomposition. Since (in a thin polygon) none of the rays intersect, we can obtain the pixels (and with it, the pixelation-graph and D) in linear time. Since D is planar, we can compute an $O(1)$-approximation of its treewidth in linear time [13], and hence can find $\mathcal{T}$ with width $O(tw(D))$. Next we need to simplify U and Γ. The run-time to do so depends on the exact form of the original U and Γ, but as long as those have a simple enough form that we can answer queries such as "does the interior of pixel ψ intersect U" in constant time, the overall time is $O(1)$ per pixel and hence overall linear.

Next we need to find the rectangles $\mathcal{R}$. In a thin polygon, all maximal rectangles are either a "slice" defined by the vertical or horizontal decomposition, or are a maximal line segment composed of pixel sides. All such slices and maximal line segments can be found from the pixelation in linear time, and there are $O(n)$ of them. This may yield some rectangles that are not maximal, but we can retain those without harm since even then any pixel intersects $O(1)$ rectangles. Constructing H from these three sets, and building $\mathcal{T}^H$ given $\mathcal{T}$, can also clearly be done in linear time. Putting everything together, we hence have:

▶ **Theorem 8.** *Let P be a thin polygon for which the dual graph has treewidth k. Then for any set $U \subseteq P$ and $\Gamma \subseteq P \cup \Psi$, we can solve the (U, Γ, P)-rGuarding problem in time $O(f(k)n)$ time for some computable function f.*

4 Generalizations

In this section, we give some applications and generalizations of Theorem 8.

4.1 Thin Polygons with Few Holes

We claimed earlier that a simple thin polygon is a tree polygon, and give here a formal proof because it will be useful later.

▶ **Lemma 9.** *Let P be a thin polygon. If P has no holes, then the dual graph D of the pixelation of P is a tree.*

Proof. Assume for contradiction that D contains a cycle. By tracing along the midpoints of the pixels-sides corresponding to this cycle, we can create a simple closed curve C that is inside P, yet has pixel-corners both inside and outside C. In a thin polygon, all pixel-corners are on the boundary of P, so the boundary of P has points both inside and outside a simple closed curve that is strictly within P. This is possible only if P has holes. ◀

Since every tree has treewidth 1, we hence have:

▶ **Corollary 10.** *Let P be a thin polygon that has no holes. Then for any sets $U \subseteq P$ and $\Gamma \subseteq P \cup \Psi$, we can solve the (U, Γ, P)-rGuarding problem in $O(n)$ time.*

Inspecting the proof of Lemma 9, we see that in fact every cycle of D gives rise to a hole that is inside the curve defined by the cycle. If D has f inner faces, then each face defines a cycle in D, and the insides of these cycles are disjoint. Therefore, D has at least f holes. Turning things around, if the polygon has h holes, then D has at most h inner faces. In consequence, D is a so-called h-outerplanar graph (i.e., if we remove all vertices from the outer-face and repeat h times, then all vertices have been removed). It is well-known that h-outerplanar graphs have treewidth $O(h)$ (see e.g. [7]).

▶ **Corollary 11.** *Let P be a thin polygon with h holes. Then for any sets $U \subseteq P$ and $\Gamma \subseteq P \cup \Psi$, we can solve the (U, Γ, P)-rGuarding problem in time $O(f(h)n)$ time for some computable function f.*

4.2 Polygons That are not Thin

The construction of the tree decomposition of H in Section 3.3 works even if P is not thin. However, the bound on the resulting treewidth, and the claim on the linear run-time both used that the polygon is thin. We can generalize these results to polygons that are somewhat thicker. More precisely, we say that a polygon is *K-thin* (for some integer $K \geq 1$) if the dual graph D of P contains no induced $(K + 1) \times (K + 1)$-grid. A thin polygon is a 1-thin polygon in this terminology, because a pixel-corner is in the interior if and only if the four pixels around it form a 4-cycle, hence a 2×2-grid, in D. Notice that K-thin is equivalent to saying that the pixelation-graph has no induced $(K+2)\times(K+2)$-grid. We need some observations:

▶ **Lemma 12.** *Let P be a K-thin polygon. Then, for any pixel-corner p, there exists a point on the boundary of P that is in the first quadrant relative to p and has distance at most $2K+1$ from p, where distance is measured by the length of the path in the pixelation-graph.*

Proof. Consider any path in the pixelation graph that starts at p and goes upward or rightward for at most $K+1$ edges each. If some such path reaches a point on the boundary after at most $2K+1$ edges, then we are done. Else the union of these paths forms a $(K+2)\times(K+2)$-grid in the pixelation-graph, and P is not K-thin. ◀

▶ **Lemma 13.** *The pixelation of a K-thin polygon with n vertices has $O(K^2 n)$ pixels.*

Proof. There are $O(n)$ boundary vertices: one for each vertex of P, and one whenever a ray hits the boundary (of which there are at most $n-4$ since there are $n/2-2$ reflex vertices and each emits two rays). Each vertex on the boundary has $O(K^2)$ pixel-corners within distance $2K+1$. By the previous lemma all pixel-corners must be within such distance, so there are $O(K^2 n)$ pixel-corners, and hence $O(K^2 n)$ pixels. ◀

Since a K-thin polygon contains no $(K+2)\times(K+2)$-grid in the pixelation, one can also show the following (details are in the full version [2]):

▶ **Lemma 14.** *Any pixel ψ in a K-thin polygon P is intersected by $O(K^2)$ maximal axis-aligned rectangles inside P.*

▶ **Theorem 15.** *Let P be a K-thin simple polygon. Then for any set $U\subseteq P$ and $\Gamma\subseteq P\cup\Psi$, the (U,Γ,P)-rGuarding problem can be solved in $O(f(K^3)K^2 n)$ time for some computable function $f(.)$.*

Proof. The pixelation of P has $O(k^2 n)$ vertices by Lemma 13, and can be constructed in $O(k^2 n)$ time by constructing the vertical decomposition and then ray-shooting along the horizontal rays emitted from reflex vertices. For any pixel-corner p, there exists a point on the boundary of P that and has distance at most $2K+1$ from p. It follows that the pixelation graph is $(2K+1)$-outerplanar, and hence it (and also its dual graph D) have treewidth $O(k)$. Find a tree decomposition of D with treewidth $O(k)$ and $O(k^2 n)$ bags; this can be done in linear time since D is planar [13]. Replace each pixel in each bag of $\mathcal{T}$ by points, guards and rectangles as explained in Section 3.3. Since each pixel belongs to $O(k^2)$ rectangles, the resulting tree decomposition has width $O(k^3)$. Now solve the restricted 2-dominating set problem using Courcelle's theorem. The run-time is as desired since we have $O(k^2 n)$ bags and treewidth $O(k^3)$. ◀

4.3 K-Thin Polygons with Few Holes

Both of the above generalizations can be combined, creating an algorithm that is fixed-parameter tractable in both the thinness and the number of holes.

▶ **Lemma 16.** *Let P be a polygon that is K-thin and that has h holes. Then the dual graph of P has treewidth $O(K(h+1))$.*

Proof. Let D' be the (full) dual graph of the pixelation graph, i.e., it is graph D plus a vertex for each hole and for the outerface, connected to all incident pixels. We claim that all vertices in D' have distance $O(K(h+1))$ from the outerface-vertex. This implies that D' (and hence also D) is $O(K(h+1))$-outerplanar and so has treewidth $O(K(h+1))$.

To prove the distances, we first connect holes as follows. If H is a hole, then let c be a corner of H that maximizes the sum of the coordinates (breaking ties arbitrarily). Let ψ be a pixel incident to c and let c' be some other corner of ψ. By Lemma 12, there exists a pixel-corner p on the boundary of P within distance $2K+1$ from c'. Moreover, the path from c' to p goes only up and right. Thus p is incident to the outer-face or to a hole H', where $H' \neq H$ by choice of c. Following this path, we can hence find a path in D of length $O(K)$ from the vertex representing H to the vertex representing H' or the outer-face. Combining all these paths, we can reach the outer-face from any hole in a path of length $O(K(h+1))$.

Now for any other vertex in D (hence pixel ψ), let c be one pixel-corner, and find a path in the pixelation of length at most $2K+1$ from c to some point on the boundary. Following this path, we can find a path of length $O(K)$ in D from ψ to some hole or the outer-face, and hence reach the outer-face along a path of length $O(K(h+1))$. The result follows. ◀

The following summarizes this approach, and includes all previous results.

▶ **Theorem 17.** *Let P be a polygon that is K-thin and has h holes. Then for any set $U \subseteq P$ and $\Gamma \subseteq P \cup \Psi$, the (U, Γ, P)-rGuarding problem can be solved in $O(f((K(h+1))^3)(K(h+1))^2 n)$ time for some computable function $f(.)$. In particular, the rGuarding problem is fixed-parameter tractable in $K+h$.*

5 Conclusion

In this paper, we studied the problem of guarding a thin polygon under the model that a guard can only see a point if the entire axis-aligned rectangle spanned by them is inside the polygon. We showed that this problem is NP-hard, even in thin polygons, if there are holes. If there are few holes or, more generally, the dual graph of the polygon has bounded treewidth, then we solved the problem in linear time. Our approach is quite flexible in that we can specify which points must be guarded and which points/pixels are allowed to be used as guards. In fact, with minor modifications even more flexibility is possible. We could allow any guard that consists of a connected union of pixels (as long as any pixel is intersected by $O(1)$ guards). We could even consider other guarding models by replacing the rectangles in $\mathcal{R}$ by arbitrary connected unions of pixels and pixel-sides (again as long as any pixel is intersected by $O(1)$ such shapes). For all these, the (naturally defined) auxiliary graph H has treewidth $O(tw(D))$ in thin polygons, and we can hence solve r-guarding by solving the restricted distance-2-dominating set.

Our results mean that the complexity of r-guarding is nearly resolved, with the exception of polygons that have $O(1)$ holes but are not K-thin for a constant number K. For such polygons, is the problem still NP-hard? Also, for polygons that have a large number of holes, is the problem APX-hard, or can we develop a PTAS?

Acknowledgments. The authors would like to thank Justin Iwerks and Joe Mitchell; the discussions with them made us consider thin polygons in the first place.

References

1 T. Biedl, M. T. Irfan, J. Iwerks, J. Kim, and J. S. B. Mitchell. Guarding polyominoes. In *Proc. of the ACM Symp. on Computational Geometry (SoCG'11)*, pages 387–396, 2011.

2 T. C. Biedl and S. Mehrabi. On r-guarding thin orthogonal polygons. *CoRR*, abs/1604.07100, 2016. URL: `http://arxiv.org/abs/1604.07100`.

3 B. Chazelle. Triangulating a simple polygon in linear time. *Disc. Comp. Geom.*, 6(5):485–524, 1991.

4 V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18:39–41, 1975. `doi:10.1137/S0097539796302531`.

5 B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

6 J. Culberson and R. A. Reckhow. Orthogonally convex coverings of orthogonal polygons without holes. *Journal of Computer and System Sciences*, 39(2):166–204, 1989.

7 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Mark, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, Heidelberg, Germany, 2015.

8 S. Durocher and S. Mehrabi. Guarding orthogonal art galleries using sliding cameras: algorithmic and hardness results. In *Proceedings of Mathematical Foundations of Computer Science (MFCS 2013)*, volume 8087 of *LNCS*, pages 314–324, 2013.

9 S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001. `doi:10.1007/s00453-001-0040-8`.

10 D. S. Franzblau and D. J. Kleitman. An algorithm for constructing regions with rectangles: Independence and minimum generating sets for collections of intervals. In *Proceedings of the ACM Symposium on Theory of Computing (STOC 1984)*, pages 167–174, 1984.

11 M. R. Garey and D. S. Johnson. The Rectilinear Steiner Tree Problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

12 S. K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Disc. App. Math.*, 158(6):718–722, 2010.

13 F. Kammer and T. Tholey. Approximate tree decompositions of planar graphs in linear time. In *Proc. of the ACM-SIAM Symp. on Discrete Algorithms (SODA 2012)*, pages 683–698, 2012.

14 G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.

15 M. J. Katz and G. Morgenstern. Guarding orthogonal art galleries with sliding cameras. *Int. J. Comput. Geometry Appl.*, 21(2):241–250, 2011.

16 J. M. Keil. Minimally covering a horizontally convex orthogonal polygon. In *Proceedings of the ACM Symposium on Computational Geometry (SoCG 1986)*, pages 43–51, 1986.

17 E. Krohn and B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013.

18 D. T. Lee and Arthur K. Lin. Computational complexity of art gallery problems. *IEEE Trans. on Information Theory*, 32(2):276–282, 1986. `doi:10.1109/TIT.1986.1057165`.

19 A. Lingas, A. Wasylewicz, and P. Zylinski. Linear-time 3-approximation algorithm for the r-star covering problem. *Int. J. Comput. Geometry Appl.*, 22(2):103–142, 2012.

20 R. Motwani, A. Raghunathan, and H. Saran. Perfect graphs and orthogonally convex covers. *SIAM J. Discrete Math.*, 2(3):371–392, 1989.

21 J. O'Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

22 L. Palios and P. Tzimas. Minimum r-star cover of class-3 orthogonal polygons. In *Proceedings of the International Workshop on Combinatorial Algorithms (IWOCA 2014)*, volume 8986 of *LNCS*, pages 286–297. Springer, 2015.

23 S. Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.

24 D. Schuchardt and H.-D. Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41(2):261–267, 1995.

25 P. J. Slater. R-domination in graphs. *J. ACM*, 23(3):446–450, 1976.

26 A. P. Tomás. Guarding thin orthogonal polygons is hard. In *Proceedings of Fundamentals of Computation Theory (FCT 2013)*, volume 8070 of *LNCS*, pages 305–316, 2013.

27 C. Worman and J. M. Keil. Polygon decomposition and the orthogonal art gallery problem. *Int. J. Comput. Geometry Appl.*, 17(2):105–138, 2007. doi:10.1142/S0218195907002264.

Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation

Philip Bille[1], Patrick Hagge Cording[2], Inge Li Gørtz[3], Frederik Rye Skjoldjensen[4], Hjalte Wedel Vildhøj[5], and Søren Vind[6]

1 Technical University of Denmark, DTU Compute, Lyngby, Denmark
2 Technical University of Denmark, DTU Compute, Lyngby, Denmark
3 Technical University of Denmark, DTU Compute, Lyngby, Denmark
4 Technical University of Denmark, DTU Compute, Lyngby, Denmark
5 Technical University of Denmark, DTU Compute, Lyngby, Denmark
6 Technical University of Denmark, DTU Compute, Lyngby, Denmark

Abstract

Given a static reference string R and a source string S, a relative compression of S with respect to R is an encoding of S as a sequence of references to substrings of R. Relative compression schemes are a classic model of compression and have recently proved very successful for compressing highly-repetitive massive data sets such as genomes and web-data. We initiate the study of relative compression in a dynamic setting where the compressed source string S is subject to edit operations. The goal is to maintain the compressed representation compactly, while supporting edits and allowing efficient random access to the (uncompressed) source string. We present new data structures that achieve optimal time for updates and queries while using space linear in the size of the optimal relative compression, for nearly all combinations of parameters. We also present solutions for restricted and extended sets of updates. To achieve these results, we revisit the dynamic partial sums problem and the substring concatenation problem. We present new optimal or near optimal bounds for these problems. Plugging in our new results we also immediately obtain new bounds for the string indexing for patterns with wildcards problem and the dynamic text and static pattern matching problem.

1998 ACM Subject Classification E.4 Coding and Information Theory, E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Relative compression, dynamic compression, dynamic partial sum, substring concatenation, external macro compression

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.18

1 Introduction

Given a static reference string R and a source string S, a *relative compression of S with respect to R* is an encoding of S as a sequence of references to substrings of R. Relative compression (or *external macro compression*) is a classic model of compression defined by Storer and Szymanski [34, 35] in 1978 and has since been used in a wide range of compression scenarios [26, 27, 23, 24, 6, 9, 19]. To compress massive highly-repetitive data sets, such as biological sequences and web collections, relative compression has been shown to be very practical [23, 24, 19].

Relative compression is often applied to compress multiple similar source strings. In such settings relative compression is superior to compressing the source strings individually. For

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 18; pp. 18:1–18:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

instance, human genomes are 99% similar and hence relative compression might be used to compress a large collection of sequenced genomes using, e.g., the human reference genome as the static reference string. We focus on the case of compressing a single source string, but our results trivially generalize to compressing multiple source strings.

In this paper we initiate the study of relative compression in a *dynamic setting*, where the compressed source string S is subject to edit operations (insertions, deletions, and replacements of single characters). The goal is to maintain the compressed representation compactly, while supporting edits and allowing efficient random access to the (uncompressed) source string. Efficient data structures supporting these operations allow us to avoid costly recompression of massive data sets after updates.

We provide the first non-trivial bounds for this problem. We present new data structures that achieve *optimal* time for updates and queries while using space linear in the size of the *optimal* relative compression, for nearly all combinations of parameters. We also present solutions for restricted and extended sets of updates.

To achieve these results, we revisit the *dynamic partial sums problem* and the *substring concatenation problem*. We present new optimal or near optimal bounds for both of these problems (see detailed discussion below). Furthermore, plugging in our new results immediately leads to new bounds for the *string indexing for patterns with wildcards problem* [25, 5] and the *the dynamic text and static pattern matching problem* [2].

1.1 Dynamic Relative Compression

Given a *reference string* R and a *source string* S, a *relative compression of* S *with respect to* R is a sequence $C = (i_1, j_1), ..., (i_{|C|}, j_{|C|})$ such that $S = R[i_1, j_1] \cdots R[i_{|C|}, j_{|C|}]$. We call C a *substring cover* for S. The substring cover is *optimal* if $|C|$ is minimum over all relative compressions of S with respect to R. The *dynamic relative compression problem* is to maintain a relative compression of S under the following operations. Let i be a position in S and α be a character.

access(i): return the character $S[i]$,
replace(i, α): change $S[i]$ to character α,
insert(i, α): insert character α before position i in S,
delete(i): delete the character at position i in S.

Note that operations insert and delete change the length of S by a single character. In all bounds below, the access(i) operation extends to decompressing an arbitrary substring of length ℓ using only $O(\ell)$ additional time.

Our Results. Throughout the paper, let r be the length of the reference string R, N be the length of the (uncompressed) string S, and n be the size of an optimal relative compression of S with regards to R. All of the bounds mentioned below and presented in this paper hold for a standard unit-cost RAM with w-bit words with standard arithmetic and logical operations on a word. This means that the algorithms can be implemented directly in standard imperative programming languages such as C [22] or C++ [36]. An index into R or S can be stored in a single word and hence $w \geq \log(n + r)$.

► **Theorem 1.** *Let R and S be a reference and source string of lengths r and N, respectively, and let n be the length of the optimal substring cover of S by R. Then, we can solve the dynamic relative compression problem supporting* access*,* replace*,* insert*, and* delete

(i) *in $O(n + r)$ space and $O\left(\frac{\log n}{\log \log n} + \log \log r\right)$ time per operation, or*

(ii) *in $O(n + r \log^{\epsilon} r)$ space and $O\left(\frac{\log n}{\log \log n}\right)$ time per operation, for any constant $\epsilon > 0$.*

These are the first non-trivial bounds for the problem. Together, the bounds are optimal for most natural parameter combinations. In particular, any data structure for a string of length N supporting access, insert, and delete must use $\Omega(\log N/\log\log N)$ time in the worst-case regardless of the space [13] (this is called the *list representation problem*). Since $n \leq N$, we can view $O(\log n/\log\log n)$ as a compressed version of the optimal time bound that is always $O(\log N/\log\log N)$ and better when S is compressible. Hence, Theorem 1(i) provides a linear-space solution that achieves the compressed time bound except for an $O(\log\log r)$ additive term. Note that whenever $n \geq (\log r)^{\log^{\epsilon}\log r}$, for any $\epsilon > 0$, the $\log n/\log\log n$ term dominates the query time and we match the compressed time bound. Hence, Theorem 1(i) is only suboptimal in the special case when n is almost exponentially smaller than r. In this case, we can use Theorem 1(ii) which always provides a solution achieving the compressed time bound at the cost of increasing the space to $O(n + r\log^{\epsilon} r)$.

We note that dynamic compression under different models of compression has been studied extensively [17, 11, 10, 33, 16, 12, 21, 28]. However, all of these results require space dependent on the size of the original string and hence cannot take full advantage of highly-repetitive data.

1.2 Dynamic Partial Sums

The *partial sums problem* is to maintain an array $Z[1..s]$ under the following operations.

- **sum(i):** return $\sum_{j=1}^{i} Z[j]$,
- **update(i, Δ):** set $Z[i] = Z[i] + \Delta$,
- **search(t):** return $1 \leq i \leq s$ such that $\mathsf{sum}(i-1) < t \leq \mathsf{sum}(i)$. To ensure well-defined answers, we require that $Z[i] \geq 0$ for all i.

The partial sums problem is a classic and well-studied problem, see e.g., [8, 32, 20, 13, 18, 30]. In our context, we consider the problem in the word RAM model, where each array entry stores a w-bit integer and the element of the array can be changed by δ-bit integers, i.e., the argument Δ can be stored in δ bits. In this setting, Pătraşcu and Demaine [30] gave a linear-space data structure with $\Theta(\log s/\log(w/\delta))$ time per operation. They also gave a matching lower bound.

We consider the following generalization supporting dynamic changes to the array. The *dynamic partial sums problems* is to additionally support the following operations.

- **insert(i, Δ):** insert a new entry in Z with value Δ before $Z[i]$,
- **delete(i):** delete the entry $Z[i]$ of value at most Δ.
- **merge(i):** replace entry $Z[i]$ and $Z[i+1]$ with a new entry with value $Z[i] + Z[i+1]$.
- **divide(i, t):** , where $0 \leq t \leq Z[i]$. Replace entry $Z[i]$ by two new consecutive entries with value t and $Z[i] - t$, respectively.

Hon et al. [18] and Navarro and Sadakane [29] presented optimal solutions for this problem in the case where the entries in Z are at most polylogarithmic in s (they did not explicitly consider the merge and divide operation).

Our Results. We show the following improved result.

▶ **Theorem 2.** *Given an array of length s storing w-bit integers and fixed parameter δ, such that $\Delta < 2^{\delta}$, we can solve the dynamic partial sums problem supporting* sum, update, search, insert, delete, merge, *and* divide *in linear space and $O(\log s/\log(w/\delta))$ time per operation.*

Note that this bound simultaneously matches the optimal time bound for the standard partial sums problem and supports storing arbitrary w-bit values in the entries of the array, i.e., the values we can handle in optimal time are exponentially larger than in the previous results.

To achieve our bounds we extend the static solution by Pătraşcu and Demaine [30]. Their solution is based on storing a sampled subset of *representative elements* of the array and difference encode the remaining elements. They pack multiple difference encoded elements in words and then apply word-level parallelism to speedup the operations. To support insert and delete the main challenge is to maintain the representative elements that now dynamically move within the array. We show how to efficiently do this by combining a new representation of representative elements with a recent result by Pătraşcu and Thorup [31]. Along the way we also slightly simplify the original construction by Pătraşcu and Demaine [30].

1.3 Substring Concatenation

Let R be a string of length r. A *substring concatenation query* on R takes two pairs of indices (i, j) and (i', j') and returns the start position in R of an occurrence of $R[i, j]R[i', j']$, or NO if the string is not a substring of R. The *substring concatenation problem* is to preprocess R into a data structure that supports substring concatenation queries.

Amir et al. [2] gave a solution using $O(r\sqrt{\log r})$ space with query time $O(\log \log r)$, and recently Gawrychowski et al. [15] showed how to solve the problem in $O(r \log r)$ space and $O(1)$ time.

Our Results. We give the following improved bounds.

▶ **Theorem 3.** *Given a string R of length r, the substring concatenation problem can be solved in either*

(i) $O(r \log^{\epsilon} r)$ *space and* $O(1)$ *time, for any constant* $\epsilon > 0$, *or*

(ii) $O(r)$ *space and* $O(\log \log r)$ *time.*

Hence, Theorem 3(i) matches the previous $O(1)$ time bound while reducing the space from $O(r \log r)$ to $O(r \log^{\epsilon} r)$ and Theorem 3(ii) achieves linear space while using $O(\log \log r)$ time. Plugging in the two solutions into our solution for dynamic relative compression leads to the two branches of Theorem 1.

To achieve the bound in (i), the main idea is a new construction that efficiently combines compact data structure for 1D range reporting [3] with the recent constant time weighted level ancestor data structure for suffix trees [15]. The bound in (ii) follows as a simple implication of another recent result for *unrooted LCP queries* [5] by some of the authors. Due to lack of space, we refer to the full version of the paper (see [4]) for the details of our solution.

The substring concatenation problem is a key component in several solutions to the *string indexing for patterns with wildcards problem* [5, 7, 25], where the goal is to preprocess a string T to support pattern matching queries for patterns with wildcards. Plugging in Theorem 3(i) we immediately obtain the following new bound for the problem.

▶ **Corollary 4.** *Let T be a string of length t. For any pattern string P of length p with k wildcards, we can support pattern matching queries on T using $O(t \log^{\epsilon} t)$ space and $O(p + \sigma^k)$ time for any constant $\epsilon > 0$.*

This improves the running time of fastest linear space solution by a factor $\log \log t$ at the cost of increasing the space slightly by a factor $\log^{\epsilon} t$. See [25] for detailed overview of the known results.

1.4 Extensions

Finally, we present two extensions of the dynamic relative compression problem. The proofs of these extensions are also omitted here and can be found in the full version of the paper.

1.4.1 Dynamic Relative Compression with Access and Replace

If we restrict the operations to access and replace we obtain the following improved bound.

▶ **Theorem 5.** *Let R and S be a reference and source string of lengths r and N, respectively, and let n be the length of the optimal substring cover of S by R. Then, we can solve the dynamic relative compression problem supporting* access *and* replace *in $O(n + r)$ space and $O(\log \log N)$ expected time.*

This version of dynamic relative compression is a key component in the *dynamic text and static pattern matching problem*, where the goal is to efficiently maintain a set of occurrences of a pattern P in a text T that is dynamically updated by changing individual characters. Let p and t denote the lengths of P and T, respectively. Amir et al. [2] gave a data structure using $O(t + p\sqrt{\log p})$ space which supports updates in $O(\log \log p)$ time. The computational bottleneck in the update operation is to update a substring cover of size $O(p)$. Plugging in the bounds from Theorem 5, we immediately obtain the following improved bound, matching the previous time bound while improving the space to linear.

▶ **Corollary 6.** *Given a pattern P and text T of lengths p and t, respectively, we can solve the dynamic text and static pattern matching problem in $O(t + p)$ space and $O(\log \log p)$ expected time per update.*

1.4.2 Dynamic Relative Compression with Split and Concatenate

We also consider maintaining a set of compressed strings under split and concatenate operations (as in Alstrup et al. [1]). Let R be a reference string and let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of strings compressed relative to R. In addition to access, replace, insert and delete we also define the following operations.

concat(i, j): Add string $S_i \cdot S_j$ to $\mathcal{S}$ and remove S_i and S_j.

split(i, j): Remove S_i from $\mathcal{S}$ and add $S_i[1, j - 1]$ and $S_i[j, |S_i|]$.

We obtain the following bounds.

▶ **Theorem 7.** *Let R be a reference string of length r, let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of source strings of total length N, and let n be the total length of the optimal substring covers of the strings in $\mathcal{S}$. Then, we can solve the dynamic relative compression problem supporting* access, replace, insert, delete, split, *and* concat,

(i) *in space $O(n + r)$ and time $O(\log n)$ for* access *and time $O(\log n + \log \log r)$ for* replace, insert, delete, split, *and* concat, *or*

(ii) *in space $O(n + r \log^{\epsilon} r)$ and time $O(\log n)$ for all operations.*

Hence, compared to the bounds in Theorem 1 we only increase the time bounds by an additional $\log \log n$ factor.

2 Dynamic Relative Compression

In this section we show how Theorems 2 and 3 lead to Theorem 1.

Let $C = ((i_1, j_1), ..., (i_{|C|}, j_{|C|}))$ be the compressed representation of S. From now on, we refer to C as the *cover of* S, and call each element (i_l, j_l) in C a *block*. Recall that a block (i_l, j_l) refers to a substring $R[i_l, j_l]$ of R. A cover C is *maximal* if concatenating any two consecutive blocks $(i_l, j_l), (i_{l+1}, j_{l+1})$ in C yields a string that does not occur in R, i.e., the string $R[i_l, j_l]R[i_{l+1}, j_{l+1}]$ is not a substring of R. We need the following lemma.

▶ **Lemma 8.** *If C_{MAX} is a maximal cover and C is an arbitrary cover of S, then $|C_{\text{MAX}}| \leq 2|C| - 1$.*

Proof. In each block b of C there can start at most two blocks in C_{MAX}, because otherwise two adjacent blocks in C_{MAX} would be entirely contained in the block b, contradicting the maximality of C_{MAX}. Since the last block of both C and C_{MAX} end at the last position of S, a contradiction of the maximality is already obtained when more than one block of C_{MAX} start in the last block of C. Hence, $|C_{\text{MAX}}| \leq 2|C| - 1$. ◀

Recall that n is the size of an optimal cover of S with regards to R. The lemma implies that we can maintain a compression of size at most $2n - 1$ by maintaining a maximal cover of S. The remainder of this section describes our data structure for maintaining and accessing such a cover.

Initially, we can use the suffix tree of R to construct a maximal cover of S in $O(N + r)$ time by greedily matching the maximal prefix of the remaining part of S with any suffix of R. This guarantees that the blocks constitute a maximal cover of S.

2.1 Data Structure

The high level idea for supporting the operations on S is to store the sequence of block lengths $j_1 - i_1 + 1, \ldots, j_{|C|} - i_{|C|} + 1$ in a dynamic partial sums data structure. This allows us, for example, to identify the block that encodes the k^{th} character in S by performing a $\mathsf{search}(k)$ query.

Updates to S are implemented by splitting a block in C. This may break the maximality property so we use substring concatenation queries on R to detect if blocks can be merged. We only need a constant number of substring concatenation queries to restore maximality. To maintain the correct sequence of block lengths we use update, divide and merge operations on the dynamic partial sums data structure.

Our data structure consist of the string R, a substring concatenation data structure of Theorem 3 for R, a maximal cover C for S stored in a doubly linked list, and the dynamic partial sums data structure of Theorem 2 storing the block lengths of C. We also store auxiliary links between a block in the doubly linked list and the corresponding block length in the partial sums data structure, and a list of alphabet symbols in R with the location of an occurrence for each symbol. By Lemma 8 and since C is maximal we have $|C| \leq 2n - 1 = O(n)$. Hence, the total space for C and the partial sums data structure is $O(n)$. The space for R is $O(r)$ and the space for substring concatenation data structure is either $O(r)$ or $O(r \log^{\epsilon} r)$ depending on the choice in Lemma 3. Hence, in total we use either $O(n + r)$ or $O(n + r \log^{\epsilon} r)$ space.

2.2 Answering Queries

To answer $\mathsf{access}(i)$ queries we first compute $\mathsf{search}(i)$ in the dynamic partial sums structure to identify the block $b_l = (i_l, j_l)$ containing position i in S. The local index in $R[i_l, j_l]$ of the i^{th} character in R is $\ell = i - \mathsf{sum}(l-1)$, and thus the answer to the query is the character $R[i_l + \ell - 1]$.

We perform replace and delete by first identifying $b_l = (i_l, j_l)$ and ℓ as above. Then we partition b_l into three new blocks $b_l^1 = (i_l, i_l + \ell - 2)$, $b_l^2 = (i_l + \ell - 1, i_l + \ell - 1)$, $b_l^3 = (i_l + \ell, j_l)$ where b_l^2 is the single character block for index i in S that we must change. In replace we change b_l^2 to an index of an occurrence in R of the new character (which we can find from the list of alphabet symbols), while we remove b_l^2 in delete. The new blocks and their neighbors, that is, b_{l-1}, b_l^1, b_l^2, b_l^3, and b_{l+1} may now be non-maximal. To restore maximality we perform substring concatenation queries on each consecutive pair of these 5 blocks, and replace non-maximal blocks with merged maximal blocks. All other blocks are still maximal, since the strings obtained by concatenating $b_{l'}$ with $b_{l'+1}$, for all $l' < l - 1$ and all $l' > l$, was not present in R before the change and is not present afterwards. A similar idea is used by Amir et al. [2]. We perform update, divide and merge operations to maintain the corresponding lengths in the dynamic partial sums data structure. The insert operation is similar, but inserts a new single character block between two parts of b_l before restoring maximality. Observe that using $\delta = O(1)$ bits in update is sufficient to maintain the correct block lengths.

In total, each operation requires a constant number of substring concatenation queries and dynamic partial sums operations; the latter having time complexity $O(\log n / \log(w/\delta)) = O(\log n / \log \log n)$ as $w \geq \log n$ and $\delta = O(1)$. Hence, the total time for each access, replace, insert, and delete operation is either $O(\log n / \log \log n + \log \log r)$ or $O(\log n / \log \log n)$ depending on the substring concatenation data structure used. In summary, this proves Theorem 1.

3 Dynamic Partial Sums

In this section we prove Theorem 2. We support the operations $\mathsf{insert}(i, \Delta)$ and $\mathsf{delete}(i)$ on a sequence of w-bit integer keys by implementing them using update and a divide or merge operation, respectively. This means that we support inserting or deleting keys with value at most 2^δ.

We first solve the problem for small sequences. The general solution uses a standard reduction, storing Z at the leaves of a B-tree of large outdegree. We use the solution for small sequences to navigate in the internal nodes of the B-tree.

We need the following recent result due to Pătraşcu and Thorup [31] on maintaining a set of integer keys X under insertions and deletions. The queries are as follows, where q is an integer. The membership query $\mathsf{member}(q)$ returns true if $q \in X$, predecessor $\mathsf{pred}_X(q)$ returns the largest key $x \in X$ where $x < q$, and successor $\mathsf{succ}_X(q)$ returns the smallest key $x \in X$ where $x \geq q$. The rank $\mathsf{rank}_X(q)$ returns the number of keys in X smaller than q, and $\mathsf{select}(i)$ returns the i^{th} smallest key in X.

▶ **Lemma 9** (Pătraşcu and Thorup [31]). *There is a data structure for maintaining a dynamic set of $w^{O(1)}$ w-bit integers that supports insert, delete, membership, predecessor, successor, rank and select in constant time per operation.*

3.1 Dynamic Partial Sums for Small Sequences

Let Z be a sequence of at most $B \leq w^{O(1)}$ integer keys. We will show how to store Z in linear space such that all dynamic partial sums operations can be performed in constant time. We let Y be the sequence of prefix sums of Z, defined such that each key $Y[i]$ is the sum of the first i keys in Z, i.e., $Y[i] = \sum_{j=1}^{i} Z[j]$. Observe that $\mathsf{sum}(i) = Y[i]$ and $\mathsf{search}(t)$ is the index of the successor of t in Y. Our goal is to store and maintain a representation of Y subject to the dynamic operations update, divide and merge in constant time per operation.

3.1.1 The Scheme by Pătrașcu and Demaine

We first review the solution to the static partial sums problem by Pătraşcu and Demaine [30], slightly simplified due to Lemma 9. Our dynamic solution builds on this.

The entire data structure is rebuilt every B operations as follows. We first partition Y greedily into *runs*. Two adjacent elements in Y are in the same run if their difference is at most $B2^{\delta}$, and we call the first element of each run a *representative* for all elements in the run. We use $\mathcal{R}$ to denote the sequence of representative values in Y and $\mathsf{rep}(i)$ to be the index of the representative for element $Y[i]$ among the elements in $\mathcal{R}$.

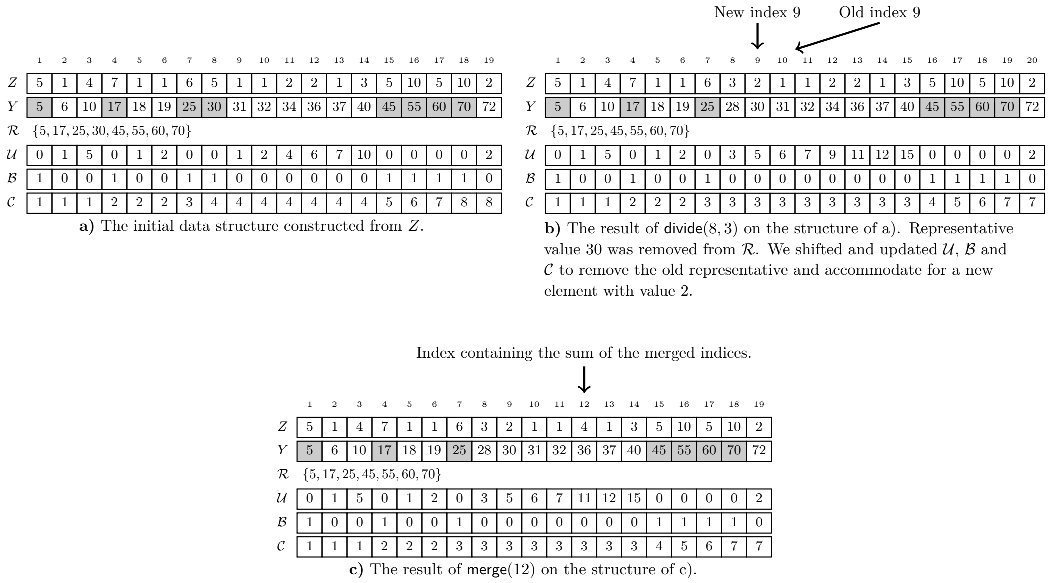
We store Y by splitting representatives and other elements into separate data structures: $\mathcal{I}$ and $\mathcal{R}$ store the representatives at the time of the last rebuild, while $\mathcal{U}$ stores each element in Y as an offset to its representative value as well as updates since the last rebuild. We ensure $Y[i] = \mathcal{R}[\mathsf{rep}(i)] + \mathcal{U}[i]$ for any i and can thus reconstruct the values of Y.

The representatives are stored as follows. $\mathcal{I}$ is the sequence of indices in Y of the representatives and $\mathcal{R}$ is the sequence of representative values in Y. Both $\mathcal{I}$ and $\mathcal{R}$ are stored using the data structure of Lemma 9. We can then define $\mathsf{rep}(i) = \mathsf{rank}_{\mathcal{I}}(\mathsf{pred}_{\mathcal{I}}(i))$ as the index of the representative for i among all representatives, and use $\mathcal{R}[\mathsf{rep}(i)] = \mathsf{select}_{\mathcal{R}}(\mathsf{rep}(i))$ to get the value of the representative for i.

We store in $\mathcal{U}$ the current difference from each element to its representative, $\mathcal{U}[i] = Y[i] - \mathcal{R}[\mathsf{rep}(i)]$ (i.e. updates between rebuilds are applied to $\mathcal{U}$). The idea is to pack $\mathcal{U}$ into a single word of B elements. Observe that $\mathsf{update}(i, \Delta)$ adds value Δ to all elements in Y with index at least i. We can support this operation in constant time by adding to $\mathcal{U}$ a word that encodes Δ for those elements. Since each difference between adjacent elements in a run is at most $B2^{\delta}$ and $|Y| = O(B)$, the maximum value in $\mathcal{U}$ after a rebuild is $O(B^2 2^{\delta})$. As B updates of size 2^{δ} may be applied before a rebuild, the changed value at each element due to updates is $O(B2^{\delta})$. So each element in $\mathcal{U}$ requires $O(\log B + \delta)$ bits (including an overflow bit per element). Thus, $\mathcal{U}$ requires $O(B(\log B + \delta))$ bits in total and can be packed in a single word for $B = O(\min\{w/\log w, w/\delta\})$.

Between rebuilds the stored representatives are potentially outdated because updates may have changed their values. However, observe that the values of two consecutive representatives differ by more than $B2^{\delta}$ at the time of a rebuild, so the gap between two representatives cannot be closed by B updates of δ bits each (before the structure is rebuilt again). Hence, an answer to $\mathsf{search}(t)$ cannot drift much from the values stored by the representatives; it can only be in a constant number of runs, namely those with a representative value $\mathsf{succ}_{\mathcal{R}}(t)$ and its two neighboring runs. In a run with representative value v, we find the smallest j (inside the run) such that $\mathcal{U}[j] + v - t > 0$. The smallest j found in all three runs is the answer to the $\mathsf{search}(t)$ query. Thus, by rebuilding periodically, we only need to check a constant number of runs when answering a $\mathsf{search}(t)$ query.

On this structure, Pătraşcu and Demaine [30] show that the operations sum, search and update can be supported in constant time each as follows:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z	5	1	4	7	1	1	6	5	1	1	2	2	1	3	5	10	5	10	2
Y	5	6	10	17	18	19	25	30	31	32	34	36	37	40	45	55	60	70	72
$\mathcal{R}$	$\{5, 17, 25, 30, 45, 55, 60, 70\}$																		
$\mathcal{U}$	0	1	5	0	1	2	0	0	1	2	4	6	7	10	0	0	0	0	2
$\mathcal{B}$	1	0	0	1	0	0	1	1	0	0	0	0	0	0	1	1	1	1	0
$\mathcal{C}$	1	1	1	2	2	2	3	4	4	4	4	4	4	4	5	6	7	8	8

a) The initial data structure constructed from Z.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Z	5	1	4	7	1	1	6	3	2	1	1	2	2	1	3	5	10	5	10	2
Y	5	6	10	17	18	19	25	28	30	31	32	34	36	37	40	45	55	60	70	72
$\mathcal{R}$	$\{5, 17, 25, 45, 55, 60, 70\}$																			
$\mathcal{U}$	0	1	5	0	1	2	0	3	5	6	7	9	11	12	15	0	0	0	0	2
$\mathcal{B}$	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0
$\mathcal{C}$	1	1	1	2	2	2	3	3	3	3	3	3	3	3	3	4	5	6	7	7

b) The result of divide$(8, 3)$ on the structure of a). Representative value 30 was removed from $\mathcal{R}$. We shifted and updated $\mathcal{U}$, $\mathcal{B}$ and $\mathcal{C}$ to remove the old representative and accommodate for a new element with value 2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z	5	1	4	7	1	1	6	3	2	1	1	4	1	3	5	10	5	10	2
Y	5	6	10	17	18	19	25	28	30	31	32	36	37	40	45	55	60	70	72
$\mathcal{R}$	$\{5, 17, 25, 45, 55, 60, 70\}$																		
$\mathcal{U}$	0	1	5	0	1	2	0	3	5	6	7	11	12	15	0	0	0	0	2
$\mathcal{B}$	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	1	1	1	0
$\mathcal{C}$	1	1	1	2	2	2	3	3	3	3	3	3	3	3	4	5	6	7	7

c) The result of merge(12) on the structure of c).

Figure 1 Illustrating operations on the data structure with $B2^\delta = 4$. a) shows the data structure immediately after a rebuild, b) shows the result of performing divide$(8, 3)$ on the structure of a), and c) shows the result of performing merge(12) on the structure of b).

sum(i): return the sum of $\mathcal{R}[\mathsf{rep}(i)]$ and $\mathcal{U}[i]$. This takes constant time as $\mathcal{U}[i]$ is a field in a word and representatives are stored using Lemma 9.

search(t): let $r_0 = \mathsf{rank}_{\mathcal{R}}(\mathsf{succ}_{\mathcal{R}}(t))$. We must find the smallest j such that $\mathcal{U}[j]+R[r]-t > 0$ for $r \in \{r_0 - 1, r_0, r_0 + 1\}$, where j is in run r. We do this for each r using standard word operations in constant time by adding $R[r] - t$ to all elements in $\mathcal{U}$, masking elements not in the run (outside indices $\mathsf{select}_{\mathcal{I}}(r)$ to $\mathsf{select}_{\mathcal{I}}(r + 1) - 1$, and counting the number of negative elements.

update(i, Δ): we do this in constant time by copying Δ to all fields $j \geq i$ by a multiplication and adding the result to $\mathcal{U}$.

To count the number of negative elements or find the least significant bit in a word in constant time, we use the technique by Fredman and Willard [14].

Notice that rebuilding the data structure every B operations takes $O(B)$ time, resulting in amortized constant time per operation. We can instead do this incrementally by a standard approach by Dietz [8], reducing the time per operation to worst case constant. The idea is to construct the new replacement data structure incrementally while using the old and complete data structure.

3.1.2 Efficient Support for divide and merge

We now show how to maintain the structure described above while supporting operations divide(i, t) and merge(i). An example supporting the following explanation is provided in Figure 1.

Observe that the operations are only local: Splitting $Z[i]$ into two parts or merging $Z[i]$ and $Z[i + 1]$ does not influence the precomputed values in Y (besides adding/removing values for the divided/merged elements). We must update $\mathcal{I}$, $\mathcal{R}$ and $\mathcal{U}$ to reflect these local changes accordingly. Because a divide or merge operation may create new representatives between rebuilds with values that do not fit in $\mathcal{U}$, we change $\mathcal{I}$, $\mathcal{R}$ and $\mathcal{U}$ to reflect these new representatives by rebuilding the data structure locally. This is done as follows.

Consider the run representatives. Both $\mathsf{divide}(i, t)$ and $\mathsf{merge}(i)$ may require us to create a new run, combine two existing runs or remove a run. In any case, we can find a replacement representative for each run affected. As the operations are only local, the replacement is either a divided or merged element, or one of the neighbors of the replaced representative. Replacing representatives may cause both indices and values for the stored representatives to change. We use insertions and deletions on $\mathcal{R}$ to update representative values.

Since the new operations change the indices of the elements, these changes must also be reflected in $\mathcal{I}$. For example, a $\mathsf{merge}(i)$ operation decrements the indices of all elements with index larger than i compared to the indices stored at the time of the last rebuild We should in principle adjust the $O(B)$ changed indices stored in $\mathcal{I}$. The cost of adjusting the indices accordingly when using Lemma 9 to store $\mathcal{I}$ is $O(B)$. Instead, to get our desired constant time bounds, we represent $\mathcal{I}$ using a resizable data structure with the same number of elements as Y that supports this kind of update. We must support $\mathsf{select}_{\mathcal{I}}(i)$, $\mathsf{rank}_{\mathcal{I}}(q)$, and $\mathsf{pred}_{\mathcal{I}}(q)$ as well as inserting and deleting elements in constant time. Because $\mathcal{I}$ has few and small elements, we can support the operations in constant time by representing it using a bitstring $\mathcal{B}$ and a structure $\mathcal{C}$ which is the prefix sum over $\mathcal{B}$ as follows.

Let $\mathcal{B}$ be a bitstring of length $|Y| \leq B$, where $\mathcal{B}[i] = 1$ iff there is a representative at index i. $\mathcal{C}$ has $|Y|$ elements, where $\mathcal{C}[i]$ is the prefix sum of $\mathcal{B}$ including element i. Since $\mathcal{C}$ requires $O(B \log B)$ bits in total we can pack it in a single word. We answer queries as follows: $\mathsf{rank}_{\mathcal{I}}(q)$ equals $\mathcal{C}[q-1]$, we answer $\mathsf{select}_{\mathcal{I}}(i)$ by subtracting i from all elements in $\mathcal{C}$ and return one plus the number of elements smaller than 0 (as done in $\mathcal{U}$ when answering search), and we find $\mathsf{pred}_{\mathcal{I}}(q)$ as the index of the least significant bit in $\mathcal{B}$ after having masked all indices larger than q. Updates are performed as follows. Using mask, shift and concatenate operations, we can ensure that $\mathcal{B}$ and $\mathcal{C}$ have the same size as Y at all times (we extend and shrink them when performing divide and merge operations). Inserting or deleting a representative is to set a bit in $\mathcal{B}$, and to keep $\mathcal{C}$ up to date, we employ the same ±1 update operation as used in $\mathcal{U}$.

We finally need to adjust the relative offsets of all elements with a changed representative in $\mathcal{U}$ (since they now belong to a representative with a different value). In particular, if the representative for $\mathcal{U}[j]$ changed value from v to v', we must subtract $v' - v$ from $\mathcal{U}[j]$. This can be done for all affected elements belonging to a single representative simultaneously in $\mathcal{U}$ by a single addition with an appropriate bitmask (update a range of $\mathcal{U}$). Note that we know the range of elements to update from the representative indices. Finally, we may need to insert or delete an element in $\mathcal{U}$, which can be done easily by mask, shift and concatenate operations on the word $\mathcal{U}$. This leads to Theorem 10.

▶ **Theorem 10.** *There is a linear space data structure for dynamic partial sums supporting each operation* search*,* sum*,* update*,* insert*,* delete*,* divide*, and* merge *on a sequence of length* $O(\min\{w/\log w, w/\delta\})$ *in worst-case constant time.*

3.2 Dynamic Partial Sums for Large Sequences

Willard [37] (and implicitly Dietz [8]) showed that a leaf-oriented B-tree with out-degree B of height h can be maintained in $O(h)$ worst-case time if: 1) searches, insertions and deletions take $O(1)$ time per node when no splits or merges occur, and 2) merging or splitting a node of size B requires $O(B)$ time.

We use this as follows, where Z is our integer sequence of length s. Create a leaf-oriented B-tree of degree $B = \Theta(\min\{w/\log w, w/\delta\})$ storing Z in the leaves, with height $h = O(\log_B n) = O(\log n/\log(w/\delta))$. Each node v uses Theorem 10 to store the $O(B)$

sums of leaves in each of the subtrees of its children. Searching for t in a node corresponds to finding the successor $Y[i]$ of t among these sums. Dividing or merging elements in Z corresponds to inserting or deleting a leaf. This concludes the proof of Theorem 2.

4 Conclusion

Our solution to DRC is built on data structures for the partial sums problem and the substring concatenation problem. Our partial sums-solution is optimal, but in order to get the desired constant query time for substring concatenation, our data structure uses $O(r \log^{\epsilon} r)$ space. Opposed to this, our linear space solution leads to $O(\log \log r)$ query time. We leave as an open problem if it is possible to get constant time substring concatenation queries using $O(r)$ space, which will also carry over to a stronger result for the DRC problem, and improved solutions for the string indexing for patterns with wildcards problem and the dynamic text and static pattern matching problem.

Currently we maintain a 2-approximation of the optimal cover. It would be useful to know if a better approximation ratio can be maintained under the same (or better) time and space bounds that we give.

Acknowledgments. We thank Pawel Gawrychowski for helpful discussions.

References

1 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *Proc. 11th SODA*, pages 819–828, 2000.

2 Amihood Amir, Gad M Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM TALG*, 3(2):19, 2007.

3 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Fast prefix search in little space, with applications. In *Proc. 18th ESA*, pages 427–438, 2010.

4 Philip Bille, Patrick Hagge Cording, Inge Li Gørtz, Frederik Rye Skjoldjensen, Hjalte Wedel Vildhøj, and Søren Vind. Dynamic relative compression, dynamic partial sums, and substring concatenation. *CoRR*, abs/1504.07851, 2015. URL: `http://arxiv.org/abs/1504.07851`.

5 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory Comput. Syst.*, 55(1):41–60, 2014.

6 B. G. Chern, Idoia Ochoa, Alexandros Manolakos, Albert No, Kartik Venkat, and Tsachy Weissman. Reference based genome compression. In *IEEE ITW*, pages 427–431, 2012.

7 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. 36th STOC*, pages 91–100, 2004.

8 Paul F. Dietz. Optimal algorithms for list indexing and subset rank. In *Proc. 1st WADS*, pages 39–46, 1989.

9 Huy Hoang Do, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Fast relative Lempel–Ziv self-index for similar sequences. *TCS*, 532:14–30, 2014.

10 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.

11 Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Succinct representation of sequences. Technical report, Università di Pisa, 2004.

12 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *TCS*, 372(1):115–121, 2007.

13 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st STOC*, pages 345–354, 1989.

14 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. System Sci.*, 47(3):424–436, 1993.

15 Paweł Gawrychowski, Moshe Lewenstein, and Patrick K Nicholson. Weighted ancestors in suffix trees. In *Algorithms – ESA 2014*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.

16 Rodrigo González and Gonzalo Navarro. Compressed text indexes with fast locate. In *Combinatorial Pattern Matching*, volume 4580, pages 216–227. Springer, 2007. doi:10.1007/978-3-540-73437-6_23.

17 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pages 841–850, 2003.

18 Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Succinct data structures for searchable partial sums with optimal worst-case performance. *TCS*, 412(39):5176–5186, 2011.

19 Christopher Hoobin, Simon J. Puglisi, and Justin Zobel. Relative Lempel-Ziv factorization for efficient storage and retrieval of web collections. *PVLDB*, 5(3):265–273, 2011.

20 Thore Husfeldt and Theis Rauhe. New lower bound techniques for dynamic partial sums and related problems. *SIAM J. Comput.*, 32(3):736–753, 2003.

21 Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. CRAM: Compressed random access memory. In *Automata, Languages, and Programming*, pages 510–521. Springer, 2012. doi:10.1007/978-3-642-31594-7_43.

22 Brian Kernighan and Dennis Ritchie. *The C Programming Language (1st Ed.)*. Prentice-Hall, 1978.

23 Shanika Kuruppu, Simon J. Puglisi, and Justin Zobel. Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval. In *Proc. 17th SPIRE*, pages 201–206, 2010.

24 Shanika Kuruppu, Simon J. Puglisi, and Justin Zobel. Optimized relative Lempel-Ziv compression of genomes. In *Proc. 34th ACSC*, pages 91–98, 2011.

25 Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. In *Proc. 31st STACS*, pages 506–517, 2014.

26 Stan Y. Liao, Srinivas Devadas, and Kurt Keutzer. A text-compression-based method for code size minimization in embedded systems. *ACM Trans. Design Autom. Electr. Syst.*, 4(1):12–38, 1999.

27 Stan Y. Liao, Srinivas Devadas, Kurt Keutzer, Steven W. K. Tjiang, and Albert Wang. Code optimization techniques in embedded DSP microprocessors. *Design Autom. for Emb. Sys.*, 3(1):59–73, 1998.

28 Gonzalo Navarro and Yakov Nekrich. Optimal dynamic sequence representations. In *Proc. 24th SODA*, pages 865–876, 2013.

29 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Alg.*, 10(3):16, 2014.

30 Mihai Pătraşcu and Erik D Demaine. Tight bounds for the partial-sums problem. In *Proc. 15th SODA*, pages 20–29, 2004.

31 Mihai Pătraşcu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *Proc. 55th FOCS*, pages 166–175, 2014.

32 Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct dynamic data structures. In *Algorithms and Data Structures*, pages 426–437. Springer, 2001. doi:10.1007/3-540-44634-6_39.

33 Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proc. 17th SODA*, pages 1230–1239, 2006.

34 James A. Storer and Thomas G. Szymanski. The macro model for data compression. In *Proc. 10th STOC*, pages 30–39, 1978.

35 James A Storer and Thomas G Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982.

36 Bjarne Stroustrup. *The C++ Programming Language: Special Edition (3rd Edition)*. Addison-Wesley, 2000. First edition from 1985.

37 Dan E. Willard. Examining computational geometry, van emde boas trees, and hashing from the perspective of the fusion tree. *SIAM J. Comput.*, 29(3):1030–1049, 2000.

Towards Plane Spanners of Degree 3

Ahmad Biniaz[*1], **Prosenjit Bose**[†2], **Jean-Lou De Carufel**[‡3], **Cyril Gavoille**[§4], **Anil Maheshwari**[¶5], **and Michiel Smid**[‖6]

1 Carleton University, Ottawa, Canada
ahmad.biniaz@gmail.com

2 Carleton University, Ottawa, Canada
jit@scs.carleton.ca

3 University of Ottawa, Canada
jdecaruf@uottawa.ca

4 University of Bordeaux, France
gavoille@labri.fr

5 Carleton University, Ottawa, Canada
anil@scs.carleton.ca

6 Carleton University, Ottawa, Canada
michiel@scs.carleton.ca

Abstract

Let S be a finite set of points in the plane that are in convex position. We present an algorithm that constructs a plane $\frac{3+4\pi}{3}$-spanner of S whose vertex degree is at most 3. Let Λ be the vertex set of a finite non-uniform rectangular lattice in the plane. We present an algorithm that constructs a plane $3\sqrt{2}$-spanner for Λ whose vertex degree is at most 3. For points that are in the plane and in general position, we show how to compute plane degree-3 spanners with a linear number of Steiner points.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases plane spanners, degree-3 spanners, convex position, non-uniform lattice

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.19

1 Introduction

Let S be a finite set of points in the plane. A *geometric graph* is a graph $G = (S, E)$ with vertex set S and edge set E consisting of line segments connecting pairs of vertices. The *length* (or *weight*) of any edge (p, q) in E is defined to be the Euclidean distance $|pq|$ between p and q. The *length* of any path in G is defined to be the sum of the lengths of the edges on this path. For any two vertices p and q of S, their *shortest-path distance* in G, denoted by $|pq|_G$, is a minimum length of any path in G between p and q. For a real number $t \geqslant 1$, the graph G is a *t-spanner* of S if for any two points p and q in S, $|pq|_G \leq t|pq|$. The smallest value of t for which G is a t-spanner is called the *stretch factor* of G. A large number of algorithms have been proposed for constructing t-spanners for any given point set; see [18].

[*] Supported by NSERC.
[†] Supported by NSERC.
[‡] Supported by NSERC.
[§] Supported by the ANR DESCARTES Project.
[¶] Supported by NSERC.
[‖] Supported by NSERC.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 19; pp. 19:1–19:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A *plane* spanner is a spanner whose edges do not cross each other. Chew [7] was the first to prove that plane spanners exist. Chew proved that the L_1-Delaunay triangulation of a finite point set has stretch factor at most $\sqrt{10} \approx 3.16$ (observe that lengths in this graph are measured in the Euclidean metric). In the journal version [8], Chew proves that the Delaunay triangulation based on a convex distance function defined by an equilateral triangle is a 2-spanner. Dobkin *et al.* [11] proved that the L_2-Delaunay triangulation is a t-spanner for $t = \frac{\pi(1+\sqrt{5})}{2} \approx 5.08$. Keil and Gutwin [16] improved the upper bound on the stretch factor to $t = \frac{4\pi}{3\sqrt{3}} \approx 2.42$. This was subsequently improved by Cui *et al.* [9] to $t = 2.33$ for the case when the point set is in convex position. Currently, the best result is due to Xia [19], who proved that t is less than 1.998. For points that are in convex position the current best upper bound on the stretch factor of plane spanners is 1.88 that was obtained by Amani *et al.* [1]. Regarding lower bounds, by considering the four vertices of a square, it is obvious that a plane t-spanner with $t < \sqrt{2}$ does not exist. Mulzer [17] has shown that every plane spanning graph for the vertices of a regular 21-gon has stretch factor at least 1.41611. Dumitrescu and Ghosh [13] improved the lower bound to 1.4308 for the vertices of a regular 23-gon.

The *degree* of a spanner is its maximum vertex degree. Das and Heffernan [10] showed the existence of spanners of maximum degree 3. Moreover, 3 is the lower bound on the maximum degree of a t-spanner, for any constant $t > 1$, because a Hamiltonian path through a set of points arranged in a grid has unbounded stretch factor; see [18] for more details. Even for points that are in convex position, 3 is a lower bound on the degree (see Kanj *et al.* [15]).

The problem of constructing bounded-degree spanners that are plane and have small stretch factor has received considerable attention (e.g., see [5, 6, 15]). Bonichon *et al.* [5] proved the existence of a degree 4 plane spanner with stretch factor 156.82. A simpler algorithm by Kanj *et al.* [15] constructs a degree 4 plane spanner with stretch factor 20; for points that are in convex position, this algorithm gives a plane spanner of degree at most 3 with the same stretch factor. Dumitrescu and Ghosh [12] considered plane spanners for uniform grids. For the infinite uniform square grid, they proved the existence of a plane spanner of degree 3 whose stretch factor is at most 2.607; the lower bound is $1 + \sqrt{2}$.

In this paper we consider bounded-degree plane spanners. In Section 3 we present an algorithm that computes a plane $\frac{3+4\pi}{3} \approx 5.189$-spanner of degree 3 for points in convex position. In Section 4 we consider finite non-uniform rectangular grids; we present an algorithm that computes a degree 3 plane spanner whose stretch factor is at most $3\sqrt{2} \approx 4.25$. In Section 5 we show that any plane t-spanner for points in the plane that are in general position, can be converted to a plane $(t + \epsilon)$-spanner of degree at most 3 that uses a linear number of Steiner points, where $\epsilon > 0$ is an arbitrary small constant.

2 Preliminaries

For any two points p and q in the plane let pq denote the line segment between p and q, $\ell(p, q)$ denote the line passing through p and q, $R(p \to q)$ denote the ray emanating from p and passing through q, and let $D(p, q)$ denote the closed disk that has pq as a diameter. Moreover, let $L(p, q)$ denote the lune of p and q, which is the intersection of the two closed disks of radius $|pq|$ that are centered at p and q.

Let S be a finite and non-empty set of points in the plane. We denote by $CH(S)$ the boundary of the convex hull of S. The *diameter* of S is the largest distance among the distances between all pairs of points of S. Any pair of points whose distance is equal to the diameter is called a *diametral pair*. Each point of diametral pair is called a *diametral point*.

Algorithm 1 MATCHING(C_1, C_2)

Input: Linearly separated chains C_1 and C_2 with the vertices of $C_1 \cup C_2$ in convex position.
Output: A matching between the points of C_1 and the points of C_2.

```
if $C_1 = \emptyset$ or $C_2 = \emptyset$ then return $\emptyset$
$(a, b) \leftarrow$ a closest pair of vertices between $C_1$ and $C_2$ such that $a \in C_1$ and $b \in C_2$
$C_1', C_1'' \leftarrow$ the two chains obtained by removing $a$ from $C_1$
$C_2', C_2'' \leftarrow$ the two chains obtained by removing $b$ from $C_2$
return $\{ab\} \cup$ MATCHING($C_1', C_2'$) $\cup$ MATCHING($C_1'', C_2''$)
```

▶ **Observation 1.** *Let S be a finite set of at least two points in the plane, and let $\{p, q\}$ be any diametral pair of S. Then, the points of S lie in $L(p, q)$.*

▶ **Theorem 2** (See Theorem 7.11 in [3])**.** *If C_1 and C_2 are convex polygonal regions with $C_1 \subseteq C_2$, then the length of the boundary of C_1 is at most the length of the boundary of C_2.*

▶ **Lemma 3** (Amani *et al.* [1])**.** *Let a, b, and c be three points in the plane, and let $\beta = \angle abc$. Then, $\frac{|ab|+|bc|}{|ac|} \leqslant \frac{1}{\sin(\beta/2)}$.*

▶ **Lemma 4** (Proof in the full version of the paper [4])**.** *Let a and b be two points in the plane. Let c be a point that is on the boundary or in the interior of $L(a, b)$. Then, $\angle acb \geqslant \frac{\pi}{3}$.*

3 Plane Spanners for Points in Convex Position

In this section we consider degree-3 plane spanners for points that are in convex position. Let S be a finite set of points in the plane that are in convex position. Consider the two chains that are obtained from $CH(S)$ by removing any two edges. Let τ be the larger stretch factor of these two chains. In Section 3.1 we present an algorithm that computes a plane $(2\tau + 1)$-spanner of maximum degree 3 for S. Based on that, in Section 3.2 we show how to compute a plane $\frac{3+4\pi}{3}$-spanner of maximum degree 3 for S. Moreover, we show that if S is centrally symmetric, then there exists a plane $(\pi + 1)$-spanner of degree 3 for S.

3.1 Spanner for Convex Double Chains

Let C_1 and C_2 be two chains of points in the plane that are separated by a straight line. Let S_1 and S_2 be the sets of vertices of C_1 and C_2, respectively, and assume that $S_1 \cup S_2$ is in convex position. Let τ be a real number. In this section we show that if the stretch factor of each of C_1 and C_2 is at most τ, then there exists a plane $(2\tau + 1)$-spanner for $S_1 \cup S_2$ whose maximum vertex degree is 3.

In order to build such a spanner, we join C_1 and C_2 by a set of edges that form a matching. Thus, the spanner consists of C_1, C_2, and a set E of edges such that each edge has one endpoint in C_1 and one endpoint in C_2. The set E is a matching, i.e., no two edges of E are incident to a same vertex. We show how to compute E recursively. Let (a, b) be the closest pair of vertices between C_1 and C_2; see Figure 1. Add this closest pair (a, b) to E. Then remove (a, b) from C_1 and C_2, and recurse on the two pairs of chains obtained on each side of $\ell(a, b)$. Stop the recursion as soon as one of the chains is empty. Given C_1 and C_2, the algorithm MATCHING computes a set E.

▶ **Theorem 5.** *Let $C_1 = (S_1, E_1)$ and $C_2 = (S_2, E_2)$ be two linearly separated chains of points in the plane, each with stretch factor at most τ, such that $S_1 \cup S_2$ is in convex*

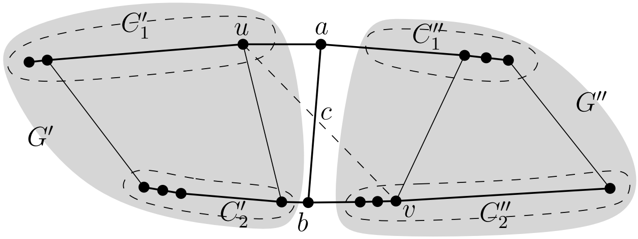


Figure 1 Illustration of the proof of Theorem 5.

position. Let E be the set of edges returned by algorithm $\textsc{Matching}(C_1, C_2)$*. Then, the graph $G = (S_1 \cup S_2, E_1 \cup E_2 \cup E)$ is a plane $(2\tau + 1)$-spanner for $S_1 \cup S_2$ in which the degree of each endpoint of C_1 and C_2 is at most 2 and every other vertex has degree at most 3.*

In the rest of this section we prove Theorem 5. The degree and planarity constraints follows from the algorithm. However, in the full version of the paper [4] we prove these constraints by induction. Now, we prove that the stretch factor of G is at most $2\tau + 1$. The proof is by induction on $\min\{|S_1|, |S_2|\}$. As for the base cases, if $|S_1| = 0$, then $G = C_2$ is a τ-spanner. If $|S_2| = 0$, then $G = C_1$ is a τ-spanner. Assume $|S_1| \geqslant 1$ and $|S_2| \geqslant 1$. Let ℓ be a line that separates C_1 and C_2. Without loss of generality assume ℓ is horizontal, C_1 is above ℓ, and C_2 is below ℓ. Let (a, b) be the pair of vertices selected by algorithm $\textsc{Matching}$, where (a, b) is a closest pair of vertices between C_1 and C_2 such that $a \in C_1$ and $b \in C_2$.
Let C'_1 and C''_1 be the left and right sub-chains of C_1, respectively, that are obtained by removing a; see Figure 1. We obtain C'_2 and C''_2 similarly. Let G' (resp. G'') be the spanner obtained for the vertices of C'_1 and C'_2 (resp. C''_1 and C''_2). By the induction hypothesis, G' (resp. G'') is a $(2\tau + 1)$-spanner for the vertices of $C'_1 \cup C'_2$ (resp. $C''_1 \cup C''_2$).

We are going to prove that for any two points $u, v \in S_1 \cup S_2$ we have $|uv|_G \leqslant (2\tau + 1)|uv|$. If both u and v belong to S_1, or both belong to S_2, then $|uv|_G \leqslant \tau|uv|$; this is valid because each of C_1 and C_2 has stretch factor at most τ. Assume $u \in S_1$ and $v \in S_2$. If $u, v \in G'$ or $u, v \in G''$ then, by the induction hypothesis, $|uv|_G \leqslant (2\tau + 1)|uv|$. Thus, it only remains to prove $|uv|_G \leqslant (2\tau + 1)|uv|$ for the following cases: (a) $u = a$ and $v \in C_2$, (b) $u \in C_1$ and $v = b$, (c) $u \in C'_1$ and $v \in C''_2$, and (d) $u \in C''_1$ and $v \in C'_2$. Because of symmetry we only prove items (a) and (c). The proofs are given in the following two lemmas.

▶ **Lemma 6.** *If $u = a$ and $v \in C_2$, then $|uv|_G \leqslant (2\tau + 1)|uv|$.*

Proof. Note that $|av|_G \leqslant |ab| + |bv|_{C_2} \leqslant |av| + \tau|bv|$, where the second inequality is valid since $|ab| \leqslant |av|$, by our choice of (a, b), and since $|bv|_{C_2} \leqslant \tau|bv|$, given that the stretch factor of C_2 is at most τ. It remains to prove that $|bv| \leqslant 2|av|$. By the triangle inequality we have $|bv| \leqslant |ab| + |av|$. Since $|ab| \leqslant |av|$, we have $|bv| \leqslant 2|av|$. ◀

▶ **Lemma 7.** *If $u \in C'_1$ and $v \in C''_2$, then $|uv|_G \leqslant (2\tau + 1)|uv|$.*

Proof. Since S is in convex position, the polygon Q formed by u, a, v, and b is convex and its vertices appear in the order u, a, v, b. Note that

$$|uv|_G \leqslant |ua|_{C_1} + |ab| + |bv|_{C_2} \leqslant \tau|ua| + |uv| + \tau|bv| = |uv| + \tau(|ua| + |bv|),$$

where the second inequality is valid since $|ab| \leqslant |uv|$, by our choice of (a, b), and since $|ua|_{C_1} \leqslant \tau|ua|$ and $|bv|_{C_2} \leqslant \tau|bv|$, given that the stretch factor of each of C_1 and C_2 is at most τ. It remains to prove that $|ua| + |bv| \leqslant 2|uv|$. Let c be the intersection point of ab and

uv; see Figure 1. By the triangle inequality, we have $|ua| \leqslant |uc| + |ca|$ and $|bv| \leqslant |bc| + |cv|$. It follows that $|ua| + |bv| \leqslant |uv| + |ab|$. Since $|ab| \leqslant |uv|$, we have $|ua| + |bv| \leqslant 2|uv|$. This completes the proof. ◀

3.2 Spanner for Points in Convex Position

In this section we show how to construct plane spanners of degree at most 3 for points that are in convex position.

▶ **Theorem 8.** *Let S be a finite set of points in the plane that is in convex position. Then, there exists a plane spanner for S whose stretch factor is at most $\frac{3+4\pi}{3}$ and whose vertex degree is at most 3.*

Proof. The proof is constructive; we present an algorithm that constructs such a spanner for S. The algorithm performs as follows. Let (p, q) be a diametral pair of S. Consider the convex hull of S. Let C_1 and C_2 be the two chains obtained from $CH(S)$ by removing p and q (and their incident edges). Note that C_1 and C_2 are separated by $\ell(p, q)$. Let G' be the graph on $S \setminus \{p, q\}$ that contains the edges of C_1, the edges of C_2, and the edges obtained by running algorithm $\textsc{Matching}(C_1, C_2)$. By Theorem 5, G' is plane and the endpoints of C_1 and C_2 have degree at most 2. We obtain a desired spanner, G, by connecting p and q, via their incident edges in $CH(S)$, to G'. In other words, $G = (S, E)$, where E is the union of the edges of $CH(S)$ and the edges of $\textsc{Matching}(C_1, C_2)$. A pseudo code for this construction is given in the full version of the paper [4].

Observe that G is plane. Moreover, all vertices of G have degree at most 3; p and q have degree 2. Now we show that the stretch factor of G is at most $\frac{3+4\pi}{3} \approx 5.19$. Note that G consists of $CH(S)$ and a matching which is returned by algorithm $\textsc{Matching}$. Since p and q are diametral points, then by a result of [1], for any point $s \in S \setminus \{p\}$ we have $|ps|_{CH(S)} \leqslant 1.88|ps|$. Since $CH(S) \subseteq G$, we have $|ps|_G \leqslant 1.88|ps|$. By symmetry, the same result holds for q and any point $s \in S \setminus \{q\}$. Since (p, q) is a diametral pair of S, both C_1 and C_2 are in $L(p, q)$. Based on this, in Theorem 11, we will see that both C_1 and C_2 have stretch factor at most $\frac{2\pi}{3}$. Then, by Theorem 5, the stretch factor of G' is at most $\frac{3+4\pi}{3}$. Since $G' \subset G$, for any two points $r, s \in S \setminus \{p, q\}$ we have $|rs|_G \leqslant \frac{3+4\pi}{3}|rs|$. Therefore, the stretch factor of G is at most $\frac{3+4\pi}{3}$. This completes the proof of the theorem. ◀

A point set S is said to be *centrally symmetric* (with respect to the origin), if for every point $p \in S$, point $-p$ also belongs to S.

▶ **Theorem 9.** *Let S be a finite centrally symmetric point set in the plane that is in convex position. Then, there exists a plane spanner for S whose stretch factor is at most $\pi + 1$ and whose vertex degree is at most 3.*

Proof. Let G be the graph obtained by the algorithm presented in the proof of Theorem 8. Recall that G is plane and its maximum vertex degree is at most 3. It remains to show that the stretch factor of G is at most $\pi + 1$. Let (p, q) be the diametral pair of S that is considered by this algorithm. Since S is centrally symmetric, all points of S are in $D(p, q)$. Based on this, in Theorem 10, we will see that both C_1 and C_2 have stretch factor at most $\frac{\pi}{2}$. Then Theorem 5 implies that the stretch factor of G is at most $\pi + 1$. ◀

3.3 Convex Chains with Diametral Endpoints

In this section we analyze the stretch factor of convex chains of points where their endpoints are a diametral pair. Let C be a chain of points. For any two points u and v on C let $\delta_C(u, v)$ denote the path between u and v on C, and let $|uv|_C$ denote the length of $\delta_C(u, v)$.

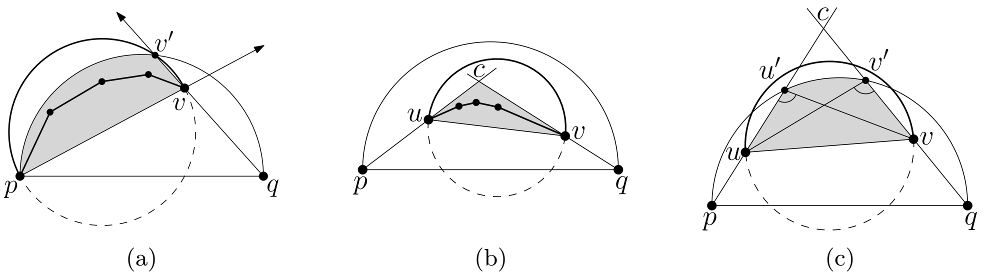


Figure 2 Proof of Theorem 10: the path $\delta_C(u,v)$ is inside the shaded regions.

▶ **Theorem 10.** *Let C be a convex chain with endpoints p and q. If C is in $D(p,q)$, then the stretch factor of C is at most $\frac{\pi}{2}$.*

Proof. Since C is convex, it is contained in a half-disk of $D(p,q)$, i.e., a half-disk with diameter pq. Let u and v be any two points of C. Let $\delta_C(u,v)$ be the path between u and v in C. We show that $\delta_C(u,v)$ is in $D(u,v)$. Then, by Theorem 2 the length of $\delta_C(u,v)$ is at most the length of the half-arc of $D(u,v)$, which is $\frac{\pi}{2}|uv|$. Without loss of generality assume that pq is horizontal, p is to the left of q, and C is above pq. Assume that u appears before v while traversing C from p to q. See Figure 2. We consider the following four cases.

- $u = p$ and $v = q$. Then $\delta_C(p,q) = C$ is in $D(p,q)$ by the hypothesis.
- $u = p$ and $v \neq q$. Let v' be the intersection point of $R(q\rightarrow v)$ with the boundary of $D(p,q)$. See Figure 2(a). Observe that $\angle pv'v = \angle pv'q = \frac{\pi}{2}$. Thus, v' is on the boundary of $D(p,v)$. Since two circles can intersect in at most two points, p and v' are the only intersection points of the boundaries of $D(p,q)$ and $D(p,v)$. Thus, the clockwise arc $\widehat{pv'}$ on the boundary of $D(p,q)$ is inside $D(p,v)$. Because of convexity, no point of $\delta_C(p,v)$ is to the right of $R(p\rightarrow v)$ or $R(q\rightarrow v)$. It follows that $\delta_C(p,v)$ is in $D(p,v)$.
- $u \neq p$ and $v = q$. The proof of this case is similar to the proof of the previous case.
- $u \neq p$ and $v \neq q$. Let c be the intersection point of $R(p\rightarrow u)$ and $R(q\rightarrow v)$. Because of convexity, $\delta_C(u,v)$ is in the triangle $\triangle ucv$. We look at two cases:
    - c is inside $D(p,q)$. See Figure 2(b). Note that $\angle ucv \geqslant \frac{\pi}{2}$. This implies that the point c, and consequently the triangle $\triangle ucv$, are in $D(u,v)$. Thus, $\delta_C(u,v)$ is inside $D(u,v)$.
    - c is outside $D(p,q)$. Let u' (resp. v') be the intersection point of $R(p\rightarrow u)$ (resp. $R(q\rightarrow v)$) with $D(p,q)$. Note that $\delta_C(u,v)$ is inside the shaded region of Figure 2(c). Observe that $\angle uv'v > \angle pv'q = \frac{\pi}{2}$, and $\angle uu'v > \angle pu'q = \frac{\pi}{2}$. Thus, both u' and v' are inside $D(u,v)$. Consequently, the clockwise arc $\widehat{u'v'}$ on the boundary of $D(p,q)$ is inside $D(u,v)$. Therefore, $\delta_C(u,v)$ is inside $D(u,v)$. ◀

▶ **Theorem 11** (Proof in the full version of the paper [4]). *Let C be a convex chain with endpoints p and q. If C is in $L(p,q)$, then the stretch factor of C is at most $\frac{2\pi}{3}$.*

4 Non-Uniform Rectangular Grid

In this section we build a plane spanner of degree at most three for the point set of the vertices of a non-uniform rectangular grid. In a finite non-uniform $m \times k$ grid, Λ, the vertices are arranged on the intersections of m horizontal and k vertical lines. The distances between the horizontal lines and the distances between the vertical lines are chosen arbitrary. The total number of vertices of Λ—the number of points of the underlying point set—is $n = m \cdot k$.

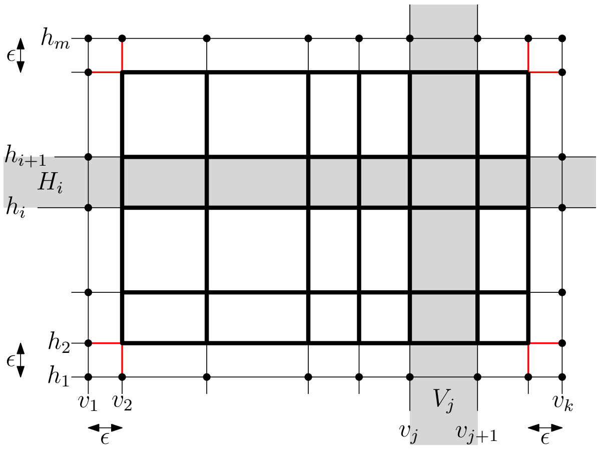


Figure 3 The augmented grid.

If $m \in \{1,2\}$ or $k \in \{1,2\}$ then Λ is a plane spanner whose maximum vertex degree is at most 3 and whose stretch factor is at most $\sqrt{2}$. Assume $m \geqslant 3$ and $k \geqslant 3$. We present an algorithm that constructs a degree-3 plane spanner, G, for the points of Λ. Note that Λ is a finite grid and has boundary vertices. In order to simplify the analysis and the proofs for boundary vertices, we augment Λ in the following way. We add four lines at distance ϵ, to the left, right, above and below Λ. We choose ϵ to be smaller than the distances among all pairs of vertical lines, and all pairs of horizontal lines of Λ. See Figure 3. For simplicity, in the rest of this section, we refer to the augmented lattice as Λ, and assume it has m horizontal and k vertical lines. Based on this assumption, the original lattice has $m-2$ horizontal lines and $k-2$ vertical lines.

Let $h_1, \dots, h_m$ be the horizontal lines of Λ from bottom to top. Similarly, let $v_1, \dots, v_k$ be the vertical lines of Λ from left to right. Note that Λ consists of $m-1$ horizontal slabs (rows) and $k-1$ vertical slabs (columns). Each horizontal slab H_i, with $1 \leqslant i < m$, is bounded by consecutive horizontal lines h_i and h_{i+1}. Each vertical slab V_j, with $1 \leqslant j < k$, is bounded by consecutive vertical lines v_j and v_{j+1}. See Figure 3. For each slab we define the *width* of that slab as the distance between the two parallel lines on its boundary. Let $p_{i,j}$, with $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant k$, be the vertex of Λ that is the intersection point of h_i and v_j. For each H_i, $1 \leqslant i < m$, we define $E(H_i) = \{(p_{i,j}, p_{i+1,j}) : 2 \leqslant j \leqslant k-1\}$ as the set of edges of H_i. Moreover, we define the set of *candidate edges* of H_i as follows:

$$CE(H_i) = \begin{cases} \{(p_{i,j}, p_{i+1,j}) : 2 \leqslant j \leqslant k-1 \text{ and } j \text{ is even}\} & \text{if } i \text{ is even,} \\ \{(p_{i,j}, p_{i+1,j}) : 2 \leqslant j \leqslant k-1 \text{ and } j \text{ is odd}\} & \text{if } i \text{ is odd.} \end{cases}$$

Similarly, for each V_j, $1 \leqslant j < k$, we define $E(V_j) = \{(p_{i,j}, p_{i,j+1}) : 2 \leqslant i \leqslant m-1\}$ as the set of edges of V_j. The set of *candidate edges* of V_j is defined as follows:

$$CE(V_j) = \begin{cases} \{(p_{i,j}, p_{i,j+1}) : 2 \leqslant i \leqslant m-1 \text{ and } i \text{ is even}\} & \text{if } j \text{ is even,} \\ \{(p_{i,j}, p_{i,j+1}) : 2 \leqslant i \leqslant m-1 \text{ and } i \text{ is odd}\} & \text{if } j \text{ is odd.} \end{cases}$$

See Figure 4(a). Informally speaking, the set of edges of each horizontal slab contains $k-2$ vertical edges of Λ that are on $v_2, \dots, v_{k-1}$, and the set of edges of each vertical slab contains $m-2$ horizontal edges of Λ that are on $h_2, \dots, h_{m-1}$. The boundary edges of Λ, i.e., the edges with both their endpoints on the boundary of Λ, do not belong to any of these sets.

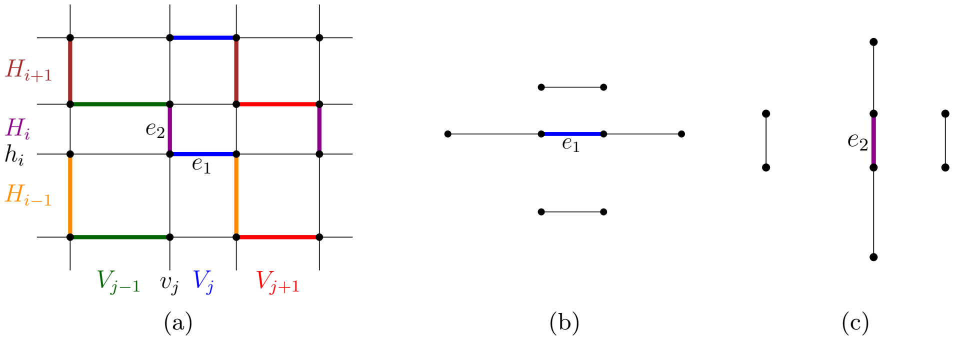


Figure 4 (a) Candidate edges in each slab are shown in their slab label color. The edges that are above, below, to the left, and to the right of a candidate edge, are not candidate in any of the slabs: (b) a horizontal candidate edge, and (c) a vertical candidate edge.

Every second vertical edge in H_i belongs to the set of candidate edges of H_i. The set of candidate edges of H_{i-1} (resp. H_{i+1}) contains every second vertical edge in H_{i-1} (rep. H_{i+1}) that is not adjacent to any candidate edge in H_i. The same observation applies on each V_i. Thus, if e is a candidate edge in some set, then the edges of Λ that are above, below, to the left, and to the right of e, are not candidate edges in any set. See Figures 4(b) and 4(c).

Now we describe the algorithm. We know that Λ is a $\sqrt{2}$-spanner of degree 4. The algorithm consists of two phases. In the first phase it removes some edges from Λ and constructs a graph G' whose largest vertex degree is 3 (G' may have large stretch factor). In the second phase the algorithm adds some edges to G' and constructs a graph G whose maximum degree is 3 and whose stretch factor is $3\sqrt{2}$. Note that $G' \subseteq G \subset \Lambda$. Refer to Figure 5 for an illustration of the two phases.

Phase 1 (Edge Deletion): In this phase, the algorithm iterates over all the slabs, $\{H_1, \dots, H_{m-1}, V_1, \dots, V_{k-1}\}$, in a non-increasing order of their widths. Let S be the current slab. The algorithm considers the candidate edges of S, i.e., all edges of $CE(S)$, from left to right if S is horizontal, and bottom-up if S is vertical (however, this ordering does not matter). The algorithm removes a candidate edge if it has at least one endpoint of degree 4. Let G' be the graph obtained at the end of this phase.

Phase 2 (Edge Insertion): Consider the graph G' obtained at the end of Phase 1. Let E' be the empty set. In the second phase, the algorithm iterates over all the slabs, $\{H_1, \dots, H_{m-1}, V_1, \dots, V_{k-1}\}$, in a non-decreasing order of their widths. Let S be the current slab. The algorithm considers all the edges of S, i.e., all edges of $E(S)$. Let $e = (a, b)$ be the current edge. The algorithms adds e to E' if both endpoints of e have degree 2 in $G' \cup E'$, i.e., $\deg_{G'}(a) + \deg_{E'}(a) = 2$ and $\deg_{G'}(b) + \deg_{E'}(b) = 2$. At the end of this phase, let G be the graph obtained by taking the union of G' and E'.

Consider the graph G obtained at the end of Phase 2. We show that G is a plane $3\sqrt{2}$-spanner of maximum degree 3 for Λ. Since the algorithm considers only the edges of Λ, then G is a subgraph of Λ and hence it is plane. As for the degree constraint, after Phase 1 the maximum degree in G' is 3. In Phase 2 we add edges between some vertices of degree 2 in G' (at most one edge per vertex) and hence no vertex of degree 4 can appear. Thus G has maximum degree 3. It only remains to show that G is a $3\sqrt{2}$-spanner. Before that, we review some properties of G' and G.

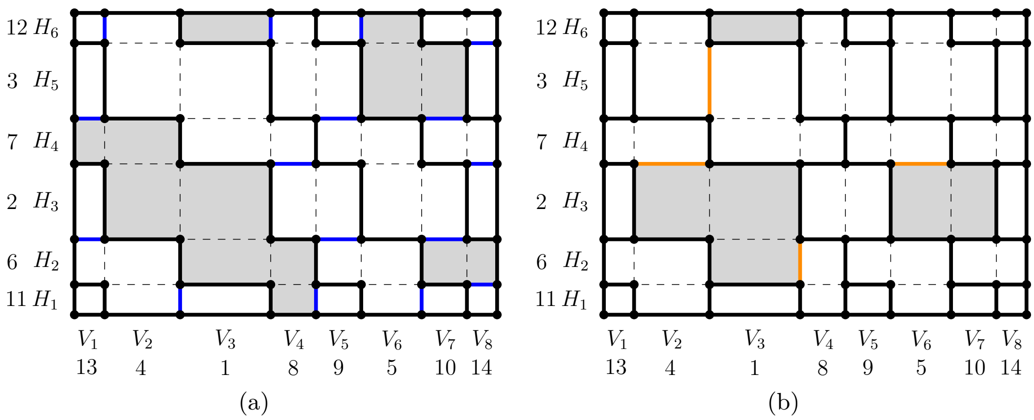


Figure 5 The numbers close to the slab labels show the order in which the slabs are considered in Phase 1. (a) The graph G' obtained at the end of Phase 1; its four types of faces are shaded (the blue edges are the candidate edges that have not been removed in Phase 1). (b) The graph G obtained at the end of Phase 2; its three types of faces are shaded (the orange edges have been added in Phase 2, and belong to E').

The candidate edges form stair-cases in Λ; see Figure 4(a). Moreover, the set of non-candidate edges (black edges of Figure 4(a)) also form stair-cases. Each internal vertex of Λ belongs to a staircase of candidate edges and a stair-case of non-candidate edges. Thus, in G', every vertex is on a stair-case of non-candidate edges that is connected to the boundary edges in both directions. Moreover, each of the stair-cases formed by candidate edges is surrounded by two stair-cases of non-candidate edges. Since G' contains all boundary edges, i.e., the edges with both endpoints on the boundary, each boundary vertex has degree at least 2 and at most 3 in G'. The edge deletion phase ensures that in G' there is no internal vertex of degree 4. Further, each internal vertex is incident on two candidate edges. Thus at the end of Phase 1, each internal vertex looses at most two edges, and hence has degree at least 2 in G'. Therefore we have the following observation.

▶ **Observation 12.** *The graph G' has the following properties.* (1) *G' contains all boundary edges of Λ,* (2) *G' is connected,* (3) *each vertex of G' has degree 2 or 3, and* (4) *each face in G' is either (see the shaded regions of Figure 5(a)):*

- *1-cell: consists of one cell of Λ, or*
- *2-cell: consists of two adjacent cells with the middle edge missing, or*
- *3-cell: consists of three adjacent cells which form an L-shape with the two middle edges missing (this L-shape might also be rotated), or*
- *stair-case: consists of more than three cells which form a stair-case with more than one vertex of degree two.*

We define $\{p_{1,1}, p_{m,1}, p_{1,k}, p_{m,k}\}$ as the set of *corner vertices* of Λ. We also define the set of *corner edges* of Λ as $\{(p_{1,2}, p_{2,2}), (p_{2,2}, p_{2,1}), (p_{1,k-1}, p_{2,k-1}), (p_{2,k-1}, p_{2,k}), (p_{m-1,1}, p_{m-1,2}), (p_{m-1,2}, p_{m,2}), (p_{m-1,k-1}, p_{m,k-1}), (p_{m-1,k-1}, p_{m-1,k})\}$. In Figure 3 the corner edges are in red. Note that each corner edge is adjacent to another corner edge. A *non-corner edge* is an edge of Λ which is not a corner edge.

▶ **Lemma 13** (Proof in the full version of the paper [4]). *All non-corner edges of Λ that are incident to a boundary vertex are in G'.*

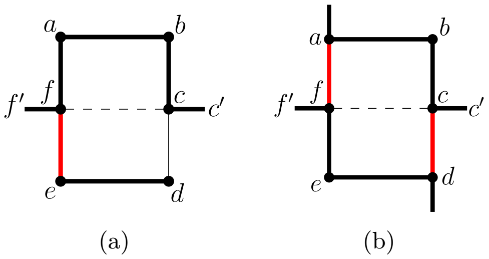


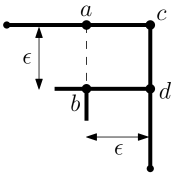
Figure 6 The edge (a, f) is a candidate edge (a) that is in G', and (b) that is not in G'.

By Lemma 13, any non-corner edge e of Λ that is not in G' has both its endpoints in the interior of Λ. That is, both endpoints of e have degree four in Λ. Based on this, and by our choice of candidate edges, we have the following observation; see Figures 4(b) and 4(c).

▶ **Observation 14.** *If a non-corner edge e is not in G', then the edges that are above, below, to the left, and to the right of e are in G', and hence in G.*

▶ **Lemma 15.** *Let (a, b) be a corner edge that is not in G. Then $|ab|_G = 3|ab|$.*

Proof. Recall that each corner edge is adjacent to another corner edge. Let (b, d) be the corner edge that is adjacent to (a, b). Let c be the corner vertex that is adjacent to a and d.



Since (a, c) and (c, d) are boundary edges, both of them are in G. We are going to show, by contradiction, that (b, d) is also in G. Assume $(b, d) \notin G$. If (b, d) was removed before (a, b), then at the moment the algorithm considers (a, b), both a and b have degree less than 4. Hence the algorithm would not remove (a, b); this contradicts the fact that $(a, b) \notin G$. If (a, b) was removed before (b, d) by a similar argument we get a contradiction. Thus $(b, d) \in G$. Note that $|ab| = |ac| = |cd| = |bd| = \epsilon$. Thus the length of the path $\delta = (a, c, d, b)$ is 3 times $|ab|$. ◀

At the end of Phase 2, all the stair-cases that have more than one vertex of degree two, have been broken into 2-cell and 3-cell faces. Thus we have the following observation.

▶ **Observation 16.** *Each face in G is either a 1-cell, a 2-cell, or a 3-cell (see the shaded faces in Figure 5(b)).*

▶ **Lemma 17.** *Let (f, c) be the missing edge of a 2-cell face in G. If (f, c) is a non-corner edge, then $|fc|_G \leqslant 3|fc|$.*

Proof. Let $F = (a, b, c, d, e, f)$ be the 2-cell face of G with the edge (f, c) is missing. Without loss of generality assume that (f, c) is horizontal, f is to the left of c, and a, b, c, d, e, f in the clockwise order of the vertices along F; see Figure 6.

Note that $|fc| = |ab| = |ed|$, $|af| = |bc|$, and $|fe| = |cd|$. Moreover, (a, b), (b, c), (d, e), and (e, f) are not candidate edges, hence they are in G' and in G, while (a, f) and (c, d) are candidate edges. Since (f, c) is a non-corner edge, in both G' and G, f is connected to a point f' and c is connected to a point c' where f' and c' are different from the vertices of F. By Observation 14, (f, f') and (c, c') are in G' and in G. Without loss of generality assume $|af| \leqslant |fe|$. We are going to prove that the length of the path (f, a, b, c) is at most three times $|fc|$. In order to prove this, we show that $|af| \leqslant |fc|$. The proof is by contradiction. Assume $|fc| < |af|$. For an edge $(u, v) \in \Lambda$, let S_{uv} be the slab containing (u, v) in its interior; if (u, v) is horizontal then S_{uv} is vertical, and vice versa. In Phase 1, the slabs S_{fe} and S_{af} are considered before S_{fc}, while in Phase 2, both are considered after S_{fc}. We consider two cases:

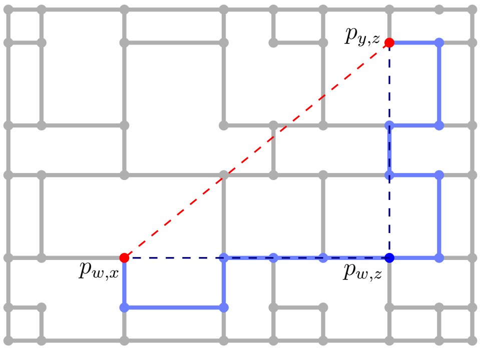
- $(a, f) \in G'$. See Figure 6(a). The reason why (a, f) was not removed is that both a and f had degree less than 4 at the moment the algorithm considered (a, f). At that moment the edge (f, c) was still in the graph. Thus, in order for f to have degree less than four, the edge (e, f) should have been removed before considering (a, f), which contradicts (e, f) being a non-candidate edge.
- $(a, f) \notin G'$. Thus (a, f) is added in Phase 2, and hence, both a and f have degree two in G'. See Figure 6(b). Recall that (c, d) is a candidate edge. Notice that $(c, d) \notin G'$ because at the moment the algorithm considered (c, d), the vertex c had degree 4, and hence (c, d) is removed. Thus (c, d) is added in Phase 2, implying that both c and d have degree two in G'. Therefore a, f, c, and d have degree two in G'. Since, in Phase 2, S_{fc} is considered before both S_{fe} and S_{af}, the edge (f, c) should have been inserted before considering (a, f) and (c, d). This contradicts the fact that (f, c) is not in G.

◀

▶ **Lemma 18** (Proof in the full version of the paper [4]). *Let (b, e) be a missing edge of a 3-cell face in G. If (b, e) is a non-corner edge, then $|be|_G \leqslant 3|be|$.*

▶ **Theorem 19.** *Let Λ be a finite non-uniform rectangular grid. Then, there exists a plane spanner for the point set of the vertices of Λ such that its degree is at most 3 and its stretch factor is at most $3\sqrt{2}$.*

Proof. Assume Λ has m rows and k columns.



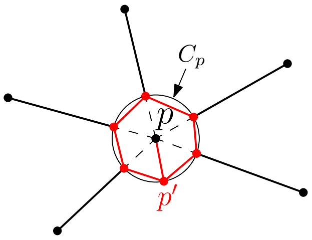
If $m \in \{1, 2\}$ or $k \in \{1, 2\}$, then Λ is a plane spanner whose degree is at most 3 and whose stretch factor is at most $\sqrt{2}$. Assume $m \geqslant 3$ and $k \geqslant 3$. Let $\overline{\Lambda}$ be the augmented lattice obtained from Λ as described in the beginning of this section. Let $\overline{G}$ be the graph obtained by the 2-phase algorithm described in this section. Then $\overline{G}$ is plane and its vertex degree is at most 3. By Lemmas 15, 17, and 18, for any edge $(a, b) \in \overline{\Lambda}$ that is not in $\overline{G}$, there exists a path in $\overline{G}$ whose length is at most 3 times $|ab|$. Now we are going to show that the stretch factor of $\overline{G}$ is at most $3\sqrt{2}$. Let $p_{w,x}$ and $p_{y,z}$ be any two vertices of $\overline{\Lambda}$. Consider the vertex $p_{w,z}$ in $\overline{\Lambda}$. By applying Lemmas 15, 17, and 18, in $\overline{G}$ there exists a path between $p_{w,x}$ and $p_{w,z}$ such that its length is at most 3 times $|p_{w,x}p_{w,z}|$. Similarly, in $\overline{G}$ there exists a path between $p_{y,z}$ and $p_{w,z}$ such that its length is at most 3 times $|p_{y,z}p_{w,z}|$. Since $|p_{w,x}p_{w,z}| + |p_{y,z}p_{w,z}| \leqslant \sqrt{2}|p_{w,x}p_{y,z}|$, we conclude that in $\overline{G}$ there exists a path between $p_{w,x}$ and $p_{y,z}$ that is passing through $p_{w,z}$ and whose length is at most $3\sqrt{2}$ times $|p_{w,x}p_{y,z}|$.

In order to obtain a spanner for Λ, we remove from $\overline{G}$ all the vertices of $\overline{\Lambda}$ that are not in Λ, as well as the edges incident to those vertices. Then we add all the missing boundary edges of Λ to the resulting graph. Let G be the graph that is obtained. As the boundary vertices of Λ have degree at most 3, G has vertex degree at most 3. Since all the boundary edges of Λ are in G, the stretch factor of G is not more than the stretch factor of $\overline{G}$. This completes the proof. ◀

5 Concluding Remarks

In order to obtain plane spanners with small stretch factor, one may think of adding Steiner points[1] to the point set and build a spanner on the augmented point set. In the L_1-metric, a plane 1-spanner of degree 4 can be computed by using $O(n \log n)$ Steiner points (see [14]). Arikati *et al.* [2] showed how to compute, in L_1-metric, a plane $(1 + \epsilon)$-spanner with $O(n)$ Steiner points, for any $\epsilon > 0$. Moreover, for the Euclidean metric, they showed how to construct a plane $(\sqrt{2} + \epsilon)$-spanner that uses $O(n)$ Steiner points and has degree 4.

Let S be a set of n points in the plane that is in general position; no three points are collinear. Let G be a plane t-spanner of S. We show that, for any $\epsilon > 0$, there exists a plane $(t + \epsilon)$-spanner G' for S with $O(n)$ Steiner points whose vertex degree is at most 3. We show how to construct such a spanner. Without loss of generality we assume that ϵ is smaller than the closest pair distance in S, otherwise we pick an ϵ' smaller than the closest pair distance, and construct a $(t + \epsilon')$-spanner, which is also a $(t + \epsilon)$-spanner.



For each point p of the point set S, consider a circle C_p with radius $\frac{\epsilon}{\pi n}$ that is centered at p. Introduce a Steiner point on each intersection point of C_p with the edges of G that are incident to p. Also, introduce a Steiner point p' on C_p that is different from these intersection points. Delete the part of the edges of G inside each circle C_p (each edge e of G turns into an edge e' of G' with endpoints on C_p). Add an edge from p to p', and add a cycle whose edges connect consecutive Steiner points on the boundary of C_p. This results in a degree-3 geometric plane graph G'. For each vertex of degree k in G, we added $k+1$ Steiner points in G'. Since G is planar, its total vertex degree is at most $6n - 12$. Thus, the number of Steiner points is $7n - 12$, in total (by a different construction of G' this can be reduced to $5n - 12$).

A path δ_{uv} between two vertices u and v in G can be turned into a path δ'_{uv} in G' as follows. For each point p in S corresponding to an internal vertex of δ_{uv} incident to two edges e_1 and e_2 of δ_{uv}, replace the part of e_1 and e_2 inside C_p by the shorter of the two paths along C_p connecting the corresponding Steiner points. Also, for each of point $p \in \{u, v\}$

[1] Ssome points in the plane that do not belong to the input point set.

incident to an edge e of δ_{uv} replace the part of e inside C_p with edge (p, p') together with the shorter of the two paths along C_p connecting p' and the Steiner point corresponding to e.

Note that $\frac{|\delta_{uv}|}{|uv|} \leqslant t$. Since the Steiner points are located at distance $\frac{\epsilon}{\pi n}$ from points of S, the length of the path along C_p replacing each vertex p of δ_{uv} is at most $\frac{\epsilon}{n}$. Since δ_{uv} has at most n vertices, the length of δ'_{uv} in G' is at most $|\delta_{uv}| + n \cdot \frac{\epsilon}{n}$. Thus, $\frac{|\delta'_{uv}|}{|uv|} \leqslant \frac{|\delta_{uv}|+\epsilon}{|uv|} \leqslant t + \epsilon$, is valid because ϵ is smaller than the closest pair distance in S, and hence smaller than $|uv|$.

Acknowledgements. Some parts of this work have been done at the *Fourth Annual Workshop on Geometry and Graphs*, held at the Bellairs Research Institute in Barbados, March 6–11, 2016. The authors are grateful to the organizers and to the participants of this workshop. Also, we would like to thank Günter Rote for his comments that simplified the proofs of Theorems 10 and 11. We also would like to thank the reviewers of the paper for their useful comments that simplified the proof of Theorem 17 and the spanner construction in Section 5.

References

1 Mahdi Amani, Ahmad Biniaz, Prosenjit Bose, Anil Maheshwari, Jean-Lou De Carufel, and Michiel Smid. A plane 1.88-spanner for points in convex position. In *Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 25:1–25:14, 2016. doi:10.4230/LIPIcs.SWAT.2016.25.

2 Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 4th European Symposium on Algorithms (ESA)*, pages 514–528, 1996.

3 Russel V. Benson. *Euclidean geometry and convexity.* McGraw-Hill, 1966.

4 Ahmad Biniaz, Prosenjit Bose, Jean-Lou De Carufel, Cyril Gavoille, Anil Maheshwari, and Michiel H. M. Smid. Towards plane spanners of degree 3. *CoRR*, abs/1606.08824, 2016.

5 Nicolas Bonichon, Iyad A. Kanj, Ljubomir Perković, and Ge Xia. There are plane spanners of degree 4 and moderate stretch factor. *Discrete & Computational Geometry*, 53(3):514–546, 2015.

6 Prosenjit Bose, Paz Carmi, and Lilach Chaitman-Yerushalmi. On bounded degree plane strong geometric spanners. *J. Discrete Algorithms*, 15:16–31, 2012.

7 L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the 2nd ACM Symposium on Computational Geometry*, pages 169–177, 1986.

8 L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39:205–219, 1989.

9 S. Cui, I. A. Kanj, and G. Xia. On the stretch factor of Delaunay triangulations of points in convex position. *Computational Geometry: Theory and Applications*, 44:104–109, 2011.

10 Gautam Das and Paul J. Heffernan. Constructing degree-3 spanners with other sparseness properties. *Int'l J. Found. Comput. Sci.*, 7(2):121–136, 1996.

11 D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5:399–407, 1990.

12 Adrian Dumitrescu and Anirban Ghosh. Lattice spanners of low degree. In *Proc. of the 2nd Int'l Conf. on Alg. and Disc. Appl. Math. (CALDAM)*, pages 152–163, 2016.

13 Adrian Dumitrescu and Anirban Ghosh. Lower bounds on the dilation of plane spanners. In *Proc. of the 2nd Int'l Conf. on Alg. and Disc. Appl. Math. (CALDAM)*, pages 139–151, 2016.

14 Joachim Gudmundsson, Oliver Klein, Christian Knauer, and Michiel Smid. Small Manhattan networks and algorithmic applications for the earth mover's distance. In *EuroCG'07*, pages 174–177, 2007.

15 Iyad A. Kanj, Ljubomir Perković, and Duru Türkoğlu. Degree four plane spanners: Simpler and better. In *Proc. of the 32nd Int'l Symp. on Comp. Geom.* (*SoCG*), pages 45:1–45:15, 2016. doi:10.4230/LIPIcs.SoCG.2016.45.

16 J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.

17 W. Mulzer. Minimum dilation triangulations for the regular n-gon. Master's thesis, Freie Universität Berlin, Germany, 2004.

18 G. Narasimhan and M. Smid. *Geometric Spanner Networks.* Cambridge University Press, Cambridge, UK, 2007.

19 G. Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM Journal on Computing*, 42:1620–1659, 2013.

Degree-Constrained Orientation of Maximum Satisfaction: Graph Classes and Parameterized Complexity*

Hans L. Bodlaender[1], Hirotaka Ono[2], and Yota Otachi[3]

1 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands; and**
Department of Mathematics and Computer Science, University of Technology Eindhoven, Eindhoven, The Netherlands
h.l.bodlaender@uu.nl

2 **Department of Economic Engineering, Kyushu University, Higashi-ku, Fukuoka 812-8581, Japan**
hirotaka@econ.kyushu-u.ac.jp

3 **School of Information Science, Japan Advanced Institute of Science and Technology, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan**
otachi@jaist.ac.jp

Abstract

The problem Max W-Light (Max W-Heavy) for an undirected graph is to assign a direction to each edge so that the number of vertices of outdegree at most W (resp. at least W) is maximized. It is known that these problems are NP-hard even for fixed W. For example, Max 0-Light is equivalent to the problem of finding a maximum independent set.

In this paper, we show that for any fixed constant W, Max W-Heavy can be solved in linear time for hereditary graph classes for which treewidth is bounded by a function of degeneracy. We show that such graph classes include chordal graphs, circular-arc graphs, d-trapezoid graphs, chordal bipartite graphs, and graphs of bounded clique-width.

To have a polynomial-time algorithm for Max W-Light, we need an additional condition of a polynomial upper bound on the number of potential maximal cliques to apply the metatheorem by Fomin, Todinca, and Villanger [SIAM J. Comput., 44(1):57–87, 2015]. The aforementioned graph classes, except bounded clique-width graphs, satisfy such a condition. For graphs of bounded clique-width, we present a dynamic programming approach not using the metatheorem to show that it is actually polynomial-time solvable for this graph class too.

We also study the parameterized complexity of the problems and show some tractability and intractability results.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases orientation, graph class, width parameter, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.20

1 Introduction

Let $G = (V, E)$ be an undirected graph. An *orientation of G* is a function that maps each undirected edge $\{u, v\} \in E$ to one of the two possible directed edges (u, v) and (v, u). For

[*] Partially supported by NETWORKS project and by MEXT/JSPS KAKENHI grant numbers 24106004, 24220003, 25730003, 26540005. The third author was partially supported by FY 2015 Researcher Exchange Program between JSPS and NSERC.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 20; pp. 20:1–20:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

any orientation Λ of G, define $\Lambda(E) = \bigcup_{e \in E}\{\Lambda(e)\}$ and let $\Lambda(G)$ denote the directed graph $(V, \Lambda(E))$. For any vertex $u \in V$, the *outdegree of u under* Λ is defined as $d^+_\Lambda(u) = |\{(u, v) : (u, v) \in \Lambda(E)\}|$, i.e., the number of outgoing edges from u in $\Lambda(G)$. For any non-negative integer W, a vertex $u \in V$ is called *W-light in* $\Lambda(G)$ if $d^+_\Lambda(u) \leq W$, and *W-heavy in* $\Lambda(G)$ if $d^+_\Lambda(u) \geq W$.

For any fixed integer $W \geq 0$, the following optimization problems (introduced in [3], see also [4]) are defined, where the input is an undirected graph $G = (V, E)$:

- Max W-Light: Output an orientation Λ of G
  such that $\left|\{u \in V \,:\, d^+_\Lambda(u) \leq W\}\right|$ is maximized.
- Max W-Heavy: Output an orientation Λ of G
  such that $\left|\{u \in V \,:\, d^+_\Lambda(u) \geq W\}\right|$ is maximized.

Symmetrically, we can consider the following problems:

- Min W-Light: Output an orientation Λ of G
  such that $\left|\{u \in V \,:\, d^+_\Lambda(u) \leq W\}\right|$ is minimized.
- Min W-Heavy: Output an orientation Λ of G
  such that $\left|\{u \in V \,:\, d^+_\Lambda(u) \geq W\}\right|$ is minimized.

Observe that Max W-Light (resp., Max W-Heavy) and Min $(W+1)$-Heavy (resp., Min $(W-1)$-Light) are *supplementary problems* in the sense that an exact algorithm for one gives an exact algorithm for the other, though their approximability properties and fixed-parameter tractability may differ. Since this paper mainly focuses on the polynomial-time solvability, we consider only Max W-Light and Max W-Heavy. [1]

It is shown in [3] that Max W-Light is NP-hard for any fixed $W \geq 0$, and Max W-Heavy is NP-hard for any fixed $W \geq 3$. They also show that for $W \leq 1$ Max W-Heavy can be solved in polynomial time. Recently Khoshkhah [23] has closed the gap by showing that Max 2-Heavy can be solved in polynomial time.

For these problems, the same authors of [3] investigate the approximability [4]. They got comprehensive results on the approximability of the problems. Due to the work, the general (in)approximability of the problems is well understood. In this paper, we thus investigate the problem from another aspect, that is, graph classes. For the two problems Max W-Light and Max W-Heavy, we take similar but slightly different approaches.

The main tool for Max W-Light is the metatheorem of Fomin, Todinca, and Villanger [16] that can be seen as an extension of Courcelle's theorem [1, 12]. It provides a polynomial-time algorithm for finding a maximum induced subgraph of bounded treewidth satisfying a counting monadic second-order logic formula from a given graph with polynomially many potential maximal cliques. We show that if a hereditary graph class has a polynomial upper bound on the number of potential maximal cliques and has a function depending only on degeneracy as an upper bound of treewidth, then the metatheorem of Fomin et al. can be applied to Max W-Light.

Similarly, for Max W-Heavy, we consider hereditary graph classes with treewidth bounded by a function of degeneracy. However, we do not require polynomial upper bounds on the number of potential maximal cliques. We first show that the problem for graphs of bounded treewidth can be solved in linear time. Next we present a linear-time reduction from graphs with a function of degeneracy as an upper bound of treewidth to graphs of bounded treewidth. Combining these results, we obtain a linear-time algorithm for Max W-Heavy on graph classes with the aforementioned property.

[1] We consider parameterized complexity in Section 5 where the equivalence does not hold.

We then show that our algorithms can be applied to several well-known graph classes. It is known that chordal graphs, circular-arc graphs, d-trapezoid graphs, and chordal bipartite graphs have polynomial upper bounds on the number of potential maximal cliques (see Section 4). We show that these hereditary graph classes have functions of degeneracy as upper bounds on treewidth, and thus our algorithms can be applied. Additionally, we observe that graphs of bounded clique-width admit a function of degeneracy as an upper bounded of treewidth, and thus Max W-Heavy can be solved in linear time. To show that Max W-Light can be solved in polynomial time for graphs of bounded clique-width, we present a dynamic programming based algorithm.

We also consider the parameterized complexity of the problems. We show that for any fixed W, Max W-Light is W[1]-complete, while Max W-Heavy admits a kernel of $O(Wk)$ vertices, where the parameter k is the solution size.

1.1 Related work

Graph orientations that optimize certain objective functions involving the resulting directed graph or that satisfy some special property such as acyclicity [39] or k-edge connectivity [10, 33, 37] have many applications to graph theory, combinatorial optimization, scheduling (load balancing), resource allocation, and efficient data structures. For example, an orientation that minimizes the maximum outdegree [5, 9, 40] can be used to support fast vertex adjacency queries in a sparse graph by storing each edge in exactly one of its two incident vertices' adjacency lists while ensuring that all adjacency lists are short [9]. There are many optimization criteria for graph orientation other than these. See [2] or Chapter 61 in [38] for more details and additional references.

On the other hand, degree-constrained graph orientations [17, 18, 21, 29] arise when a degree lower bound $W^l(v)$ and a degree upper bound $W^u(v)$ for each vertex v in the graph are specified in advance or as part of the input, and the outdegree of v in any valid graph orientation is required to lie in the interval $[W^l(v), \dots, W^u(v)]$. Obviously, a graph does not always have such an orientation, and in this case, one might want to compute an orientation that best fits the outdegree constraints according to some well-defined criteria [2, 3]. In case $W^l(v) = 0$ and $W^u(v) = W$ for every vertex v in the input graph, where W is a non-negative integer, and the objective is to maximize (resp., minimize) the number of vertices that satisfy (resp., violate) the outdegree constraints, then we obtain Max W-Light (resp., Min $(W + 1)$-Heavy). Similarly, if $W^l(v) = W$ and $W^u(v) = \infty$ for every vertex v in the input graph, then we obtain Max W-Heavy and Min $(W - 1)$-Light.

Another related problem is to find a maximum vertex set that induces a subgraph of bounded degeneracy. (See the next section for the definition of degeneracy.) This problem can be seen as a variant of Max W-Light, where we can use acyclic orientations only. This problem is studied in the context of parameterized [31] and exact [36] computation. Concerning graph classes, we can obtain a result similar to the one for Max W-Light as we observe in the final section of this paper.

2 Preliminaries

The *degree of* u *in* G is $d_G(u) = |N_G(u)|$. We define $\delta(G) = \min\{d_G(u) : u \in V(G)\}$. The *degeneracy* of a graph G, denoted by $\hat{\delta}(G)$, is the maximum of the minimum degrees over all induced subgraphs of G. Let $(v_1, \dots, v_n)$ be an ordering on $V(G)$ such that $d_{G_i}(v_i) = \delta(G_i)$, where $G_i = G[\{v_j : j \ge i\}]$. It is known that such an ordering can be computed in linear time and that $\hat{\delta}(G) = \max_{1 \le i \le n} \delta(G_i)$ [32]. For any $U \subseteq V(G)$, the subgraph induced by

U is denoted by $G[U]$. If $G[U]$ is a complete graph, then U is a *clique* of G. The size of a maximum clique in G is denoted by $\omega(G)$. Let $\omega_{\mathsf{b}}(G)$ be the maximum integer k such that G has a subgraph isomorphic to the complete bipartite graph $K_{k,k}$. From the definition, $\omega(G) - 1$ and $\omega_{\mathsf{b}}(G)$ are lower bounds of $\hat{\delta}(G)$. A class $\mathcal{C}$ of graphs is *hereditary* if $\mathcal{C}$ is closed under taking induced subgraphs.

For an integer $W \geq 0$, an orientation of a graph is called a *W-light orientation* if the maximum outdegree is at most W. If a W-light orientation exists, we say that the graph is *W-light orientable.* By replacing “at most” with “at least” in these definitions, we similarly define *W-heavy orientations* and *W-heavy orientable* graphs.

2.1 Minimal triangulations and potential maximal cliques

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i : i \in I\}, T = (I, F))$ such that each X_i, called a *bag*, is a subset of V, and T is a tree such that

- for each $v \in V$, there is $i \in I$ with $v \in X_i$;
- for each $\{u, v\} \in E$, there is $i \in I$ with $u, v \in X_i$;
- for $i, j, k \in I$, if j is on the i, k-path in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree-decomposition is the size of a maximum bag minus 1. A graph has *treewidth* at most t if and only if it has a tree-decomposition of width at most t. We denote the treewidth of G by $\mathbf{tw}(G)$.

A graph is *chordal* (or *triangulated*) if it has no induced cycle of length 4 or more. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $G' = (V, E')$ such that $E \subseteq E'$. A triangulation G' of G is *minimal* if no proper subgraph of G' is a triangulation of G. It is known that the treewidth of G is the minimum integer t such that there is a (minimal) triangulation H of G with the maximum clique size $t + 1$. A vertex set $P \subseteq V(G)$ is a *potential maximal clique* of G if P is a maximal clique in some minimal triangulation of G. The set of all potential maximal cliques of G is denoted by $\mathit{\Pi}_G$. A vertex set $S \subseteq V(G)$ is an *a, b-separator* for $a, b \in V(G)$ if a and b are in different components in $G - S$. An a, b-separator is *minimal* if no proper subset of it is an a, b-separator. A vertex set is a *minimal separator* if it is a minimal a, b-separator for some pair a, b. The set of all minimal separators of G is denoted by $\mathit{\Delta}_G$. By the following proposition, graphs have a polynomial number of minimal separators if and only if they have a polynomial number of potential maximal cliques.

▶ **Proposition 2.1** (Bouchitté and Todinca [8]). *For every n-vertex graph G, it holds that $|\mathit{\Delta}_G|/n \leq |\mathit{\Pi}_G| \leq n|\mathit{\Delta}_G|^2 + n|\mathit{\Delta}_G| + 1$.*

3 Metatheorems

In this section we present metatheorems for Max W-Light and Max W-Heavy. We apply them to some well-studied graph classes in the next section.

We now introduce the *monadic second-order logic* (MSO) of graphs. The syntax of MSO of graphs includes (i) the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, (ii) variables for vertices, edges, vertex sets, and edge sets, (iii) the quantifiers $\forall$ and $\exists$ applicable to these variables, and (iv) the following binary relations:

- $u \in U$ for a vertex variable u and a vertex set variable U;
- $d \in D$ for an edge variable d and an edge set variable D;
- $\mathsf{inc}(d, u)$ for an edge variable d and a vertex variable u, where the interpretation is that d is incident with u;
- equality of variables.

In the *counting monadic second-order logic* (CMSO) of graphs, we have an additional sentence of checking the cardinality of a set modulo some constant.

▶ **Lemma 3.1.** *For any fixed W,* Max W-Heavy *and* Max W-Light *for graphs of bounded treewidth can be expressed in an optimization version of MSO and thus solved in linear time.*

Proof (sketch). Let $G = (V, E)$ be a graph of treewidth at most k. It is known that for such a graph, an edge orientation can be expressed in MSO by a proper coloring $\gamma \colon V \to \{1, \dots, k+1\}$ and an edge set $F \subseteq E$ [6].

Let $\mathsf{prop\text{-}col}(V_1, \dots, V_{k+1})$ be an MSO formula that means $V_1, \dots, V_{k+1}$ give a proper $k + 1$ coloring of G. For an edge $e \in E$ and a vertex $v \in V$, there is an MSO formula $\mathsf{out}_{V_1,\dots,V_{k+1},F}(e, v)$ that means e is an out-going edge from v. Under the orientation represented by $(V_1, \dots, V_{k+1})$ and F, W-heaviness and W-lightness of a vertex can be expressed in MSO. Let $W\text{-}\mathsf{heavy}_{V_1,\dots,V_{k+1},F}(v)$ and $W\text{-}\mathsf{light}_{V_1,\dots,V_{k+1},F}(v)$ be such formulas. The problems are equivalent to finding a maximum vertex set X in the following formulas:

$$\exists V_1, \dots, V_{k+1}, \exists F \left(\mathsf{prop\text{-}col}(V_1, \dots, V_k) \wedge \forall v \in X \left(W\text{-}\mathsf{heavy}_{V_1,\dots,V_{k+1},F}(v)\right)\right),$$

$$\exists V_1, \dots, V_{k+1}, \exists F \left(\mathsf{prop\text{-}col}(V_1, \dots, V_k) \wedge \forall v \in X \left(W\text{-}\mathsf{light}_{V_1,\dots,V_{k+1},F}(v)\right)\right).$$

It is known that for a fixed MSO formula on a graph of bounded treewidth, one can find in linear time a maximum vertex subset satisfying the formula (see [1, 12]). ◀

▶ **Corollary 3.2.** *For fixed W and k, the property of being W-light orientable can be expressed in MSO for graphs of treewidth at most k.*

3.1 Max W-Light

We can see that the problem of finding a maximum W-light orientable induced subgraph is polynomially equivalent to Max W-Light.

▶ **Lemma 3.3.** *A graph G has a W-light orientable induced subgraph of at least k vertices if and only if the edges of G can be oriented so that at least k vertices have outdegree at most W. Furthermore, if a maximum W-light orientable induced subgraph of G can be found in $O(f(m, n))$ time, then* Max W-Light *can be solved in $O(f(m, n) + m^{1.5})$ time, where m and n are the numbers of edges and vertices in G, respectively.*

Recently, Fomin, Todinca, and Villanger [16] have presented the following metatheorem.

▶ **Proposition 3.4** (Fomin, Todinca, and Villanger [16])**.** *For any fixed t and a CMSO-expressible property $\mathcal{P}$, the following problem can be solved in polynomial time for any class of graphs with a polynomial number of potential maximal cliques: Given a graph G, find a maximum induced subgraph H of treewidth at most t that has the property $\mathcal{P}$.*

This metatheorem is quite powerful and allows us to solve many problems for graphs with polynomially many potential maximal cliques. However, we cannot apply it to our problem Max W-Light in general because W-light orientable graphs may have large treewidth. For example, grid graphs are 2-light orientable but have unbounded treewidth.

In the following, we show that with an additional restriction to graph classes, we can apply the metatheorem of Fomin, Todinca, and Villanger to Max W-Light.

▶ **Lemma 3.5.** *Every W-light orientable graph has degeneracy at most $2W$.*

▶ **Theorem 3.6.** *For any fixed W,* Max W-Light *can be solved in polynomial time for a hereditary graph class $\mathcal{C}$ with a polynomial number of potential maximal cliques if the treewidth of each graph in $\mathcal{C}$ is bounded from above by a function of its degeneracy.*

Proof. Let f be a function such that $\mathbf{tw}(G) \leq f(\hat{\delta}(G))$ for each $G \in \mathcal{C}$. By Lemma 3.5, a W-light orientable graph in $\mathcal{C}$ has treewidth at most $f(2W)$. Since $\mathcal{C}$ is hereditary, a maximum W-light orientable induced subgraph of a graph in $\mathcal{C}$ can be found in polynomial time by Proposition 3.4 and Corollary 3.2. Now, by Lemma 3.3, the theorem follows. ◀

3.2 Max W-Heavy

Unlike Max W-Light, the problem Max W-Heavy is not equivalent to the problem of finding a maximum orientable induced subgraph. We here present a way of directly finding an orientation with as many W-heavy vertices as possible for graphs with treewidth bounded by a function of degeneracy.

▶ **Proposition 3.7** ([4]). *Every graph of minimum degree at least $2W + 1$ is W-heavy orientable and a W-heavy orientation of it can be found in linear time.*

▶ **Theorem 3.8.** *For any fixed W,* Max W-Heavy *can be solved in linear time for a hereditary graph class $\mathcal{C}$ if the treewidth of each graph in $\mathcal{C}$ is bounded from above by a function of its degeneracy.*

Proof. Let f be a function such that $\mathbf{tw}(G) \leq f(\hat{\delta}(G))$ for each $G \in \mathcal{C}$. Let $G \in \mathcal{C}$ be a graph with n vertices. Let $(v_1, v_2, \ldots, v_n)$ be an ordering of $V(G)$ such that for each i, the vertex v_i has the minimum degree in G_i, where $G_i = G[\{v_j : i \leq j \leq n\}]$. Let h be the first index such that $d_{G_h}(v_h) \geq 2W + 1$. If there is no such index, we set $h = n + 1$.

Let $H = G[\{v_j : 1 \leq j < h\}]$. Since $\mathcal{C}$ is hereditary, we have $H \in \mathcal{C}$, and thus $\mathbf{tw}(H) \leq f(\hat{\delta}(H)) \leq f(2W)$. We obtain H' from H as follows: add a clique C of size $2W+1$; for each vertex v in H, add edges from v to arbitrarily chosen $d_G(v) - d_H(v)$ vertices in C. It holds that $\mathbf{tw}(H') \leq \mathbf{tw}(H) + |C| \leq f(2W) + 2W + 1$.

By Lemma 3.1, an orientation Λ' of H' with the maximum number of W-heavy vertices can be found in linear time. Note that all vertices in C are W-heavy under Λ' even in $H'[C]$. Otherwise, by Proposition 3.7, we can change the directions of edges in $H'[C]$ so that all vertices in C become W-heavy. Since this modification does not decrease the outdegree of any vertex in $V(H)$, the new orientation has strictly more W-heavy vertices than Λ'. This contradicts the optimality of Λ'.

Let Λ'' be a W-heavy orientation of $G_h = G[\{v_h, \ldots, v_n\}]$. By Proposition 3.7, such an orientation can be found in linear time. We next construct an orientation Λ of G from Λ' and Λ'' as follows: for each edge in $E(H)$ or $E(G_h)$, we use the direction in Λ' or Λ'', respectively; for each edge between $V(H)$ and $V(G_h)$, we use the direction from $V(H)$ to $V(G_h)$. All vertices in $V(G_h)$ are W-heavy in G under Λ. Under Λ, each vertex in $V(H)$ has at least as many out-neighbors as under Λ'. Thus a vertex in $V(H)$ is W-heavy in G under Λ if it is W-heavy in H' under Λ'.

We now show the optimality of Λ. Suppose to the contrary that there is an orientation Λ_{OPT} of G with strictly more W-heavy vertices than Λ. Let F and F_{OPT} be the W-heavy vertices in $V(H)$ under Λ and Λ_{OPT}, respectively. Since the vertices in $V(G_h)$ are W-heavy under Λ, we have $|F| < |F_{\mathsf{OPT}}|$. Now let Λ'_{OPT} be an orientation of H' such that the edges in H are oriented as in Λ_{OPT}, the edges between $V(H)$ and C are oriented from $V(H)$ to C, and the edges in $H[C]$ are oriented so that all the vertices in C become W-heavy. Then, at least $|C| + |F_{\mathsf{OPT}}| > |C| + |F|$ vertices are W-heavy in H' under Λ'_{OPT}. This contradicts the optimality of Λ' since at most $|C| + |F|$ vertices are W-heavy in H' under Λ'. ◀

4 Graph classes

In this section, we show that Theorems 3.6 and 3.8 can be applied to several important graph classes. More precisely, we show the following theorems.

▶ **Theorem 4.1.** *For any fixed W,* Max W-Light *can be solved in polynomial time for the classes of chordal graphs, d-trapezoid graphs, circular-arc graphs, chordal bipartite graphs, and graphs of bounded clique-width.*

▶ **Theorem 4.2.** *For any fixed W,* Max W-Heavy *can be solved in linear time for the classes of chordal graphs, d-trapezoid graphs, circular-arc graphs, chordal bipartite graphs, and graphs of bounded clique-width.*

To prove Theorems 4.1 and 4.2, we show for each graph class that it satisfies conditions of Theorems 3.6 and 3.8 in the following subsections. To solve Max W-Light for graphs of bounded clique-width, we present a direct solution as we cannot apply the metatheorem. Note that all graph classes studied in this section are hereditary.

4.1 Chordal graphs, d-trapezoid graphs, and circular-arc graphs

It is well known that a chordal graph of n vertices has at most n maximal cliques (see [22]). Since a chordal graph is the unique minimal triangulation of itself, the number of potential maximal cliques is at most n for every n-vertex chordal graph. From the definition of chordal graphs, the following equality follows.

▶ **Proposition 4.3** (Folklore). *For every chordal graph G, $\mathbf{tw}(G) = \hat{\delta}(G) = \omega(G) - 1$.*

The *co-comparability graph* of a partial order $(V, \prec)$ is a graph with the vertex set V in which two vertices u and v are adjacent if and only if they are incomparable, that is, $u \not\prec v$ and $v \not\prec u$. A partial order $(V, \prec)$ is an *interval order* if each element $v \in V$ can be represented by an interval $[l_v, r_v]$ such that $u \prec v$ if and only if $r_u < l_v$. A graph is a *d-trapezoid graph* if it is the co-comparability graph of a partial order defined as the intersection of d interval orders [7]. It is known that every d-trapezoid graph of n vertices has at most $(2n-3)^{d-1}$ minimal separators [28]. Habib and Möhring showed in the proof of Theorem 3.4 in [20] that for every d-trapezoid graph G, $\mathbf{tw}(G) \le 4d \cdot \omega_{\mathsf{b}}(G) - 1$. This gives the following fact as a direct corollary.

▶ **Proposition 4.4** ([20]). *For every d-trapezoid graph G, $\mathbf{tw}(G) \le 4d \cdot \hat{\delta}(G) - 1$.*

A graph is a *circular-arc graph* if it is the intersection graph of arcs on a circle. Every n-vertex circular-arc graph has at most $2n^2 - 3n$ minimal separators [26]. A graph is an *interval graph* if it is the intersection graph of intervals on a line. From the definition, every interval graph is a circular-arc graph. Also, every interval graph is a chordal graph [30].

▶ **Lemma 4.5.** *For every circular-arc graph G, $\mathbf{tw}(G) \le 2\hat{\delta}(G)$.*

4.2 Chordal bipartite graphs

A bipartite graph is a *chordal bipartite graph* if it has no induced cycle of length 6 or more. Every chordal bipartite graph has $O(m+n)$ minimal separators [27]. We can show that for every chordal bipartite graph G, $\mathbf{tw}(G) \le 2\hat{\delta}(G) - 1$. The proof is a bit more involved than the ones in the previous subsection. We use the techniques developed by Kloks and Kratsch [25] for computing the treewidth of a chordal bipartite graph exactly.

▶ **Theorem 4.6.** *For every chordal bipartite graph G, $\mathbf{tw}(G) \le 2 \cdot \omega_{\mathsf{b}}(G) - 1$.*

4.3 Graphs of bounded clique-width

A *k-expression* is a rooted binary tree such that

- each leaf has label $\circ_i$ for some $i \in \{1, \ldots, k\}$,
- each node with one child has a label $\rho_{i,j}$ or $\eta_{i,j}$ ($i, j \in \{1, \ldots, k\}$, $i \neq j$), and
- each node with two children has a label $\cup$.

Each node in a k-expression represents a vertex-labeled graph as follows:

- a $\circ_i$-node represents a graph with one vertex of label i;
- a $\cup$-node represents the disjoint union of the labeled graphs represented by its children;
- a $\rho_{i,j}$-node represents the labeled graph obtained from the one represented by its child by relabeling the label-i vertices with label j;
- an $\eta_{i,j}$-node represents the labeled graph obtained from the one represented by its child by adding edges between the label-i vertices and the label-j vertices.

A k-expression represents the graph represented by its root. The *clique-width* of a graph G, denoted by $\mathbf{cw}(G)$, is the minimum integer k such that there is a k-expression representing a graph isomorphic to G.

It is known that graphs of bounded treewidth have bounded clique-width [11]. The converse is not true in general. For example, the complete graph K_n ($n \geq 2$) has clique-width 2 and treewidth $n - 1$. On the other hand, the following bound is known for graphs with no large complete bipartite subgraphs.

▶ **Proposition 4.7** (Gurski and Wanke [19]). *For every graph G of clique-width at most k, $\mathbf{tw}(G) \leq 3k \cdot \omega_{\mathsf{b}}(G) - 1$.*

The proposition above with Theorem 3.8 imply that Max W-Heavy can be solved in linear time for graphs of bounded clique-width. However, we cannot apply Theorem 3.6 since graphs of bounded clique-width may have a super-polynomial number of potential maximal cliques. In the rest of this section, we directly show that Max W-Light is polynomial-time solvable for graphs of bounded clique-width. A k-expression of a graph is *irredundant* if for each edge $\{u, v\}$, there is exactly one node $\eta_{i,j}$ that adds the edge between u and v. We will show that:

▶ **Theorem 4.8.** *Given a graph with an irredundant k-expression,* Max W-Light *can be solved in time $O(n^{2k(W+2)+4} \log n)$.*

For a graph of clique-width k, one can compute a $(2^{3k} - 1)$-expression of it in polynomial time [34] (see also [35]), while exact computation of the clique-width and a corresponding k-expression is NP-hard [15]. A k-expression of a graph can be transformed into an irredundant one with $O(n)$ nodes in linear time [13]. Now the following is a corollary to Theorem 4.8.

▶ **Corollary 4.9.** *For graphs of clique-width at most k,* Max W-Light *can be solved in time $O(n^{2(2^{3k}-1)(W+2)+4} \log n)$.*

We now prove Theorem 4.8. Let G be an n-vertex graph and T be an irredundant k-expression of G with $O(n)$ nodes. We denote by r the root of T. For each node t in T, let G_t be the graph represented by t with $V_t := V(G_t)$. For each $i \in \{1, \ldots, k\}$, let V_t^i be the set of label-i vertices in G_t.

For a node t in T, a $k \times (W + 2)$ integer matrix $A = (A_{i,j})_{i \in \{1,\ldots,k\},\, j \in \{0,\ldots,W+1\}}$ is an *outdegree signature* of G_t if there is an orientation Λ of G_t such that for each $i \in \{1, \ldots, k\}$ and $j \in \{0, \ldots, W\}$, $A_{i,j}$ is the number of label-i vertices with outdegree j in G_t under Λ, and for each $i \in \{1, \ldots, k\}$, $A_{i,W+1}$ is the number of label-i vertices with outdegree at least $W+1$

in G_t under Λ. The *weight* $w(A)$ of an outdegree signature A is $\sum_{i \in \{1,\dots,k\},\, j \in \{0,\dots,W\}} A_{i,j}$. Note that there are at most $n^{k(W+2)}$ outdegree signatures for each node in T.

▶ **Observation 4.10.** *The optimal value of* Max W-Light *for G is $\max_A w(A)$, where the maximum is taken over all outdegree signatures A of $G_r = G$.*

By Observation 4.10, if we have all possible outdegree signatures for all nodes in T, then we can obtain the optimal value of Max W-Light. We compute the outdegree signatures by a bottom-up dynamic programming over the k-expression T. In a standard way, we can modify the dynamic programming to compute an optimal solution as well.

Computing outdegree signatures for the leaf, $\cup$-, and $\rho_{p,q}$-nodes is fairly straightforward. For $\eta_{p,q}$-nodes, we need the following result.

▶ **Proposition 4.11** (Asahiro, Jansson, Miyano, and Ono [2]). *Given an undirected n-vertex m-edge graph $G = (V, E)$ with lower and upper bounds $(\mathtt{l}(v), \mathtt{u}(v)) \in \{0, \dots, n-1\} \times \{0, \dots, n-1\}$ for each $v \in V$, it can be decided in $O(m^{1.5} \log n)$ time whether there is an orientation Λ such that $\mathtt{l}(v) \le d^+_\Lambda(v) \le \mathtt{u}(v)$ for each $v \in V$.*

▶ **Lemma 4.12.** *For an $\eta_{p,q}$-node, its outdegree signatures can be computed in time $O(n^{2k(W+2)+3} \log n)$ from the outdegree signatures of its child.*

Proof. Let t be an $\eta_{p,q}$-node with the child t'. By the definition of k-expression, $V^i_t = V^i_{t'}$ for all i. Recall that T is irredundant. Hence there is no edge between V^p_t and V^q_t in $G_{t'}$, while G_t has all possible edges between V^p_t and V^q_t.

Let Λ' be an orientation of $G_{t'}$ and A' the corresponding outdegree signature. We say that A' can be *extended* to an outdegree signature A of G_t if there is an orientation Λ of G_t that corresponds to A such that $\Lambda(e) = \Lambda'(e)$ for every $e \in E(G_{t'})$.

▶ **Claim 4.13.** *If A' can be extended to A, then there is an orientation Λ of G_t that corresponds to A such that $d^+_{\Lambda'}(u) < d^+_{\Lambda'}(v)$ implies $d^+_\Lambda(u) \le d^+_\Lambda(v)$ for $u, v \in V^i_t$ and $i \in \{p, q\}$.*

Let A be a candidate of an outdegree signature of G_t. That is, A is a $k \times (W+2)$ integer matrix $A = (A_{i,j})_{i \in \{1,\dots,k\},\, j \in \{0,\dots,W+1\}}$. For $i \in \{p, q\}$, let $(d_{i,1}, \dots, d_{i,|V^i_t|})$ be the nondecreasing sequence such that for each $j \in \{0, \dots, W+1\}$, the value j appears exactly $A_{i,j}$ times. From A', we define $(d'_{i,1}, \dots, d'_{i,|V^i_t|})$ in the same way. For $i \in \{p, q\}$ and $h \in \{1, \dots, |V^i_t|\}$, we define the lower bound $\mathtt{l}_{i,h}$ and the upper bound $\mathtt{u}_{i,h}$ as follows:

$$\mathtt{l}_{i,h} = d_{i,h} - d'_{i,h},$$

$$\mathtt{u}_{i,h} = \begin{cases} d_{i,h} - d'_{i,h} & \text{if } d_{i,h} \le W, \\ n-1 & \text{if } d_{i,h} = W+1. \end{cases}$$

Now let $B = (W_p, W_q; E_B)$ be the complete bipartite graph, where $W_i = \{w_{i,h} : i \in \{p, q\},\ h \in \{1, \dots, |V^t_i|\}\}$ for $i \in \{p, q\}$.

▶ **Claim 4.14.** *A' can be extended to A if and only if there is an orientation Λ_B of B such that for each vertex $w_{i,h}$, it holds that $\mathtt{l}_{i,h} \le d^+_{\Lambda_B}(w_{i,h}) \le \mathtt{u}_{i,h}$.*

For each candidate A, we construct B from A and A'. We also compute the lower and upper bounds of outdegree as described above. Then we check orientability under these bounds. By Proposition 4.11, it can be done in time $O(|E_B|^{1.5} \log |W_p \cup W_q|)$. We can bound this by $O(n^3 \log n)$, and thus the lemma holds. ◀

We have proved that for each node in T, we can compute its outdegree signatures in $O(n^{2k(W+2)+3} \log n)$ time. This completes the proof of Theorem 4.8.

5 Parameterized complexity

In this section, we study the parameterized complexity of the problems. See the recent textbook [14] for standard concepts in the field of parameterized complexity. The parameter is the number of vertices of outdegree at most (at least) W in Max W-Light (resp. Max W-Heavy). We call it the *solution size.*

By using a general theorem in [24], we can easily show the following result.

▶ **Corollary 5.1.** *For any fixed integer $W \geq 0$,* Max W-Light *is W[1]-complete when parameterized by the solution size.*

Let (G, k) be an instance of the parameterized version of Max W-Heavy, where the parameter k is the solution size. We show the following theorem.

▶ **Theorem 5.2.** Max W-Heavy *parameterized by the solution size k admits a kernel with at most $(2W + 4)k + W - 2$ vertices.*

In the following, we assume that $W \geq 3$ since otherwise the problem can be solved in polynomial time [3, 23]. Let $A \subseteq V(G)$ be the set of vertices of degree at least W, and let $B = V(G) \setminus A$. We first bound the number of vertices in A.

▶ **Lemma 5.3.** *If $|A| \geq k \cdot (W + 1)$, then (G, k) is a yes-instance.*

By the lemma above, we can assume that $|A| < k \cdot (W + 1)$. We now modify the graph:

1. remove all vertices of B from G;
2. add an independent set B' of size $\lceil |A| \cdot W/(W - 1)\rceil + W - 2$;
3. for each $v \in A$, repeat the following process:
   a. find $\min\{|N_G(v) \cap B|, W\}$ vertices in B' with degree at most $W - 2$;
   b. add the edges between v and the vertices chosen.

We call the resultant graph G'. Because $W \geq 3$, it holds that $(W + 1)W/(W - 1) \leq W + 3$, and thus $|B'| \leq k(W+3)+W-2$. This implies that $|V(G')| = |A|+|B'| \leq k(2W+4)+W-2$.

To see that the step 3a is always possible, observe that before an execution of the step 3a, at most $W(|A| - 1)$ edges between A and B' are added. On the other hand, if there are at most $W - 1$ vertices of degree at most $W - 2$ in B', then there are at least $(W - 1)(|B'| - (W - 1)) \geq (W - 1)(|A| \cdot W/(W - 1) + W - 2 - (W - 1)) = W(|A| - 1) + 1$ edges between A and B'.

▶ **Lemma 5.4.** *(G, k) is a yes-instance if and only if so is (G', k).*

6 Concluding remarks

We have presented metatheorems to show linear-time and polynomial-time solvability of Max W-Heavy and Max W-Light, respectively. The metatheorems are applied to several important classes of graphs. We believe our metatheorems can be applied to many other graph classes. As the final remark, we present a similar result for the problem of finding a maximum induced subgraph with bounded degeneracy.

▶ **Theorem 6.1.** *For any fixed W, the problem of finding a maximum set of vertices that induces a subgraph of degeneracy at most W can be solved in polynomial time for the classes of chordal graphs, d-trapezoid graphs, circular-arc graphs, and chordal bipartite graphs, and in linear time for graphs of bounded clique-width.*

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

2 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, and Hirotaka Ono. Upper and lower degree bounded graph orientation with minimum penalty. In *CATS*, volume 128 of *CRPIT*, pages 139–146, 2012. URL: http://crpit.com/abstracts/CRPITV128Asahiro.html.

3 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, and Hirotaka Ono. Graph orientations optimizing the number of light or heavy vertices. *Journal of Graph Algorithms and Applications*, 19:441–465, 2015. doi:10.7155/jgaa.00371.

4 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, and Hirotaka Ono. Degree-constrained graph orientation: Maximum satisfaction and minimum violation. *Theory Comput. Syst.*, 58:60–93, 2016. doi:10.1007/s00224-014-9565-5.

5 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. *J. Comb. Optim.*, 22:78–96, 2011. doi:10.1007/s10878-009-9276-z.

6 Hans L. Bodlaender, Pinar Heggernes, and Jan Arne Telle. Recognizability equals definability for graphs of bounded treewidth and bounded chordality. *Electronic Notes in Discrete Mathematics*, 49:559–568, 2015. doi:10.1016/j.endm.2015.06.076.

7 Hans L. Bodlaender, Ton Kloks, Dieter Kratsch, and Haiko Müller. Treewidth and minimum fill-in on d-trapezoid graphs. *J. Graph Algorithms Appl.*, 2:1–23, 1998. doi:10.7155/jgaa.00008.

8 Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276:17–32, 2002. doi:10.1016/S0304-3975(01)00007-X.

9 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci.*, 86:243–266, 1991. doi:10.1016/0304-3975(91)90020-3.

10 Fan R. K. Chung, Michael R. Garey, and Robert E. Tarjan. Strongly connected orientations of mixed multigraphs. *Networks*, 15:477–484, 1985. doi:10.1002/net.3230150409.

11 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34:825–847, 2005. doi:10.1137/S0097539701385351.

12 Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *Theor. Inform. Appl.*, 26:257–286, 1992.

13 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

15 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23:909–939, 2009. doi:10.1137/070687256.

16 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44:54–87, 2015. doi:10.1137/140964801.

17 A. Frank and A. Gyárfás. How to orient the edges of a graph? In *Combinatorics*, volume 18 of *Colloq. Math. Soc. Janos Bolyai*, pages 353–364, 1978.

18 Harold N. Gabow. Upper degree-constrained partial orientations. In *SODA 2006*, pages 554–563, 2006.

19 Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In *WG 2000*, volume 1928 of *LNCS*, pages 196–205, 2000. doi:10.1007/3-540-40064-8_19.

20 Michel Habib and Rolf H. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *Order*, 11:47–60, 1994. doi:10.1007/BF01462229.

21 S. Louis Hakimi. On the degrees of the vertices of a directed graph. *Journal of the Franklin Institute*, 279:290–308, 1965. doi:10.1016/0016-0032(65)90340-6.

22 Pinar Heggernes. Treewidth, partial k-trees, and chordal graphs. Partial curriculum in INF334 – Advanced algorithmical techniques, University of Bergen, Norway, 2005. URL: http://www.ii.uib.no/~pinar/chordal.pdf.

23 Kaveh Khoshkhah. On finding orientations with the fewest number of vertices with small out-degree. *Discrete Applied Mathematics*, 194:163–166, 2015. doi:10.1016/j.dam.2015.05.007.

24 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.*, 289:997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

25 Ton Kloks and Dieter Kratsch. Treewidth of chordal bipartite graphs. *J. Algorithms*, 19:266–281, 1995. doi:10.1006/jagm.1995.1037.

26 Ton Kloks, Dieter Kratsch, and C. K. Wong. Minimum fill-in on circle and circular-arc graphs. *J. Algorithms*, 28:272–289, 1998. doi:10.1006/jagm.1998.0936.

27 Ton Kloks, Ching-Hao Liu, and Sheung-Hung Poon. Feedback vertex set on chordal bipartite graphs. *CoRR*, abs/1104.3915, 2011. URL: http://arxiv.org/abs/1104.3915.

28 Dieter Kratsch. *The Structure of Graphs and the Design of Efficient Algorithms*. Habilitation thesis, Friedrich-Schiller-University of Jena, Germany, 1996.

29 H. G. Landau. On dominance relations and the structure of animal societies: III The condition for a score structure. *Bulletin of Mathematical Biology*, 15:143–148, 1953. doi:10.1007/BF02476378.

30 C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962. URL: http://eudml.org/doc/213681.

31 Luke Mathieson. The parameterized complexity of editing graphs for bounded degeneracy. *Theor. Comput. Sci.*, 411:3181–3187, 2010. doi:10.1016/j.tcs.2010.05.015.

32 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30:417–427, 1983. doi:10.1145/2402.322385.

33 C. St. J. A. Nash-Williams. On orientations, connectivity and odd vertex pairings in finite graphs. *Canad. J. Math*, 12:555–567, 1960. doi:10.4153/CJM-1960-049-6.

34 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5, 2008. Article No. 10. doi:10.1145/1435375.1435385.

35 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96:514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

36 Marcin Pilipczuk and Michal Pilipczuk. Finding a maximum induced degenerate subgraph faster than 2^n. In *IPEC 2012*, volume 7535 of *LNCS*, pages 3–12, 2012. doi:10.1007/978-3-642-33293-7_3.

37 H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *American Mathematical Monthly*, 46:281–283, 1939. doi:10.2307/2303897.

38 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

39 Richard P. Stanley. Acyclic orientations of graphs. *Discrete Mathematics*, 5:171–178, 1973. doi:10.1016/0012-365X(73)90108-8.

40 Venkat Venkateswaran. Minimizing maximum indegree. *Discrete Applied Mathematics*, 143:374–378, 2004. doi:10.1016/j.dam.2003.07.007.

Online Packet Scheduling with Bounded Delay and Lookahead*

Martin Böhm[1], Marek Chrobak[2], Łukasz Jeż[3], Fei Li[4], Jiří Sgall[5], and Pavel Veselý[6]

1 **Computer Science Institute of Charles University, Prague, Czech Republic**
`bohm@iuuk.mff.cuni.cz`

2 **Department of Computer Science and Engineering, University of California, Riverside, USA**
`marek@cs.ucr.edu`

3 **Institute of Computer Science, University of Wrocław, Poland**
`lje@cs.uni.wroc.pl`

4 **Department of Computer Science, George Mason University, USA**
`lifei@cs.gmu.edu`

5 **Computer Science Institute of Charles University, Prague, Czech Republic**
`sgall@iuuk.mff.cuni.cz`

6 **Computer Science Institute of Charles University, Prague, Czech Republic**
`vesely@iuuk.mff.cuni.cz`

Abstract

We study the *online bounded-delay packet scheduling problem (PacketScheduling)*, where packets of unit size arrive at a router over time and need to be transmitted over a network link. Each packet has two attributes: a non-negative weight and a deadline for its transmission. The objective is to maximize the total weight of the transmitted packets. This problem has been well studied in the literature, yet its optimal competitive ratio remains unknown: the best upper bound is 1.828 [6], still quite far from the best lower bound of $\phi \approx 1.618$ [10, 2, 4].

In the variant of PacketScheduling with *s-bounded instances*, each packet can be scheduled in at most s consecutive slots, starting at its release time. The lower bound of ϕ applies even to the special case of 2-bounded instances, and a ϕ-competitive algorithm for 3-bounded instances was given in [3]. Improving that result, and addressing a question posed by Goldwasser [8], we present a ϕ-competitive algorithm for 4-*bounded* instances.

We also study a variant of PacketScheduling where an online algorithm has the additional power of *1-lookahead*, knowing at time t which packets will arrive at time $t+1$. For PacketScheduling with 1-lookahead restricted to 2-bounded instances, we present an online algorithm with competitive ratio $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$ and we prove a nearly tight lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases buffer management, online scheduling, online algorithm, lookahead

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.21

[*] M. Böhm, J. Sgall, and P. Veselý were supported by project 14-10003S of GA ČR and by the GAUK project 548214. M. Chrobak was supported by NSF grants CCF-1217314 and CCF-1536026. Ł. Jeż was supported by NCN grant DEC-2013/09/B/ST6/01538. F. Li was supported by NSF grant CCF-1216993.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 21; pp. 21:1–21:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Optimizing the flow of packets across an IP network gives rise to a plethora of challenging algorithmic problems. In fact, even scheduling packet transmissions from a router across a specific network link can involve non-trivial tradeoffs. Several models for such tradeoffs have been formulated, depending on the architecture of the router, on characteristics of the packets, and on the objective function.

In the model that we study in this paper, each packet has two attributes: a non-negative weight and a deadline for its transmission. The time is assumed to be discrete (slotted), and only one packet can be sent in each slot. The objective is to maximize the total weight of the transmitted packets. We focus on the online setting, where at each time step the router needs to choose a pending packet for transmission, without the knowledge about future packet arrivals. This problem, which we call *online bounded-delay packet scheduling problem* (PacketScheduling), was introduced by Kesselman *et al.* [11] as a theoretical abstraction that captures the constraints and objectives of packet scheduling in networks that need to provide quality of service (QoS) guarantees. The combination of deadlines and weights is used to model packet priorities. In the literature, the PacketScheduling problem is sometimes referred to as *bounded-delay buffer management in QoS switches.* It can also be formulated as the job-scheduling problem $1|p_j = 1, r_j| \sum w_j U_j$, where packets are represented by unit-length jobs with deadlines, with the objective to maximize the weighted throughput.

A router transmitting packets across a link needs to make scheduling decisions on the fly, based only on the currently available information. This motivates the study of online competitive algorithms for PacketScheduling. A simple online greedy algorithm that always schedules the heaviest pending packet is known to be 2-competitive [10, 11]. In a sequence of papers [5, 7, 12, 6], this ratio was gradually improved, and the best currently known ratio is 1.828 [6]. The best lower bound, widely believed to be the optimal ratio, is $\phi = (1 + \sqrt{5})/2 \approx 1.618$ [10, 2, 4]. Closing the gap between these two bounds is one of the most intriguing open problems in online scheduling.

s-Bounded instances. In an attempt to bridge this gap, restricted models have been studied. In the *s-bounded* variant of PacketScheduling, each packet must be scheduled within k consecutive slots, starting at its release time, for some $k \leq s$ possibly depending on the packet. The lower bound of ϕ from [10, 2, 4] holds even in the 2-bounded case. A matching ϕ-competitive algorithm was given Kesselman *et al.* [11] for 2-bounded instances and by Chin *et al.* [3] for 3-bounded instances. Both results are based on the algorithm EDF$_\alpha$, with $\alpha = \phi$, which always schedules the earliest-deadline packet whose weight is at least the weight of the heaviest pending packet divided by α (ties are broken in favor of heavier packets). EDF$_\phi$ is not ϕ-competitive for 4-bounded instances; however, a different choice of α yields a 1.732-competitive algorithm for the 4-bounded case [3].

We present a ϕ-competitive online algorithm for PacketScheduling restricted to 4-bounded instances, matching the lower bound of ϕ (see Section 3). This improves the results from [3] and answers the question posed by Goldwasser in his SIGACT News survey [8].

Algorithms with 1-lookahead. We investigate a variant of PacketScheduling where an online algorithm is able to learn at time t which packets will arrive by time $t + 1$. This property is known as *1-lookahead.* From a practical point of view, 1-lookahead corresponds to the situation in which a router can see the packets that are just arriving to the buffer and that will be available for transmission in the next time slot.

The notion of lookahead is quite natural and it has appeared in the online algorithm literature for paging [1], scheduling [13] and bin packing [9] since the 1990s. Ours is the first paper, to our knowledge, that considers lookahead in the context of packet scheduling.

We provide two results about PacketScheduling with 1-lookahead, restricted to 2-bounded instances. First, in Section 4, we present an online algorithm with competitive ratio of $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$. Then, in Section 5, we give a lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$ on the competitive ratio of algorithms with 1-lookahead which holds already for the 2-bounded case.

2 Definitions and Notation

Formally, we define the PacketScheduling problem as follows. The instance is a set of packets, with each packet p specified by a triple (r_p, d_p, w_p), where r_p and $d_p \geq r_p$ are integers representing the *release time* and *deadline* of p, and $w_p \geq 0$ is a real number representing the *weight* of p. Time is discrete, divided into unit *time slots*, also called *steps*. A *schedule* assigns time slots to some subset of packets such that (i) any packet p in this subset is assigned a slot in the interval $[r_p, d_p]$, and (ii) each slot is assigned to at most one packet. The objective is to compute a schedule that maximizes the total weight of the scheduled packets, also called the *profit*.

In the *s-bounded* variant of PacketScheduling, we assume that each packet p in the instance satisfies $d_p \leq r_p + s - 1$. In other words, this packet must be scheduled within k_p consecutive slots, starting at its release time, for some $k_p \leq s$.

In the online variant of PacketScheduling, which is the focus of our work, at any time t only the packets released at times up to t are revealed. Thus an online algorithm needs to decide which packet to schedule at time t (if any) without any knowledge of packets released after time t.

As is common in the area of online optimization, we measure the performance of an online algorithm $\mathcal{A}$ by its competitive ratio. An algorithm is R-competitive if, for all instances, the total weight of the optimal schedule (computed offline) is at most R times the weight of the schedule computed by $\mathcal{A}$.

We say that a packet is *pending* for an algorithm at time t, if $r_p \leq t \leq d_p$ and p is not scheduled before time t. A (pending) packet p is *expiring* at time t if $d_p = t$, that is, it must be scheduled now or never. A packet p is *tight* if $r_p = d_p$; thus p is expiring already at its release time.

In Sections 4 and 5, we investigate the PacketScheduling problem *with 1-lookahead.* With 1-lookahead, the problem definition changes so that at time t, an online algorithm can also see the packets that will be released at time $t+1$, in addition to the pending packets. Naturally, only a pending packet can be scheduled at time t.

Other terminology and assumptions. We will make several assumptions about our problem that do not affect the generality of our results. First, we can assume that all packets have different weights. Any instance can be transformed into an instance with distinct weights through infinitesimal perturbation of the weights, without affecting the competitive ratio. Second, we assume that at each step there is at least one pending packet. (If not, we can always release a tight packet of weight 0 at each step.)

We define the *earliest-deadline relation* on packets, or *canonical ordering*, denoted $\prec$, where $x \prec y$ means that either $d_x < d_y$ or $d_x = d_y$ and $w_x > w_y$ (so the ties are broken in favor of heavier packets). At any step t, the algorithm maintains the earliest-deadline relation on the set of its pending packets. Throughout the paper, "earliest-deadline packet" means the earliest packet in the canonical ordering.

Regarding the adversary (optimal) schedule, we can assume that it satisfies the following *earliest-deadline property*: if packets p, p' are scheduled in steps t and t', respectively, where $r_{p'} \le t < t' \le d_p$ (that is, p and p' can be swapped in the schedule without violating their release times and deadlines), then $p \prec p'$. This can be rephrased in the following useful way: at any step, the optimum schedule transmits the earliest-deadline packet among all the pending packets that it transmits in the future.

3 An Algorithm for 4-bounded Instances

In this section, we present a ϕ-competitive algorithm for 4-bounded instances. Ratio ϕ is of course optimal [10, 2, 4, see also Section 1]. Up until now, the best competitive ratio for 4-bounded instances was $\sqrt{3} \approx 1.732$, achieved by algorithm $\mathrm{EDF}_{\sqrt{3}}$ in [3]. Our algorithm can be seen as a modification of EDF_{ϕ}, which under certain conditions schedules a packet lighter than w_h/ϕ where h is the heaviest pending packet.

We remark that our algorithm uses memory; in particular, it marks one pending packet under certain conditions. It is an interesting question whether there is a memoryless ϕ-competitive algorithm for 4-bounded instances.

Our algorithm, which we call ToggleH, maintains one mark that may be assigned to one of the pending packets. For a given step t, we choose the following packets from among all pending packets:

- h = the heaviest packet,
- s = the second-heaviest packet,
- f = the earliest-deadline packet with $w_f \ge w_h/\phi$, and
- e = the earliest-deadline packet with $w_e \ge w_h/\phi^2$.

We then proceed as follows:

```
**if** ($h$ is not marked) $\vee$ ($w_s \ge w_h/\phi$) $\vee$ ($d_e > t$)
    schedule $f$
    **if** there is a marked packet **then** unmark it
    **if** ($d_h = t+3$) $\wedge$ ($d_f = t+2$) **then** mark $h$
**else** // ($h$ is marked) $\wedge$ ($w_s < w_h/\phi$) $\wedge$ ($d_e = t$)
    schedule $e$
    unmark $h$
```

Note that when $f \ne h$, then the algorithm will always schedule f. This is because in this case f is a candidate for s, so the condition $w_s \ge w_h/\phi$ holds. The algorithm never specifically chooses s for scheduling – it is only used to determine if there is one more relatively heavy pending packet other than h. (But s *may* get scheduled if it so happens that $s = f$ or $s = e$.) Note also that, if $e \ne f$, then e is scheduled only in a very specific scenario, when all of the following hold: e is expiring, h is marked, and $w_s < w_h/\phi$.

We have two types of packets scheduled by Algorithm ToggleH: *f-packets*, scheduled using the first case, and *e-packets*, scheduled using the second case. Similarly, we refer to the steps as *f-steps* and *e-steps*.

Let us give a high-level view of the analysis using charging schemes and an example that motivates both our algorithm and its analysis. The example consists of four packets j, k, f, h released in step 1, with deadlines $1, 2, 3, 4$ and weights $1-\varepsilon, 1-\varepsilon, 1, \phi$ for a small $\varepsilon > 0$, respectively. The optimum schedules all packets.

Algorithm EDF$_\phi$ performs only f-steps; in our example it schedules f and h in steps 1 and 2, while j and k are lost. Thus the ratio is larger than ϕ. (In fact, after optimizing the threshold and the weight of h, this is the tight example for EDF$_{\sqrt{3}}$ on 4-bounded instances.) ToggleH avoids this example by performing e-step in step 2 and scheduling k which has the role of e and s in the algorithm.

This example and its variants are also important for our analysis. We analyze the algorithms by charging schemes, where the weight of each packet scheduled by the adversary is charged to one or more of the slots of the algorithm's schedule. If the weight charged to each slot is at most R times the weight of the packet scheduled by the algorithm in that slot, the algorithm is R-competitive. In the case of EDF, we charge the weight of each packet j scheduled by the adversary at time t either fully to the step where EDF schedules j, if it is before t, or fully to step t otherwise. In our example, the weight charged to step 1 is $2 - \varepsilon$ while EDF schedules only weight 1, giving the ratio 2. Considering steps 1 and 2 together leads to a better ratio and after balancing the threshold it gives the tight analysis of EDF$_{\sqrt{3}}$.

Our analysis of ToggleH is driven by the variants of the example above where step 2 is an f-step. This may happen in several cases. One case is if in step 2 another packet s with $w_s \geq w_h/\phi$ arrives. If s is not scheduled in step 2, then s is pending in step 3, thus ToggleH schedules a relatively heavy packet in step 3, and we can charge a part of the weight of f, scheduled in step 3 by the adversary, to step 3. This motivates the definition of regular up and back charges below and corresponds to Case 5.1 in the analysis. Another case is when the weight of k is changed to $1/\phi - \varepsilon$. Then ToggleH performs an f-step because k is not a candidate for e, thus the role of e is taken by the non-expiring packet h. However, then the weight of the four packets charged to steps 1 and 2 in the way described above is at most ϕ times the weight of f and h; this corresponds to Case 5.2 of the analysis. Lemma 3.3 gives a subtle argument showing that in the 4-bounded case essentially these two variants of our example are the only difficult situations. Finally, in the original example, ToggleH schedules k in step 2 which is an e-step. Then again h is a pending heavy packet and we can charge some weight of f to step 3. Intuitively it is important that an e-step is performed only in a very specific situation where it is guaranteed that h can be scheduled in the next two steps (as it is marked) and that there is no other packet of comparable weight due to the condition $w_s < w_h/\phi$. Still, there is a case to be handled: If more packets arrive in step 3, it is also possible that the adversary schedules h already in step 2 and we need to redistribute its weight. This case motivates the definition of the special up and back charges below.

▶ **Theorem 3.1.** *Algorithm ToggleH is ϕ-competitive on 4-bounded instances.*

Proof. Fix some optimal adversary schedule. Without loss of generality, we can assume that this schedule satisfies the earliest-deadline property (see Section 2).

Let t be the current step. By h, f, e, and s we denote the packets from the definition of ToggleH. By j we denote the packet scheduled by the adversary. By h' and h'' we denote the heaviest pending packets in steps $t+1$ and $t+2$, respectively. We use the same convention for packets f, e, s, and j.

Our analysis uses a new charging scheme which we now define. The adversary packet j scheduled in step t is charged according to the first case below that applies:

1. If t is an e-step and $j = h$, we charge w_h/ϕ to step t and w_h/ϕ^2 to step $t-1$. We call these charges a *special up charge* and a *special back charge*, respectively. Note that the total charge is equal to $w_h = w_j$.
2. If j is pending for ToggleH in step t, charge w_j to step t. We call this charge a *full up charge.*

3. Otherwise j is scheduled before step t. We charge w_h/ϕ^2 to step t and $w_j - w_h/\phi^2$ to the step where ToggleH scheduled j. We call these charges a *regular up charge* and a *regular back charge*, respectively. We point out that the regular back charge may be negative, but this causes no problems in the proof.

We start with an easy observation that we use several times throughout the proof.

▶ **Lemma 3.2.** *If an f-step t receives a regular back charge, then the up charge it receives is less than w_h/ϕ.*

Proof. For a regular up charge the lemma is trivial (with a slack of a factor of ϕ). For a full up charge, the existence of a back charge implies that the adversary schedules f after j, thus the earliest-deadline property of the adversary schedule implies that $j \prec f$, as both j and f are pending for the adversary at t. Thus ToggleH would schedule j if $w_j \geq w_h/\phi$. Finally, an f-step does not receive a special up charge. ◀

We examine packets scheduled by ToggleH from left to right, that is in order of time. For each time step t, if p is the packet scheduled at time t, we want to show that the charge to step t is at most ϕw_p. However, as it turns out, this will not always be true. In one case we will also consider the next step $t+1$ and the packet p' scheduled in step $t+1$, and show that the total charge to steps t and $t+1$ is at most $\phi(w_p + w_{p'})$.

Let t be the current step. We consider several cases.

Case 1: t is an e-step. By the definition of ToggleH, $w_e \geq w_h/\phi^2$ and $d_e = t$; the latter implies that step t receives no regular back charge. We further note that the heaviest pending packet h' in step $t+1$ is either released at time $t+1$ or it coincides with h, which is still pending and became unmarked by the algorithm in step t; in either case h' is unmarked at the beginning of step $t+1$, which implies that step $t+1$ is an f-step. Thus, step t receives no special back charge, which, combined with the previous observation, implies it receives no back charge of any kind.

Now we claim that the up charge is at most w_h/ϕ. For a special or regular up charge this follows from its definition. For a full up charge, the job j is pending at time t for ToggleH and $j \neq h$ (as for $j = h$ the special charges are used). This implies that $w_j < w_h/\phi$, as otherwise $w_s \geq w_h/\phi$ and t would be an f-step. Thus the full charge is $w_j \leq w_h/\phi$ as well.

Using $w_e \geq w_h/\phi^2$, the charge is at most $w_h/\phi \leq \phi w_e$ and we are done.

Case 2: t is an f-step and t does not receive a back charge. Then t can only receive an up-charge, and this up charge is at most $w_h \leq \phi w_f$, where the inequality follows from the definition of f.

Case 3: t is an f-step and t receives a special back charge. From the definition of special charges, the next step is an e-step, and therefore h' is marked at its beginning. Since the only packet that may be marked after an f-step is h, we thus have $h = h' = j'$, and the special back charge is w_h/ϕ^2. Since $f \prec h$, the adversary cannot schedule f after step t, so step t cannot receive a regular back charge.

We claim that the up charge to step t is at most w_f. Indeed, a regular up charge is at most $w_h/\phi^2 \leq w_f$, and a special up charge does not happen in an f-step. To show this bound for a full up charge, assume for contradiction that $w_j > w_f$. This implies that $j \neq f$ and, since ToggleH scheduled f, we have $d_j > d_f$. In particular j is pending at time $t+1$.

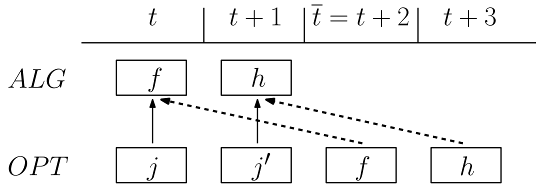


Figure 1 An illustration of the situation in Case 5.2. Up charges are denoted by solid arrows and back charges by dashed arrows.

Thus $w_{s'} \geq w_j > w_f \geq w_h/\phi$, contradicting the fact that $t+1$ is an e-step. Therefore the full charge is $w_j \leq w_f$, as claimed.

As $w_h \leq \phi w_f$, the total charge to t is at most $w_f + w_h/\phi^2 \leq w_f + w_f/\phi = \phi w_f$.

Case 4: t is an f-step, t receives a regular back charge and no special back charge, and $f = h$. The up charge is at most w_h/ϕ by Lemma 3.2 and the back charge is at most w_h, thus the total charge is at most $w_h + w_h/\phi = \phi w_h$, and we are done.

Case 5: t is an f-step, t receives a regular back charge and no special back charge, and $f \neq h$. Let $\bar{t}$ be the step when the adversary schedules f. We distinguish two sub-cases.

Case 5.1: In step $\bar{t}$, a packet of weight at least w_h/ϕ is pending for the algorithm. Then the regular back charge to t is at most $w_f - (w_h/\phi)/\phi^2 = w_f - w_h/\phi^3$. As the up charge to t is at most w_h/ϕ by Lemma 3.2, the total charge to t is at most $w_h/\phi + w_f - w_h/\phi^3 = w_f + w_h/\phi^2 \leq (1+1/\phi)w_f = \phi w_f$, and we are done.

Case 5.2: In step $\bar{t}$, no packet of weight at least w_h/ϕ is pending for the algorithm. In this case we consider the charges to steps t and $t+1$ together. First, we claim the following.

▶ **Lemma 3.3.** *ToggleH schedules h in step $t+1$. Furthermore, step $t+1$ receives no special charge and it receives an up charge of at most w_h/ϕ^2.*

Proof. Since $f \neq h$, we have $f \prec h$ and thus, using also the definition of $\bar{t}$ and 4-boundedness, $\bar{t} \leq d_f < d_h \leq t+3$. The case condition implies that h is not pending at $\bar{t}$, thus ToggleH schedules h before $\bar{t}$. The only possibility is that ToggleH schedules h in step $t+1$, $\bar{t} = d_f = t+2$, and $d_h = t+3$; see Figure 1 for an illustration. This also implies that ToggleH marks h in step t.

We claim that $w_{s'} < w_h/\phi$. Indeed, otherwise either s' is pending in step $t+2$, contradicting the condition of Case 5.2, or $d_{s'} = t+1 < d_h$, thus s' is a better candidate for f' than h, which contradicts the fact that the algorithm scheduled $f' = h$.

The claim also implies that $h' = h$, as otherwise $w_{s'} \geq w_h$. Since $h = h'$ is scheduled in step $t+1$, there is no marked packet in step $t+2$ and $t+2$ is an f-step; thus there is no special back charge to $t+1$.

We note that step $t+1$ is also an f-step, since ToggleH schedules h in step $t+1$ and $d_h > t+1$. Since $h' = h$ is marked when step $t+1$ starts and $w_{s'} < w_h/\phi$, the reason that step $t+1$ is an f-step must be that $d_{e'} > t+1$.

There is no special up charge to step $t+1$ as it is an f-step. If the up charge to step $t+1$ is a regular up charge, by definition it is at most $w_{h'}/\phi^2 = w_h/\phi^2$ and the lemma holds.

The only remaining case is that of a full up charge to step $t+1$ from a packet j' scheduled by the adversary in step $t+1$ and pending for ToggleH in step $t+1$. Since $j' \neq h$, it is a candidate for s', and thus $w_{j'} < w_h/\phi \le w_f$. The earliest-deadline property of the adversary schedule implies that $j' \prec f$; together with $d_f = t+2$ and $w_{j'} < w_f$ this implies $d_{j'} = t+1$. Therefore $w_{j'} < w_h/\phi^2$, as otherwise j' is a candidate for e', but we have shown that $d_{e'} > t+1$. Thus the regular up charge is at most $w_{j'} < w_h/\phi^2$ and the lemma holds also in the remaining case. ◀

By Lemma 3.3, step $t+1$ receives no special charge and an up charge of at most w_h/ϕ^2 and ToggleH schedules h in step $t+1$. Step $t+1$ thus also receives a regular back charge of at most w_h. So the total charge to step $t+1$ is at most $w_h/\phi^2 + w_h \le w_f/\phi + w_h$. Moreover, using Lemma 3.2, the total charge to step t is at most $w_h/\phi + w_f$. Thus, the total charge to these two steps is at most $(w_h/\phi + w_f) + (w_f/\phi + w_h) = \phi(w_f + w_h)$, as f and h are the two packets scheduled by ToggleH.

In each case we have shown that a step or a pair of consecutive steps receive a total charge of at most ϕ times the weight of packets scheduled in these steps. Thus ToggleH is ϕ-competitive for the 4-bounded case. ◀

4 An Algorithm for 2-Bounded Instances with Lookahead

In this section, we present an algorithm for 2-*bounded* PacketScheduling *with 1-lookahead*, as defined in Section 2.

Consider some online algorithm $\mathcal{A}$. Recall that, for a time step t, packets *pending* for $\mathcal{A}$ are those that are released at or before time t and have neither expired nor been scheduled by $\mathcal{A}$ before time t. *Lookahead* packets at time t are the packets with release time $t+1$. For $\mathcal{A}$, we define the *plan* in step t to be the optimal schedule in the time interval $[t, \infty)$ that consists of pending and lookahead packets at time t and has the earliest-deadline property. For 2-bounded instances, this plan will only use slots t, $t+1$ and $t+2$. We will typically denote the packets in the plan scheduled in these slots by p_1, p_2, p_3, respectively. The earliest-deadline property then implies that if both p_1 and p_2 have release time t and deadline $t+1$ then p_1 is heavier than p_2 and similarly for p_2 and p_3.

Fix some parameter $\alpha > 1$. At any time step t, our algorithm CompareWithBias(α) proceeds as follows:

```
let $p_1, p_2, p_3$ be the plan at time $t$
if $r_{p_2} = t$ and $w_{p_1} < \min(\, w_{p_2}\,,\, w_{p_3}\,,\, \frac{1}{2\alpha}(w_{p_2} + w_{p_3})\,)$
    then schedule $p_2$
else schedule $p_1$
```

Note that if the algorithm schedules p_2 then p_1 must be expiring, for otherwise $w_{p_1} > w_{p_2}$ (by canonical ordering). Also, the scheduled packet is at least as heavy as the heaviest expiring packet q, since clearly $w_{p_1} \ge w_q$ and the algorithm schedules p_2 only if $w_{p_1} < w_{p_2}$.

▶ **Theorem 4.1.** *The algorithm* CompareWithBias(α) *is R-competitive for packet scheduling on 2-bounded instances for* $R = \frac{1}{2}(\sqrt{13} - 1) \approx 1.303$ *if* $\alpha = \frac{1}{4}(\sqrt{13} + 3) \approx 1.651$.

Let ALG be the schedule produced by CompareWithBias. Let us consider an optimal schedule OPT (a.k.a. schedule of the adversary) satisfying the canonical ordering, i.e., if a packet x is scheduled before a packet y in OPT then either y is released after x is scheduled or $x \prec y$. Recall that we are assuming w.l.o.g. that the weights of packets are different.

The analysis of CompareWithBias is based on a charging scheme. First we define a few packets by their schedule times:

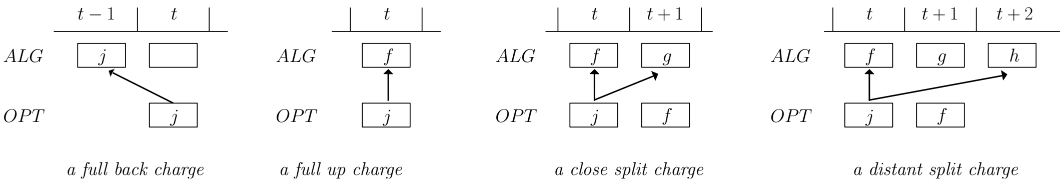


Figure 2 Non-chaining charges. Note that for split charges f is scheduled in step $t+1$ in OPT which follows from the fact that we do not charge j using a full up charge.

- j = packet scheduled in step t in OPT,
- f = packet scheduled in step t in ALG,
- g = packet scheduled in step $t+1$ in ALG.

Informal description of charging. We use three types of charges. The adversary's packet j in step t is charged using a *full charge* either to step $t-1$ if ALG schedules j in step $t-1$ or to step t if $w_f \geq w_j$ (including the case $f = j$) and f is not in step $t+1$ in OPT; the last condition assures that step t does not receive two full charges.

The second type are *split charges* that occur in step t if $w_f > w_j$, j is pending in step t in ALG and f is in step $t+1$ in OPT, i.e., step t receives a full back charge from f. In this case, we distribute the charge from j to f and another relatively large packet f' scheduled in step $t+1$ or $t+2$ in ALG; we shall prove that one of these steps satisfies $2\alpha w_j < w_f + w'_f$. We charge to step $t+2$ only when it is necessary, which allows us to prove that split-charge pairs are pairwise disjoint. Also, in this case we analyze the charges to both steps together, thus it is not necessary to fix a distribution of the weight to the two steps.

The remaining case is when $w_f < w_j$ and j is not scheduled in $t-1$ in ALG. We analyze these steps in maximal consecutive intervals, called *chains* and the corresponding charges are *chain charges*. Inside each chain we distribute the charge of each packet j scheduled at t in OPT to steps $t-1$, t and $t+1$, if these steps are also in the chain. The distribution of weights shall depend on a parameter δ. Packets at the beginning and at the end of the chain are charged in a way that minimizes the charge to steps outside of the chain. In particular, the step before a chain receives no charge from the chain.

Notations and the charging scheme. A step t for which $w_f < w_j$ and j is pending in step t in ALG is called *a chaining step*. A maximal sequence of successive chaining steps is called a *chain*. The chains with a single step are called *singleton chains*, the chains with at least two steps are called *long chains*.

The pair of steps that receive a split charge from the same packet is called a *split-charge pair*. The charging scheme does not specify the distribution of the weight to the two steps of the split-charge pair, as the charges to them are analyzed together.

Let $\delta = \frac{1}{6}(5 - \sqrt{13}) \approx 0.232$. Packet j scheduled in OPT at time t is charged according to the first rule below that applies. See Figures 2 and 3 for an illustration of different types of charges.

1. If j is scheduled in step $t-1$ in ALG, charge w_j to step $t-1$. We call this charge a *full back charge*.
2. If $w_f \geq w_j$ and f is not scheduled in step $t+1$ in OPT (in particular, if $j = f$), charge w_j to step t. We call this charge a *full up charge*.

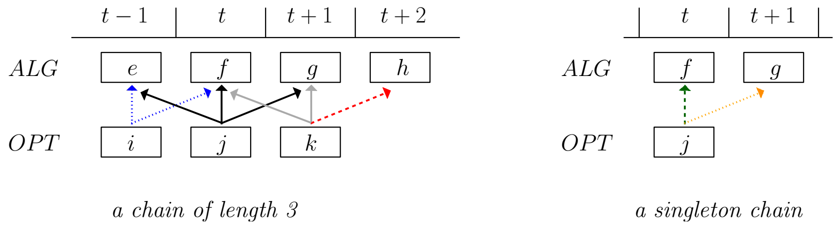


Figure 3 On the left, a chain of length 3 starting in step $t-1$ and ending in step $t+1$. The *chain beginning charges* are denoted by dotted (blue) lines, the chain end charges are denoted by gray lines and the *forward charge from a chain* is depicted by a dashed (red) arrow. Black arrows denote the *chain link charges.* On the right, an example of a singleton chain, with the *up charge from a singleton chain* denoted with a dashed (green) line and the *forward charge from a singleton chain* denoted with a dotted (orange) line.

3. If $w_f > w_j$ and at least one of the following holds:
   - $2\alpha w_j < w_f + w_g$,
   - g does not get a full back charge and $2\alpha(w_{p_1} - w_g) < w_f + w_g$ where p_1 is the first packet in the plan at time t,

   then charge w_j to the pair of steps t and $t+1$. We call this charge a *close split charge.*
4. If $w_f > w_j$, then charge w_j to the pair of steps t and $t+2$. We call this charge a *distant split charge.*
5. Otherwise step t is a chaining step, as $w_f < w_j$ and ALG does not schedule f in step $t-1$ by the previous cases. We distinguish the following subcases.
   a. If step t is (the only step of) a singleton chain, then charge $\min(w_j, Rw_f)$ to step t and $w_j - Rw_f$ to step $t+1$ if $w_j > Rw_f$. We call these charges an *up charge from a singleton chain* and a *forward charge from a singleton chain.*
   b. If step t is the first step of a long chain, charge $2\delta w_j$ to step t, and $(1-2\delta)w_j$ to step $t+1$. We call these charges *chain beginning charges.*
   c. If step t is the last step of a long chain, charge δw_j to step $t-1$, $(R-1+2\delta)w_f$ to step t, and $(1-\delta)w_j - (R-1+2\delta)w_f$ to step $t+1$. We call these charges *chain end charges*; the charge to step $t+1$ is called a *forward charge from a chain.* (Note that we always have $(1-\delta)w_j > (R-1+2\delta)w_f$, since $w_j > w_f$ and $1-\delta = R-1+2\delta$.)
   d. Otherwise, i.e., step t is inside a long chain, charge δw_j to step $t-1$, δw_j to step t, and $(1-2\delta)w_j$ to step $t+1$. We call these charges *chain link charges.*

The analysis of our charging scheme is omitted due to space limitation.

5 A Lower Bound for 2-bounded Instances with Lookahead

In this section, we prove that there is no online algorithm for PacketScheduling with 1-lookahead that has competitive ratio smaller than $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$, even for 2-bounded instances. The idea of our proof is somewhat similar to the proof of the lower bound of ϕ for PacketScheduling [10, 2, 4].

▶ **Theorem 5.1.** *Let $R = \frac{1}{4}(1+\sqrt{17})$. For each $\varepsilon > 0$, no deterministic online algorithm for PacketScheduling with 1-lookahead can be $(R-\varepsilon)$-competitive, even for 2-bounded instances.*

Proof. Fix some online algorithm $\mathcal{A}$ and some $\varepsilon > 0$. We will show that, for some sufficiently large integer n and sufficiently small $\delta > 0$, there is a 2-bounded instance of PacketScheduling with 1-lookahead, parametrized by n and δ, for which the optimal profit is at least $(R - \varepsilon)$ times the profit of $\mathcal{A}$.

Our instance will consist of phases $0, \ldots, k$, for some $k \leq n$. In each phase $i < n$ we will release three packets whose weights will grow roughly exponentially from one phase to next. The number k of phases is determined by the adversary based on the behavior of $\mathcal{A}$.

The adversary strategy is as follows. We start with phase 0. Suppose that some phase i, where $0 \leq i < n$, has been reached. In phase i the adversary releases the following three packets:

- A packet a_i with weight w_i, release time $2i + 1$ and deadline $2i + 1$, i.e., a tight packet.
- A packet b_i with weight w_{i+1}, release time $2i + 1$ and deadline $2i + 2$.
- A packet c_i with weight w_{i+1}, release time $2i + 2$ and deadline $2i + 3$.

(The weights w_i will be specified later.) Now, if $\mathcal{A}$ schedules an expiring packet in step $2i + 1$ (a tight packet a_i or c_{i-1}, which may be pending from the previous phase), then the game continues; the adversary will proceed to phase $i + 1$. Otherwise, the algorithm schedules packet b_i, in which case the adversary lets $k = i$ and the game ends. Note that in step $2i + 2$ the algorithm may schedule only b_i or c_i, each having weight w_{i+1}. Also, importantly, in step $2i + 1$ the algorithm cannot yet see whether the packets from phase $i + 1$ will arrive or not.

If phase $i = n$ is reached, then in phase n the adversary releases a single packet a_n with weight w_n and release time and deadline $2n + 1$, i.e., a tight packet.

We calculate the ratio between the weight of packets in an optimal schedule and the weight of packets sent by the algorithm. Let $S_k = \sum_{i=0}^{k} w_i$. There are two cases: either $k < n$, or $k = n$.

Case 1: $k < n$. In all steps $2i + 1$ for $i < k$ algorithm $\mathcal{A}$ scheduled an expiring packet of weight w_i and in step $2k + 1$ it scheduled packet b_k of weight w_{k+1}. In an even step $2i + 2$ for $i \leq k$ it scheduled a packet of weight w_{i+1}. Note that there is no packet scheduled in step $2k+3$. Overall, $\mathcal{A}$ scheduled packets of total weight $S_{k-1} + w_{k+1} + S_{k+1} - w_0 = 2S_{k+1} - w_k - w_0$.

The adversary schedules packets of weight w_{i+1} in steps $2i + 1$ and $2i + 2$ for $i < k$ and all packets from phase k in steps $2k + 1$, $2k + 2$ and $2k + 3$. In total, the optimum has a schedule of weight $2S_{k+1} - 2w_0 + w_k$. The ratio is

$$R_k = \frac{2S_{k+1} + w_k - 2w_0}{2S_{k+1} - w_k - w_0}.$$

Case 2: $k = n$. As before, in all odd steps $2i + 1$ for $i < n$ algorithm $\mathcal{A}$ scheduled an expiring packet of weight w_i and in all even steps $2i + 2$ for $i < n$ it scheduled a packet of weight w_{i+1}. In the last step $2n + 1$ it scheduled a packet of weight w_n as there is no other choice. Overall, the total weight of $\mathcal{A}$'s schedule is $2S_n - w_0$.

The adversary schedules packets of weight w_{i+1} in steps $2i + 1$ and $2i + 2$ for $i < n$ and a packet of weight w_n in the last step $2n + 1$ which adds up to $2S_n - 2w_0 + w_n$. The ratio is

$$\widehat{R}_n = \frac{2S_n + w_n - 2w_0}{2S_n - w_0}.$$

We start with an intuitive explanation which leads to the optimal setting of weights w_i and the ratio R for the instances of the type described above. We normalize the instances so that $w_0 = 1$. We want to set the weights so that $R_k \geq R - \varepsilon$ for all $k \geq 0$ and $\widehat{R}_n \geq R - \varepsilon$. We first find the weights depending on δ such that $R_k = R$ for all $k \geq 1$. Using $w_k = S_k - S_{k-1}$ for $k \geq 1$ and $w_0 = 1$, the condition $R_k = R$ for $k \geq 1$ is rewritten as

$$R = \frac{2S_{k+1} + S_k - S_{k-1} - 2}{2S_{k+1} - S_k + S_{k-1} - 1}, \tag{1}$$

or equivalently as

$$(2R-2)S_{k+1} - (R+1)S_k + (R+1)S_{k-1} = -(2-R)\,. \tag{2}$$

A general solution of this linear recurrence with $S_0 = w_0 = 1$ and a parameter δ is

$$S_k = (\gamma+1)\alpha^k + \delta(\beta^k - \alpha^k) - \gamma\,, \tag{3}$$

where $\alpha < \beta$ are the two roots of the characteristic polynomial of the recurrence $(2R-2)x^2 - (R+1)x + (R+1)$ and $\gamma = (2-R)/(2R-2)$. To justify (3), a general solution is $A\alpha^k + B\beta^k - \gamma$ for parameters A and B and a suitable constant γ. Considering $A = B = 0$, the value $\gamma = (2-R)/(2R-2)$ follows. Considering the constraint $S_0 = 1$, we obtain $A + B = \gamma + 1$; our parametrization by δ in (3) is equivalent but more convenient for further analysis.

In our case of $R = \frac{1}{4}(1+\sqrt{17})$ a calculation gives

$$\alpha = R + \tfrac{1}{2} = \tfrac{1}{4}(3+\sqrt{17})\,, \quad \beta = R+1 = \tfrac{1}{4}(5+\sqrt{17}) \text{ and } \quad \gamma = R = \tfrac{1}{4}(1+\sqrt{17})\,. \tag{4}$$

A calculation shows that for $\delta = 0$, the solution satisfies $R_0 = R$. We choose a solution with a sufficiently small $\delta > 0$ which guarantees $R_0 \geq R - \varepsilon$. Since $1 < \alpha < \beta$, for large n, the dominating term in S_n is $\delta\beta^n$. Thus

$$\lim_{n\to\infty} \widehat{R}_n = \lim_{n\to\infty} \frac{2S_n + S_n - S_{n-1}}{2S_n} = \lim_{n\to\infty} \frac{3\delta\beta^n - \delta\beta^{n-1}}{2\delta\beta^n} = \frac{3\beta-1}{2\beta} = R\,. \tag{5}$$

The last equality is verified by a direct calculation; actually it is the equation that defines the optimal R for our construction (if β as the root of the characteristic polynomial of the recurrence is expressed in terms of R).

For a formal proof, we set $w_0 = 1$ and for $i = 1, 2, \ldots$,

$$w_i = (\gamma+1)\alpha^{k-1}(\alpha-1) + \delta(\beta^{k-1}(\beta-1) - \alpha^{k-1}(\alpha-1))\,,$$

where the parameters α, β and γ are given by (4) and $\delta > 0$ is sufficiently small. By a routine calculation we verify (3) and (2). Thus $R_k = R$ for $k \geq 1$. For R_0, we first verify that $\delta = 0$ would yield $w_1 = \alpha$ and $R_0 = R$. By continuity of the dependence of w_1 and R_0 on δ, for a sufficiently small $\delta > 0$, we have $R_0 \geq R - \varepsilon$; fix such a $\delta > 0$. Now, for $n \to \infty$, $S_n = \delta\beta^n + O(\alpha^n) = \delta\beta^n(1 + o(1))$. Thus, the calculation (5) gives $\lim_{n\to\infty} \widehat{R}_n = R$. Consequently, $\widehat{R}_n \geq R - \varepsilon$ for a sufficiently large n of our choice. This defines the required instance and completes the proof. ◀

References

1 Susanne Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997. doi:10.1007/PL00009158.

2 Nir Andelman, Yishay Mansour, and An Zhu. Competitive queueing policies for QoS switches. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 761–770, 2003.

3 Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. of Discrete Algorithms*, 4(2):255–276, 2006.

4 Francis Y. L. Chin and Stanley P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

5 Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Improved online algorithms for buffer management in QoS switches. In *Proc. 12th Annual European Symposium (ESA'04)*, pages 204–215, 2004.

6 Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 209–218, 2007.

7 Matthias Englert and Matthias Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009.

8 Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

9 Edward F. Grove. Online bin packing with lookahead. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 430–436, 1995.

10 Bruce Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proc. Conference on Information Sciences and Systems*, pages 434–438, 2001.

11 Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

12 Fei Li, Jay Sethuraman, and Clifford Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 801–802, 2005.

13 Rajeev Motwani, Vijay Saraswat, and Eric Torng. Online scheduling with lookahead: Multipass assembly lines. *INFORMS J. on Computing*, 10(3):331–340, 1998.

Biconnectivity, Chain Decomposition and *st*-Numbering Using $O(n)$ Bits

Sankardeep Chakraborty[1], **Venkatesh Raman**[2], and **Srinivasa Rao Satti**[3]

1 **The Institute of Mathematical Sciences, Homi Bhabha National Institute (HBNI), Chennai, India**
sankardeep@imsc.res.in

2 **The Institute of Mathematical Sciences, Homi Bhabha National Institute (HBNI), Chennai, India**
vraman@imsc.res.in

3 **Seoul National University, Seoul, South Korea**
ssrao@cse.snu.ac.kr

Abstract

Recent work by Elmasry et al. (STACS 2015) and Asano et al. (ISAAC 2014) reconsidered classical fundamental graph algorithms focusing on improving the space complexity. Elmasry et al. gave, among others, an implementation of depth first search (DFS) of a graph on n vertices and m edges, taking $O(m \lg\lg n)$ time[1] using $O(n)$ bits of space improving on the time bound of $O(m \lg n)$ due to Asano et al. Subsequently Banerjee et al. (COCOON 2016) gave an $O(m+n)$ time implementation using $O(m+n)$ bits, for DFS and its classical applications (including testing for biconnectivity, and finding cut vertices and cut edges). Recently, Kammer et al. (MFCS 2016) gave an algorithm for testing biconnectivity using $O(n + \min\{m, n \lg\lg n\})$ bits in linear time.

In this paper, we consider $O(n)$ bits implementations of the classical applications of DFS. These include the problem of finding cut vertices, and biconnected components, chain decomposition and *st*-numbering. Classical algorithms for them typically use DFS and some $\Omega(\lg n)$ bits of information at each node. Our $O(n)$-bit implementations for these problems take $O(m \lg^c n \lg\lg n)$ time for some small constant c ($c \leq 3$). Central to our implementation is a succinct representation of the DFS tree and a space efficient partitioning of the DFS tree into connected subtrees, which maybe of independent interest for space efficient graph algorithms.

1998 ACM Subject Classification F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases biconnectivity, st-number, chain decomposition, tree cover, space efficient algorithms, read-only memory

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.22

1 Introduction

Motivated by the rapid growth of huge data sets ("big data"), space efficient algorithms are becoming increasingly important than ever before. The proliferation of handheld or embedded devices that are equipped with only a small amount of general-purpose memory provides another motivation for studying space efficient algorithms. In some of these devices, writing in the memory is a costly operation in terms of both speed and time than reading.

[1] We use lg to denote logarithm to the base 2.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 22; pp. 22:1–22:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In such scenarios, algorithms that do not modify the input and use only a limited amount of work space are very much desired.

The standard model to study space efficient algorithms is the read-only memory model, and there is a rich history in computational complexity theory of such algorithms which use as little space as possible. In particular, L (also known as LSPACE or DLOGSPACE) is the complexity class containing decision problems that can be solved by a deterministic Turing machine using only logarithmic amount of work space for computation. There are several important algorithmic results [15, 12] for this class, the most celebrated being Reingold's method [30] for checking reachability between two vertices in an undirected graph. Barnes et al [7] gave a slightly sublinear space (using $n/2^{\Theta(\sqrt{\lg n}}$ bits) algorithm for directed s-t connectivity with polynomial running time. Space-efficient algorithms for classical selection and sorting problems [26, 27], and problems in computational geometry have also been studied [5, 6]. Recent work has focused on space requirement in special classes of graphs like planar and H-minor free graphs [9, 2].

For most of these algorithms using small space i.e., sublinear bits, their running time is often some polynomial of very high degree. Tompa [36] showed that for directed s-t connectivity, if the number of bits available is $o(n)$ then some natural algorithmic approaches to the problem require superpolynomial time. Thus it is sensible to focus (as in the case of some the recent papers like that of [16, 1, 4, 3, 24]) on designing algorithms that use $O(n)$ bits of workspace. Our main objective here is to reduce the working space of the classical algorithms to $O(n)$ bits with little or no penalty in running time. In these recent series of papers [16, 1, 4, 3, 24] space-efficient algorithms for only a few basic algorithmic graph problems are discussed: DFS, BFS, topological sort, strongly connected components, sparse spanning biconnected subgraph, among others. We add to this growing body of space-efficient algorithm design literature by providing such algorithms for a few more classical algorithmic graph problems, namely biconnectivity, st-numbering and chain decomposition.

1.1 Our results and organization of the paper

Our starting point is an $O(m + n)$ time and $O(n \lg(m/n))$ bits implementation for DFS and for finding a 'chain decomposition' using which we can find cut vertices, bridges, maximal biconnected components and ear decomposition (see Section 2 for definitions). This improves an earlier $O(m + n)$ time and $O(m + n)$ bits implementation [3] (see Theorem 4). The space used by these algorithms, for some ranges of m (say $\Theta(n(\lg \lg n)^c$ for some constant c), is even better than that of the recent work by Kammer et al [24], that computes cut vertices using $O(n + \min\{m, n \lg \lg n\})$ bits. This implementation appears in Section 3.

Chain decomposition is an important preprocessing routine for an algorithm to find cut vertices and biconnected components and also to test 3-connectivity [31] among others. In Section 5, we give an algorithm that takes $O(m \lg^3 n \lg \lg n)$ time using $O(n)$ bits, improving on previous implementations that took $\Omega(n \lg n)$ bits or $\Theta(m + n)$ bits.

In Section 6, we give improved algorithms for finding cut vertices and biconnected components by giving a space efficient implementation of Tarjan's classical lowpoint algorithm. This takes $O(m \lg n \lg \lg n)$ time.

Given a biconnected graph, and two distinguished vertices s and t, st-numbering is a numbering of the vertices of the graph so that s gets the smallest number, t gets the largest and every other vertex is adjacent both to a lower-numbered and to a higher-numbered vertex. Finding an st-numbering is an important preprocessing routine for a planarity testing algorithm. In Section 7, we give an algorithm to determine an st-numbering of a biconnected graph that takes $O(m \lg^2 n \lg\lg n)$ time using $O(n)$ bits. This improves the earlier implementations that take $\Omega(n \lg n)$ bits.

Techniques. There are several approaches to find cut vertices and biconnected components. An algorithm due to Tarjan [34] is the standard 'textbook' algorithm, and another due to Schmidt [32] is based on chain decomposition of graphs. Both these approaches compute DFS and process the DFS tree in specific order maintaining some auxiliary information of the nodes. To implement these in $O(n)$ bits, our main idea is to process the nodes of the DFS tree in batches of $O(n/\lg n)$ nodes. Towards that, we use tree-cover algorithms (that are used in succinct representations of trees) that partition a tree into connected subtrees. This is described in detail in Section 4.

Model of Computation. Like all the recent research that focused on space-efficient graph algorithms [16, 1, 4, 3, 24], here also we assume that the input graph is given in a read-only memory (and so cannot be modified). If an algorithm must do some outputting, this is done on a separate write-only memory. When something is written to this memory, the information cannot be read or rewritten again. So the input is "read only" and the output is "write only". In addition to the input and the output media, a limited random-access workspace is available. The data on this workspace is manipulated at word level as in the standard word RAM model, where the machine consists of words of size $w = \Omega(\lg n)$ bits; and any logical, arithmetic, and bitwise operations involving a constant number of words take a constant amount of time. We count space in terms of the number of bits in the workspace used by the algorithms. Historically, this model is called the *register input model* and it was introduced by Frederickson [20] while studying some problems related to sorting and selection. We assume that the input graph $G = (V, E)$ is represented using *adjacency array*, i.e., given a vertex v and an integer k, we can access the kth neighbor of vertex v in constant time. This representation was used in [16, 3, 24] recently to design various space efficient graph algorithms. We use n and m to denote the number of vertices and the number of edges respectively, in the input graph G. Throughout the paper, we assume that the input graph is a connected graph, and hence $m \geq n - 1$.

2 Preliminaries

Rank-Select. Given a bitvector B of length n, the rank and select operations are defined as follows:

- $rank_a(i, B)$ = number of occurrences of $a \in \{0, 1\}$ in $B[1, i]$, for $1 \leq i \leq n$;
- $select_a(i, B)$ = position in B of the ith occurrence of $a \in \{0, 1\}$.

The following theorem gives an efficient structure to support these operations.

▶ **Theorem 1** ([11]). *Given a bitstring B of length n, one can construct a $o(n)$-bit auxiliary structure to support rank and select operations in $O(1)$ time. Also, such a structure can be constructed from the given bitstring in $O(n)$ time.*

Space-efficient DFS. Elmasry et al. [16] showed the following tradeoff result for DFS,

▶ **Theorem 2** ([16]). *For every function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n)$ can be computed within the resource bound of this theorem (e.g., in $O(n)$ time using $O(n)$ bits), the vertices of a graph G can be visited in depth first order in $O((m + n)t(n))$ time with $O(n + n\frac{\lg\lg n}{t(n)})$ bits.*

In particular, fixing $t(n) = O(\lg\lg n)$, we obtain a DFS implementation which runs in $O(m \lg\lg n)$ time using $O(n)$ bits. We build on top of this DFS algorithm to design all of our space-efficient algorithms.

Graph theoretic terminology. A cut vertex in an undirected graph is a vertex that when removed (with its incident edges) from a graph creates more components than previously in the graph. Similarly, a bridge is an edge that when removed (the vertices stay in place) from a graph creates more components than previously in the graph. A graph is biconnected if it is connected and contains at least 3 vertices, but no cut vertex. A graph is 2-edge-connected if it is connected and contains at least 2 vertices, but no bridge. Let $G = (V, E)$ be a biconnected graph and $s \neq t \in V$. An ordering $s = v_1, v_2, \cdots, v_n = t$ of the vertices of G is called an st-ordering, if for all vertices $v_j, 1 < j < n$, there exist $1 \leq i < j < k \leq n$ such that $\{v_i, v_j\}, \{v_j, v_k\} \in E$. It is well-known that G is biconnected if and only if, for every edge $\{s, t\} \in E$, it has an st-ordering.

Chain decomposition and its application. Schmidt [31] introduced a decomposition of the input graph that partitions the edge set of the graph into cycles and paths, called chains, and used this to design an algorithm to find cut vertices and biconnected components [32] and also to test 3-connectivity [31] among others. In this section we discuss the details of the decomposition algorithm and some of the applications for which we give space efficient implementations in the paper later.

The algorithm first performs a depth first search on G. Let r be the root of the DFS tree T. DFS assigns an index to every vertex v, namely, the time vertex v is discovered for the first time during DFS – call it the depth-first-index of v ($DFI(v)$). Imagine that the back edges are directed away from r and the tree edges are directed towards r. The algorithm decomposes the graph into a set of paths and cycles called chains as follows. First we mark all the vertices as unvisited. Then we visit every vertex starting at r in the increasing order of DFI, and do the following. For every back edge e that originates at v, we traverse a directed cycle or a path. This begins with v and the back edge e and proceeds along the tree towards the root and stops at the first visited vertex or the root. During this step, we mark every encountered vertex as visited. This forms the first chain. Then we proceed with the next back edge at v, if any, or move towards the next vertex in the increasing DFI order and continue the process. Let D be the collection of all such cycles and paths. Notice that the cardinality of this set is exactly the same as the number of back edges in the DFS tree as each back edge contributes to a cycle or a path. Also, as initially every vertex is unvisited, the first chain would be a cycle as it would end in the starting vertex. Schmidt proved the following theorem.

▶ **Theorem 3** ([32]). *Let D be a chain decomposition of a connected graph $G(V, E)$. Then G is 2-edge-connected if and only if the chains in D partition E. Also, G is 2-vertex-connected if and only if $\delta(G) \geq 2$ (where $\delta(G)$ denotes the minimum degree of G) and D_1 is the only cycle in the set D where D_1 is the first chain in the decomposition. An edge e in G is bridge if and only if e is not contained in any chain in D. A vertex v in G is a cut vertex if and only if v is the first vertex of a cycle in $D \setminus D_1$.*

Banerjee et al. [3] gave a space-efficient implementation of Theorem 3. En route they also provided an improved implementation for DFS (over Theorem 2) in sparse graphs ($m = O(n)$ edges). In particular, they proved the following,

▶ **Theorem 4** ([3]). *A DFS traversal of an undirected or directed graph G can be performed in $O(m+n)$ time using $O(m+n)$ bits. In the same amount of time and space, given a connected undirected graph G, we can perform a chain decomposition of G, and using that we can determine whether G is 2-vertex (and/or edge) connected. If not, in the same amount of time and space, we can compute all the bridges, cut vertices, and output 2-vertex (and edge) connected components.*

Kammer et al. [24] recently improved the space bound for finding cut vertices, still using linear time to $O(n + min\{m, n \lg\lg n\})$ bits.

3 DFS and applications using $O(n \lg(m/n))$ bits

One can easily implement the tests in Theorem 3 in $O(m)$ time using $O(m)$ words, by storing the DFIs and the entire chain decomposition, D. It is not too hard to improve the space to $O(n)$ words, still maintaining the $O(m)$ running time. Theorem 4 shows how to perform the tests using $O(m+n)$ bits and $O(m)$ time. The central idea there is to maintain the DFS tree using $O(m+n)$ bits using an unary encoding of the degree sequence of the graph. We first show how the space for the DFS tree representation can be improved to $O(n \lg m/n)$ bits.

▶ **Lemma 5.** *Given the adjacency array representation of an undirected graph G on n vertices with m edges, using $O(m)$ time, one can construct an auxiliary structure of size $O(n \lg(m/n))$ bits that can store a "pointer" into an arbitrary position within the adjacency array of each vertex. Also, updating any of these pointers (within the adjacency array) takes $O(1)$ time.*

Proof. We first scan the adjacency array of each vertex and construct a bitvector B as follows: starting with an empty bitvector B, for $1 \leq i \leq n$, if d_i is the length of the adjacency array of vertex v_i (i.e., its degree), then we append the string $0^{\lceil \lg d_i \rceil - 1}1$ to B. The length of B is $\sum_{i=1}^{n} \lceil \lg d_i \rceil$, which is bounded by $O(n \lg(m/n))$. We construct auxiliary structures to support *select* queries on B in constant time, using Theorem 1. We now construct another bitvector P of the same size as B, which stores pointers into the adjacency arrays of each vertex. The pointer into the adjacency array of vertex v_i is stored using the $\lceil \lg d_i \rceil$ bits in P from position $select(i-1, B) + 1$ to position $select(i, B)$, where $select(0, B)$ is defined to be 0. Now, using select operations on B and using constant time word-level read/write operations, one can access and/or modify these pointers in constant time. ◀

▶ **Lemma 6.** *Given a graph G with n vertices and m edges, in the adjacency array representation in the read-only memory model, the representation of a DFS tree can be stored using $O(n \lg(m/n))$ additional bits, which can be constructed on the fly during the DFS algorithm.*

Proof. We use the representation of Lemma 5 to store *parent* pointers into the adjacency array of each vertex. In particular, whenever the DFS outputs an edge (u, v), where u is the parent of v, we scan the adjacency array of v to find u and store a pointer to that position (within the adjacency array of v). The additional time for scanning the adjacency arrays adds upto $O(m)$ which would be subsumed by the running time of the DFS algorithm. ◀

We call the representation of the DFS tree of Lemma 6 as the *parent pointer representation*. Now given Lemma 5 and 6, we can simulate the DFS algorithm of [3] (Theorem 4) to obtain an $O(n \lg(m/n))$ bits and $O(m+n)$ time (see [3] for details) DFS implementation. The proof of Theorem 4 then uses another $O(m+n)$ bits to construct the chain decomposition of G and perform the tests as mentioned in Theorem 3, and we show here how even the space for the construction of a chain decomposition and performing the tests can be improved. We summarize our results in the following theorem below:

▶ **Theorem 7** (♠[2])**.** *A DFS traversal of an undirected or directed graph G can be performed in $O(m+n)$ time using $O(n \lg(m/n))$ bits of space. In the same amount of time and space, given a connected undirected graph G, we can perform a chain decomposition of G, and using that we can determine whether G is 2-vertex (and/or edge) connected. If not, in the same amount of time and space, we can compute all the bridges, cut vertices, and output 2-vertex (and edge) connected components.*

The above result for DFS improves the tradeoff result of Theorem 2 for relatively sparse graphs. Specifically, to achieve $O(m+n)$ time for DFS, the algorithm of Theorem 2 uses $O(n \lg\lg n)$ bits. This is $\Omega(n \lg(m/n))$ for all values of m where $m = O(n \lg n)$. Hence, for sparse graphs we obtain a better tradeoff. Also, it improves the space bound of Theorem 4, from $O(m+n)$ to $O(n \lg(m/n))$, while maintaining the same linear running time. In addition, it improves the algorithm for finding the cut vertices by Kammer et al. [24] from $O(n + \min\{m, n \lg\lg n\})$ to $O(n \lg(m/n))$.

4 Tree Cover and Space Efficient Construction

Central to all of our algorithms is a decomposition of the DFS tree. For this we use the well-known tree covering technique which was first proposed by Geary et al. [21] in the context of succinct representation of rooted ordered trees. The high level idea is to decompose the tree into subtrees called *minitrees*, and further decompose the mini-trees into yet smaller subtrees called *microtrees*. The microtrees are tiny enough to be stored in a compact table. The root of a minitree can be shared by several other minitrees. To represent the tree, we only have to represent the connections and links between the subtrees. Later He et al. [23] extended this approach to produce a representation which supports several additional operations. Farzan and Munro [18] modified the tree covering algorithm of [21] so that each minitree has at most one node, other than the root of the minitree, that is connected to the root of another minitree. This simplifies the representation of the tree, and guarantees that in each minitree, there exists at most one non-root node which is connected to (the root of) another minitree. The tree decomposition method of Farzan and Munro [18] is summarized in the following theorem:

▶ **Theorem 8** ([18])**.** *A rooted ordered tree with n nodes can be decomposed into $\Theta(n/L)$ minitrees of size at most $2L$ which are pairwise disjoint aside from the minitree roots. Furthermore, aside from edges stemming from the minitree root, there is at most one edge leaving a node of a minitree to its child in another minitree. The decomposition can be performed in linear time.*

In our algorithms, we apply Theorem 8 with $L = n/\lg n$. For this parameter L, since the number of minitrees is only $O(\lg n)$, we can represent the structure of the minitrees within the original tree (i.e., how the minitrees are connected with each other) using $O(\lg^2 n)$ bits. The decomposition algorithm of [18] ensures that each minitree has at most one 'child' minitree (other than the minitrees that share its root) in this structure. (We use this property crucially in our algorithms.) We refer to this as the *minitree-structure*.

Explicitly storing all the minitrees requires $\omega(n)$ bits. One way to represent them efficiently is to store them using any linear-bit encoding of a tree. But this representation doesn't allow us to efficiently compute the preorder numbers of the nodes, for example. Instead, we encode

[2] For proofs of results marked with (♠), please refer to the full version [10].

the entire tree structure using a linear-bit encoding, and store pointers into this encoding to represent the minitrees, as described below. We first encode the tree using the *balanced parenthesis* (BP) representation [28], summarized in the following theorem.

▶ **Theorem 9** ([28]). *Given a rooted ordered tree T on n nodes, it can be represented as a sequence of balanced parentheses of length $2n$. Using an additional $o(n)$ bits, we can support subtree size and navigational queries on T based on preorder and postorder.*

We now represent each minitree by storing pointers to the set of all *chunks* in the BP representation that together constitute the minitree. Farzan et al. [19, Lemma 2] show that each minitree is split into a constant number of consecutive chunks in the BP sequence. Hence, one can store a representation of the minitrees by storing a bitvector of length n that marks the starting positions of these chunks in the BP sequence, together with an $O(\lg^2 n)$-bit structure that stores, for each minitree, pointers to all the chunks in BP sequence indicating the starting positions of the chunks corresponding to the minitrees. The bit vector has $O(\lg n)$ 1's since there are $O(\lg n)$ minitrees, and each minitree is split into a constant number of chunks. We refer to the representation obtained using this tree covering (TC) approach as the TC representation of the tree.

The following lemma shows that we can construct the TC representation of the DFS tree of a given graph, using $O(n)$ additional bits.

▶ **Lemma 10** (♠). *Given a graph G on n vertices and m edges, if there is an algorithm that takes $t(n, m)$ time and $s(n, m)$ bits to perform DFS on G, then one can create the TC representation of the DFS tree in $t(n, m) + O(n)$ time, using $s(n, m) + O(n)$ bits.*

We use the following lemma in the description of our algorithms.

▶ **Lemma 11** (♠). *Let G be a graph, and T be its DFS tree. If there is an algorithm that takes $t(n, m)$ time and $s(n, m)$ bits to perform DFS on G, then, using $s(n, m)+O(n)$ bits, one can reconstruct any minitree given by its ranges in the BP sequence of the TC representation of T, along with the labels of the corresponding nodes in the graph in $O(t(n, m))$ time.*

5 Chain decomposition using $O(n)$ bits

Theorem 7 gives a chain decomposition algorithm that runs in $O(m + n)$ time, using $O(n \lg(m/n))$ bits. In this section we describe how one can implement Schmidt's chain decomposition algorithm described in Section 2 using only $O(n)$ bits using our partition of the DFS tree of Section 4. The main idea of our implementation is to process all the back edges out of each minitree, in the *preorder* of the minitrees. Also, when processing back edges out of a minitree τ, we process all the back edges that go from τ to the other minitrees in their *postorder*, processing all the edges from τ to a minitree τ_1 before processing any other back edges going out of τ to a different minitree. This requires us to go through all the edges out of each minitree at most $O(\lg n)$ (number of minitrees) times (although it is subsumed by the other parts of the computation, and doesn't affect the overall running time). Thus the order in which we process the back edges is different from the order in which we process them in Schmidt's algorithm, but we argue that this does not affect the correctness of the algorithm. In particular, we observe that Schmidt's algorithm correctly produces a chain decomposition

- even if we change the order in which we process vertices to any other order (instead of preorder), as long as we process a vertex v only after all its ancestors are also processed – for example, in level order.

This also implies that as long as we process the back edges coming to a vertex v (from any of its descendants) only after we process all the back edges going to any of its ancestors from any of v's descendants, we can produce a chain decomposition correctly. To process (i.e., to output the chain containing) a back edge (u, v) between a pair of minitrees τ_1 and τ_2, where u belongs to τ_1, v belongs to τ_2, and τ_1 is an anscestor of τ_2 in the minitree-structure, we first output the edge (u, v), and then traverse the path from v to the root of τ_2, outputting all the traversed edges. We then start another DFS to produce the minitree τ_p containing the parent p of the root of τ_2, and output the path from p to the root of τ_p, and continue the process untill we reach a vertex that has already been output as part of any chain (including the current chain). We maintain a bitvector of length n to keep track of the vertices that have been output as part of any chain, to perform this efficiently. A crucial observation that we use in bounding the runtime is that once we produce a minitree τ_p for a particular pair (τ_1, τ_2) of minitrees, we don't need to produce it again, as the root of τ_2 will be marked after the first time we output it as part of a chain. Also, once we generate the two minitrees τ_1 and τ_2, we go through all the vertices of τ_1 in preorder, and process all the edges that go between τ_1 and τ_2. For a particular minitree τ_1, once we process the back edges between τ_1 and all its descendant minitrees (i.e., descendants of the node corresponding to τ_1 in the minitree-structure), we finally process all the back edges that go within the minitree τ_1.

The time taken for the initial part, where we construct the DFS tree, decompose it into minitrees, and construct the auxiliary structures, is $O(m \lg\lg n)$, using Theorem 2 with $t(n) = \lg\lg n$. The running time of the algorithm is dominated by the cost of processing the back edges. For each pair of minitrees, we may, in the worst-case, need to generate $O(\lg n)$ minitrees. Since there are $O(\lg^2 n)$ pairs of minitrees, and since generating each minitree requires $O(m \lg\lg n)$ time (using the DFS algorithm), the total running time is $O(m \lg^3 n \lg\lg n)$. Thus, we obtain the following.

▶ **Theorem 12.** *Given an undirected graph G on n vertices and m edges, we can output a chain decomposition of G in $O(m \lg^3 n \lg\lg n)$ time using $O(n)$ bits of space.*

6 Finding cut vertices and biconnected components using $O(n)$ bits

A naïve algorithm to test for biconnectivity of a graph $G = (V, E)$ is to check if $(V \setminus \{v\}, E)$ is connected, for each $v \in V$. Using the $O(n)$ bits and $O(m + n)$ time BFS algorithm [3] for checking connectivity, this gives a simple $O(n)$ bits algorithm running in time $O(mn)$. Another approach is to use Theorem 12, as in the proof of Theorem 7, to test biconnectivity and output cut vertices in $O(m \lg^3 n \lg\lg n)$ time using $O(n)$.

Here we show that using $O(n)$ bits we can design an even faster algorithm running in $O(m \lg n \lg\lg n)$ time. If G is not biconnected, then we also show how to find out all the cut-vertices and biconnected components within the same time and space bounds. We implement the classical low-point algorithm of Tarjan [34]. Recall that, the algorithm computes for every vertex v, a value lowpoint$[v]$ which is defined as

$$\text{lowpoint}[v] = \min\{DFI(v) \cup \{\text{lowpoint}[s] |\ s \text{ is a child of } v\} \cup \{DFI(w) | (v, w) \text{ is a back-edge}\}\}$$

Tarjan proved that if vertex v is not the root, then v is a cut vertex if and only if v has a child s such that lowpoint$[s] \geq v$. (The root of a DFS tree is a cut vertex if and only if the root has more than one child.) Since the lowpoint values requires $\Omega(n \lg n)$ bits in the worst case, this poses the challenge of efficiently testing the condition for biconnectivity when only $O(n)$ bits. To deal with this, as in the case of the chain decomposition algorithm, we compute lowpoint values in $O(\lg n)$ batches using our tree covering algorithm. Cut vertices encountered in the process, if at all, are stored in a separate bitmap. We show that each batch can be processed in $O(m \lg \lg n)$ time using DFS, resulting in an overall runtime of $O(m \lg n \lg \lg n)$.

6.1 Computing lowpoint, cut vertices and biconnected components

We first obtain a TC representation of the DFS tree using the decomposition algorithm of Theorem 8 with $L = n/\lg n$. We then *process* each minitree, in the postorder of the minitrees in the minitree structure. To process a minitree, we compute the lowpoint values of each of the nodes in the minitree (except possibly the root). in overall $O(m)$ time. During the processing of any minitree, if we determine that a vertex is a cut vertex, we store this information by marking the corresponding node in a seperate bit vector. Each minitree can be reconstructed in $O(m \lg \lg n)$ time using Lemma 11. The lowpoint value of a node is a function of the lowpoints of all its children. However the root of a minitree may have children in other minitress. Hence for the root of the minitree, we store the partial lowpoint value (till that point) which will be used to update its value when all its subtrees have computed their lowpoint values (possibly in other minitrees). Computing the lowpoint values in each of the minitrees is done in a two step process. In the first step, we compute the DFI number of the deepest back edge node of each node in the minitree. Here the deepest back edge node of a node v is defined the smallest DFI value among the vertices w such that (v, w) is a back edge. Banerjee et al. show in [3] how one can compute the deepest back edge from any node while discussing a space-efficient implementation for computing a sparse spanning biconnected subgraph of a given biconnected graph. The corresponding algorithm makes two passes of DFS and hence takes $O(m \lg \lg n)$ time using $O(n)$ bits. We use that subroutine here to compute the deepest back edges. As there are only $\Theta(n/\lg n)$ nodes, we have space to store these values. In the second step, we do another DFS starting at the root of this minitree and compute the lowpoint values as we will do in a normal DFS (as deepest back edge values have been stored).

To compute the effect of the roots of the minitrees on the lowpoint computation, we store various $\Theta(\lg n)$ bit information with each of the $\Theta(\lg n)$ minitree roots including their partial/full lowpoint values, the rank of its first/last child in its subtree. After we process one minitree, we generate the next minitree, in postorder, and process it in a similar fashion and continue until we exhaust all the minitrees.

As we can mark all the cut vertices (if any) in a bitvector of length n, reporting them and computing 2-connected components is a routine task. Clearly we have taken $O(n)$ space and the total running time is $O(m \lg \lg n \lg n)$ as we run the DFS algorithm $O(\lg n)$ times. We formalize our theorem below.

▶ **Theorem 13.** *Given an undirected graph G, in $O(m \lg n \lg \lg n)$ time and $O(n)$ bits of space we can determine whether G is 2-vertex connected. If not, in the same amount of time and space, we can compute all the cut vertices of the graph and also output all the 2-vertex connected components.*

7 st-numbering

The *st*-ordering of vertices of an undirected graph is a fundamental tool for many graph algorithms, e.g. in planarity testing, graph drawing. The first linear-time algorithm for*st*-ordering the vertices of a biconnected graph is due to Even and Tarjan [17], and is further simplified by Ebert [14], Tarjan [35] and Brandes [8]. All these algorithms, however, preprocess the graph using depth-first search, essentially to compute lowpoints which in turn determine an (implicit) open ear decomposition. A second traversal is required to compute the actual *st*-ordering. All of these algorithms take $O(n \lg n)$ bits of space. We give an $O(n)$ bits implementation of Tarjan's [35] algorithm. We first describe the two pass classical algorithm of Tarjan without worrying about the space requirement. Let us denote by $p(v)$ the parent of the vertex v in the DFS tree. $DFI(v)$ and $lowpoint(v)$ have the usual meaning as defined previously. The first pass is a depth first search during which for every vertex v, $p(v)$, $DFI(v)$ and $lowpoint(v)$ are computed and stored. The second pass constructs a list L, which is initialized with $[s, t]$, such that if the vertices are numbered in the order in which they occur in L, then we obtain an *st*-ordering. In addition, we also have a sign array of n bits, intialized with sign$[s]$=-. The second pass is a preorder traversal of the spanning tree starting from the root s of the DFS tree. It is easy to see that the procedure runs in linear time using $O(n \lg n)$ bits of space. To make it space effcient, we use ideas similar to our biconnectivity algorithm. At a high level, we generate the lowpoint values of the first $n/\lg n$ vertices in depth first order and process them. Due to space restriction, we cannot store the list L as in Tarjan's algorithm; instead we use the BP sequence of the DFS tree and augment it with some extra information to 'encode' the final *st*-ordering, as described below.

Similar to our algorithms in the last two sections, this algorithm also runs in $O(\lg n)$ phases and in each phase it processes $n/\lg n$ vertices. In the first phase, to obtain the lowpoint values of the first $n/\lg n$ vertices in depth first order, we run as in our biconnectivity algorithm a procedure to store explicitly for these vertices their lowpoint values in an array. Also during the execution of the biconnectivity algorithm, the BP sequence is generated and stored in the BP array. We create two more arrays, of size n bits, that have one to one correspondence with the BP array. First array is for storing the sign for every vertex as in Tarjan's algorithm, and call it Sign. To simulate the effect of the list L, we create the second array, called P, where we store the relative position, i.e., "before" or "after", of every vertex with respect to its parent. Namely, if u is the parent of v, and v comes before (after, respectively) u in the list L in Algorithm 3, then we store $P[v] = b$ ($P[v] = a$, respectively). One crucial observation is that, even though the list L is dynamic, the relative position of the vertex v does not change with respect to the position of u, and is determined at the time of insertion of v into the list L (new verices may be added between u and v later). In what follows, we show how to use the BP sequence, and the array P to emulate the effect of list L and produce the *st*-ordering.

We first describe how to reconstruct the list L using the BP sequence and the P array. The main observation we use in the reconstruction L is that a node v appears in L after all the nodes in its child subtrees whose roots are marked with b in P, and also before all the nodes in its child subtrees whose roots are marked with a in P. Also, all the nodes in a subtree appear "together" (consecutively) in the list L. Thus by looking at the $P[v]$ values of all the children of a node u, and computing their subtree sizes, we can determine the position in L of u among all the nodes in its subtree. With this approach, we can reconstruct the list L, and hence output the *st*-numbers of all the nodes in linear time, if L can be stored in memory - which requires $O(n \lg n)$ bits. Now to perform this step with $O(n)$ bits, we repeat the whole process of reconstruction $\lg n$ times, where in the i-th iteration, we reproduce sublist $L[(i-1)n/\lg n + 1, \ldots, in/\lg n]$ – by ignoring any node that falls outside this range – and reporting all these nodes with *st*-numbers in the range $[(i-1)n/\lg n + 1, \ldots, in/\lg n]$. As each of these reconstruction takes $O(m \lg n \lg\lg n)$ time, we obtain the following.

▶ **Theorem 14.** *Given an undirected biconnected graph G on n vertices and m edges, and two distinct vertices s and t, we can output an st-numbering of all the vertices of G in $O(m \lg^2 n \lg\lg n)$ time, using $O(n)$ bits of space.*

7.1 An application of st-numbering

Given vertices $a_1, \cdots, a_k$ of a graph G and natural numbers $c_1, \cdots, c_k$ with $c_1 + \cdots + c_k = n$, we want to find a partition of V into sets $V_1, \cdots, V_k$ with $a_i \in V_i$ and $|V_i| = c_i$ for every i such that every set V_i induces a connected graph in G. This problem is called the *k-partitioning problem*. The problem is NP-hard even when $k = 2$, G is bipartite and the condition $a_i \in V_i$ is relaxed [13]. But, Györi [22] and Lovász [25] proved that such a partition always exists if the input graph is k-connected. Thus, let G be k-connected. In particular, if $k = 2$, the k-partitioning problem can be solved in the following manner [33, 29]: Let $v_1 := a_1$ and $v_n := a_2$, compute an $v_1 v_n$-numbering $v_1, v_2, \cdots, v_n$ and note that for any vertex v_i (in particular for $i = c_1$) the graphs induced by $v_1, \cdots, v_i$ and by $v_i, \cdots, v_n$ are connected. Thus applying Theorem 14, we obtain the following:

▶ **Theorem 15.** *Given an undirected biconnected graph G, two distinct vertices a_1, a_2, and two natural numbers c_1, c_2 such that $c_1 + c_2 = n$, we can obtain a partition (V_1, V_2) of the vertex set V of G in $O(m \lg^2 n \lg\lg n)$ time, using $O(n)$ bits of space, such that $a_1 \in V_1$ and $a_2 \in V_2$, $|V_1| = c_1$, $|V_2| = c_2$, and both V_1 and V_2 induce connected subgraph on G.*

8 Conclusions and Open Problems

We have given $O(m \lg^c n)$-time, $O(n)$-bit algorithms for a number of important applications of DFS. Obtaining linear time algorithms for them while maintaing $O(n)$ bits of space usage is an interesting open problem. Note that while there are $O(n)$-bit, $O(m + n)$-time algorithms for BFS, obtaining such an implementation for DFS is open. Another open problem is whether our $O(n)$-bit st-numbering algorithm can be used to design a $O(m \lg^c n)$ time planarity test using $O(n)$ bits of extra space.

References

1 T. Asano, T. Izumi, M. Kiyomi, M. Konagaya, H. Ono, Y. Otachi, P. Schweitzer, J. Tarui, and R. Uehara. Depth-first search using O(n) bits. In *25th ISAAC*, pages 553–564, 2014. doi:10.1007/978-3-319-13075-0_44.

2 T. Asano, D. G. Kirkpatrick, K. Nakagawa, and O. Watanabe. $\tilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *39th MFCS LNCS 8634*, pages 45–56, 2014. doi:10.1007/978-3-662-44465-8_5.

3 N. Banerjee, S. Chakraborty, and V. Raman. Improved space efficient algorithms for BFS, DFS and applications. In *22nd COCOON*, volume 9797, pages 119–130. Springer, 2016.

4 N. Banerjee, S. Chakraborty, V. Raman, S. Roy, and S. Saurabh. Time-space tradeoffs for dynamic programming in trees and bounded treewidth graphs. In *21st COCOON*, volume 9198 of *LNCS*, pages 349–360. Springer, 2015.

5 L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space-time tradeoffs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015. doi:10.1007/s00453-014-9893-5.

6 L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing a visibility polygon using few variables. *Comput. Geom.*, 47(9):918–926, 2014. doi:10.1016/j.comgeo.2014.04.001.

7 G. Barnes, J. Buss, W. Ruzzo, and B. Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

8 U. Brandes. Eager st-ordering. In *10th ESA*, pages 247–256, 2002. doi:10.1007/3-540-45749-6_25.

9 D. Chakraborty, A. Pavan, R. Tewari, N. V. Vinodchandran, and L. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *FSTTCS*, pages 585–595, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.585.

10 S. Chakraborty, V. Raman, and S. R. Satti. Biconnectivity, chain decomposition and *st*-numbering using O(n) bits. *CoRR*, abs/1606.08645, 2016. URL: http://arxiv.org/abs/1606.08645.

11 D. R. Clark. *Compact Pat Trees.* PhD thesis. University of Waterloo, Canada, 1996.

12 S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. Planar graph isomorphism is in log-space. In *24th CCC*, pages 203–214, 2009. doi:10.1109/CCC.2009.16.

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

14 J. Ebert. st-ordering the vertices of biconnected graphs. *Computing*, 30(1):19–33, 1983. doi:10.1007/BF02253293.

15 M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *51th FOCS*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

16 A. Elmasry, T. Hagerup, and F. Kammer. Space-efficient basic graph algorithms. In *32nd STACS*, pages 288–301, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

17 S. Even and R. E. Tarjan. Computing an *st*-numbering. *Theo. Comp. Sci.*, 2(3):339–344, 1976. doi:10.1016/0304-3975(76)90086-4.

18 A. Farzan and J. Ian Munro. Succinct representation of dynamic trees. *Theor. Comput. Sci.*, 412(24):2668–2678, 2011. doi:10.1016/j.tcs.2010.10.030.

19 A. Farzan, R. Raman, and S. S. Rao. Universal succinct representations of trees? In *36th ICALP*, pages 451–462, 2009. doi:10.1007/978-3-642-02927-1_38.

20 G. N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987. doi:10.1016/0022-0000(87)90002-X.

21 R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. *ACM Trans. Algorithms*, 2(4):510–534, 2006. doi:10.1145/1198513.1198516.

22 E. Győri. Partition conditions and vertex-connectivity of graphs. *Combinatorica*, 1(3):263–273, 1981.

23 M. He, J. I. Munro, and S. R. Satti. Succinct ordinal trees based on tree covering. *ACM Trans. Algorithms*, 8(4):42, 2012. doi:10.1145/2344422.2344432.

24 F. Kammer, D. Kratsch, and M. Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. In *41st MFCS*, 2016.

25 L. Lovasz. A homology theory for spanning tress of a graph. *Acta Mathematica Hungarica*, 30(3-4):241–251, 1977.

26 J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

27 J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165(2):311–323, 1996. doi:10.1016/0304-3975(95)00225-1.

28 J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

29 S. Nakano, M. S. Rahman, and T. Nishizeki. A linear-time algorithm for four-partitioning four-connected planar graphs. *Inf. Process. Lett.*, 62(6):315–322, 1997. doi:10.1016/S0020-0190(97)00083-5.

30 O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.

31 J. M. Schmidt. *Structure and constructions of 3-connected graphs*. PhD thesis, Free University of Berlin, 2010. URL: http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000022941.

32 J. M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Inf. Process. Lett.*, 113(7):241–244, 2013. doi:10.1016/j.ipl.2013.01.016.

33 J. M. Schmidt. The mondshein sequence. In *41st ICALP*, pages 967–978, 2014. doi:10.1007/978-3-662-43948-7_80.

34 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

35 R. E. Tarjan. Two streamlined depth-first search algorithms. *Fund. Inf.*, 9:85–94, 1986.

36 M. Tompa. Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations. *SIAM J. Comput.*, 11(1):130–137, 1982. doi:10.1137/0211010.

On (1,ϵ)-Restricted Max-Min Fair Allocation Problem[*][†]

T.-H. Hubert Chan[1], Zhihao Gavin Tang[2], and Xiaowei Wu[3]

1 **Department of Computer Sicence, The University of Hong Kong, Hong Kong**
hubert@cs.hku.hk
2 **Department of Computer Sicence, The University of Hong Kong, Hong Kong**
zhtang@cs.hku.hk
3 **Department of Computer Sicence, The University of Hong Kong, Hong Kong**
xwwu@cs.hku.hk

Abstract

We study the max-min fair allocation problem in which a set of m indivisible items are to be distributed among n agents such that the minimum utility among all agents is maximized. In the restricted setting, the utility of each item j on agent i is either 0 or some non-negative weight w_j. For this setting, Asadpour et al. [2] showed that a certain configuration-LP can be used to estimate the optimal value within a factor of $4+\delta$, for any $\delta > 0$, which was recently extended by Annamalai et al. [1] to give a polynomial-time 13-approximation algorithm for the problem. For hardness results, Bezáková and Dani [5] showed that it is NP-hard to approximate the problem within any ratio smaller than 2.

In this paper we consider the $(1,\epsilon)$-restricted max-min fair allocation problem, in which for some parameter $\epsilon \in (0,1)$, each item j is either heavy ($w_j = 1$) or light ($w_j = \epsilon$). We show that the $(1,\epsilon)$-restricted case is also NP-hard to approximate within any ratio smaller than 2. Hence, this simple special case is still algorithmically interesting.

Using the configuration-LP, we are able to estimate the optimal value of the problem within a factor of $3+\delta$, for any $\delta > 0$. Extending this idea, we also obtain a quasi-polynomial time $(3+4\epsilon)$-approximation algorithm and a polynomial time 9-approximation algorithm. Moreover, we show that as ϵ tends to 0, the approximation ratio of our polynomial-time algorithm approaches $3+2\sqrt{2} \approx 5.83$.

1998 ACM Subject Classification G.1.2 Approximation, G.1.6 Optimization, G.2.1 Combinatorics

Keywords and phrases Max-Min Fair Allocation, Hypergraph Matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.23

1 Introduction

We consider the *Max-Min Fair Allocation problem.* A problem instance is defined by (A, B, w), where A is a set of n agents, B is a set of m items and the utility of each item $j \in B$ perceived by agent $i \in A$ has weight w_{ij}. An allocation of items to agents is $\sigma : B \to A$ such that $\sigma(j) = i$ iff item j is assigned to agent i. The max-min fair allocation problem aims at finding an allocation such that the minimum total weight received by an agent $\min_{i \in A} \sum_{j \in \sigma^{-1}(i)} w_{ij}$ is maximized. The problem is also known as the Santa Claus Problem [4]. In the restricted

[*] A full version of the paper is available at https://arxiv.org/abs/1611.08060.

[†] This research is supported in part by the Hong Kong RGC grant 17202715.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 23; pp. 23:1–23:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

version of the problem, it is assumed that each item j has a fixed weight w_j such that for each $i \in A$ and $j \in B$, $w_{ij} \in \{0, w_j\}$, i.e., if an agent has non-zero utility for an item j, the utility is w_j. We focus on this paper the restricted version of the problem (restricted allocation problem) and refer to the problem with general weights the *unrestricted* allocation problem. For the restricted allocation problem, let $B_i = \{j \in B : w_{ij} > 0\}$ be the set of items agent i is interested in. For a collection of items $S \subseteq B$, let $w(S) = \sum_{j \in S} w_j$.

The problem can be naturally formulated as an integer program, with variable x_{ij} for each $i \in A$ and $j \in B$ indicating whether item j is assigned to agent i. Its linear program relaxation *Assignment-LP* (ALP) is shown as below.

$$\begin{array}{rrl} & \max & T \\ \text{s.t.} & \sum_{j \in B_i} x_{ij} w_j \geq T, & \forall i \in A \\ & \sum_{i \in A} x_{ij} \leq 1, & \forall j \in B \\ & x_{ij} \geq 0, & \forall i \in A, j \in B. \end{array}$$

Let OPT be the maximum value of the restricted allocation problem such that in the optimal allocation, every agent is assigned a set of items with total weight at least OPT. Bezáková and Dani [5] showed that any feasible solution x and T for the ALP can be rounded into an allocation such that every agent i receives at least $T - \max_{j \in B_i} w_j$ total value, which implies $\mathsf{OPT} \geq T^* - \max_{j \in B} w_j$, where T^* is the optimal value of the ALP. However, the above result does not yield any guarantee on the integrality gap. Actually, it can be easily shown that the integrality gap of ALP is unbounded since it is possible to have a feasible solution with $T > 0$ while $\mathsf{OPT} = 0$ (e.g., when $|B| < |A|$). It was shown in [5] that it is NP-hard to approximate the problem within any ratio smaller than 2 by a reduction from 3-dimensional matching.

To overcome the limitation of ALP, a stronger linear program called *Configuration-LP* (CLP) was proposed by Bansal and Sviridenko [4], in which an $O(\frac{\log n}{\log \log n})$-approximation algorithm was obtained for the restricted allocation problem. For any $T > 0$, we call an allocation a T-allocation if it assigns to every agent a set of items with total weight at least T.

▶ **Definition 1** (Bundles with Sufficient Utility)**.** For all $i \in A$, the collection of bundles with utility at least T for agent i is $C(i,T) := \{S \subseteq B_i : w(S) \geq T\}$.

The CLP is a feasibility LP associated with T indicating whether it is possible to (fractionally) assign to each agent one unit of bundle with sufficient utility. The LP ($\mathsf{CLP}(T)$) and its dual are shown as follows.

$$\begin{array}{rll} \textbf{Primal} & \min & 0 \\ \text{s.t.} & \sum_{S \in C(i,T)} x_{i,S} \geq 1, & \forall i \in A \\ & \sum_{i,S : j \in S \in C(i,T)} x_{i,S} \leq 1, & \forall j \in B \\ & x_{i,S} \geq 0, & \forall i \in A, S \in C(i,T). \end{array}$$

$$\begin{array}{rll} \textbf{Dual} & \max & \sum_{i \in A} y_i - \sum_{j \in B} z_j \\ \text{s.t.} & y_i \leq \sum_{j \in S} z_j, & \forall i \in A, S \in C(i,T) \\ & y_i \geq 0, & \forall i \in A \\ & z_j \geq 0, & \forall j \in B. \end{array}$$

Although $\mathsf{CLP}(T)$ has an exponential number of variables, it is claimed in [4] that the separation problem for the dual LP is the minimum knapsack problem: given a candidate dual solution (y, z), a violated constraint can be identified by finding an agent i and a configuration $S \in C(i,T)$ such that $y_i > \sum_{j \in S} z_j$. Hence, we can solve $\mathsf{CLP}(T)$ to any desired precision. Note that any feasible solution x of $\mathsf{CLP}(T)$ induces a feasible solution $\hat{x}$ for the ALP by setting $\hat{x}_{ij} = \sum_{S : j \in S \in C(i,T)} x_{i,S} \leq 1$ for all $i \in A$ and $j \in B$.

▶ **Definition 2** (Integrality Gap). Let T^* be the maximum value such that $\mathsf{CLP}(T^*)$ is feasible. The ratio $\frac{T^*}{\mathsf{OPT}}$ is known as the *integrality gap*.

Note that any upper bound c for the integrality gap implies that we can estimate the optimal value of the problem within a factor of $c + \delta$, for any $\delta > 0$. It is shown in [4] that the integrality gap of CLP for the unrestricted allocation problem is bounded by $O(\sqrt{n})$. By repeatedly using the Lovasz Local Lemma, Uriel Feige [8] proved that the integrality gap of CLP for the restricted allocation problem is bounded by a constant. The result was later turned into a constructive proof by Haeupler [11], who obtained the first constant approximation algorithm for the restricted allocation problem, although the constant is unspecified. The integrality gap of CLP was later shown in [2] to be no larger than 4 by a local search technique developed from Haxell [12] for finding perfect matchings in bipartite hypergraphs. However, the algorithm is not guaranteed to terminate in polynomial time. It is later shown by Polacek and Svensson [15] that a simple modification of the local search algorithm can improve the running time from $2^{O(n)}$ to $n^{O(\log n)}$, which implies a quasi-polynomial $(4 + \delta)$-approximation algorithm, for any $\delta > 0$. Very recently, Annamalai et al. [1] further extended the local search technique developed in [2, 15] for the restricted allocation problem and obtained a polynomial-time 13-approximation algorithm for the problem.

1.1 The $(1, \epsilon)$-Restricted Allocation Problem

We consider in this paper the $(1, \epsilon)$-restricted allocation problem, in which for some $\epsilon \in (0, 1)$, each item $j \in B$ is either *heavy* ($w_j = 1$) or *light* ($w_j = \epsilon$). As the simplest case of the allocation problem, the problem is not well understood. The current best approximation results for the problem are for the restricted allocation problem. Indeed, we believe that a better understanding of the $(1, \epsilon)$-restricted setting will shed light on improving the restricted (and even the unrestricted) allocation problem.

The $(1, \epsilon)$-restricted setting has been studied under different names. Golovin [10] studied the "Big Goods/Small Goods" max-min allocation problem, which is exactly the same as the problem we consider in this paper, in which a small item has weight either 0 or 1 for each agent; a big item has weight either 0 or $x > 1$ for each agent. They gave an $O(\sqrt{n})$-approximation algorithm for this problem and proved that it is NP-hard to approximate the "Big Goods/Small Goods" max-min allocation problem within any ratio smaller than 2 by giving a hard instance with $x = 2$. We show in this paper that the inapproximability result holds for any fixed $x \geq 2$ by generalizing the hardness instance shown in [5]. Later Khot and Ponnuswami [13] generalized the "Big Goods/Small Goods" setting and considered the $(0, 1, U)$-max-min allocation problem with sub-additive utility function in which the weight of an item to an agent is either 0, 1 or U for some $U > 1$ and obtained an $\frac{n}{\alpha}$-approximation algorithm with $m^{O(1)} n^{O(\alpha)}$ running time, for any $\alpha \leq \frac{n}{2}$. Note that in their setting an item can have weight 1 for an agent and U for another. In the seminal paper, Bansal and Sviridenko [4] obtained an $O(\frac{\log n}{\log \log n})$-approximation algorithm for the restricted allocation problem by first reducing the problem to the $(1, \epsilon)$-restricted case for an arbitrarily small $\epsilon > 0$ while losing a constant factor on the approximation ratio, and then proving an $O(\frac{\log n}{\log \log n})$-approximation algorithm for the $(1, \epsilon)$-restricted case.

The max-min fair allocation problem is closely related to the problem of scheduling jobs on *unrelated machines* to minimize *makespan*, which we call the *min-max allocation problem*. The problem has the same input as the max-min fair allocation problem but aims at finding an allocation that minimizes $\max_{i \in A} \sum_{j \in \sigma^{-1}(i)} w_{ij}$. Lenstra et al. [14] showed a

2-approximation algorithm for the min-max allocation problem by rounding the ALP for the problem. Applying the techniques developed for the max-min fair allocation problem, Svensson [16] gave a $\frac{5}{3}+\epsilon$ upper bound for the CLP's integrality gap of the $(1,\epsilon)$-restricted min-max allocation problem and then extended it to a 1.9412 upper bound for the general case. However, their algorithm is not known to converge in polynomial time. Recently Chakrabarty et al. [7] obtained the first $(2-\delta)$-approximation algorithm for the $(1,\epsilon)$-restricted min-max allocation problem, for some constant $\delta > 0$. They considered the case when ϵ is close to 0 since it is easy to obtain a $(2-\epsilon)$-approximation algorithm for the $(1,\epsilon)$-restricted min-max allocation problem.

Since the $(1,\epsilon)$-restriction is considered in the community to be interesting for the min-max setting, in this paper we consider this restriction for the max-min setting.

1.2 Summary of Our Results

We first show that we can slightly improve the hardness result of Golovin [10] for the $(1,\epsilon)$-restricted allocation problem. Note that in the unweighted case ($\epsilon = 1$), the problem can be solved in polynomial time by combining the max-flow computation between A and B, with a binary search on the optimal value. The above algorithm for the unweighted case actually provides a trivial $\frac{1}{\epsilon}$-approximation algorithm for the $(1,\epsilon)$-restricted allocation problem. Hence, we have a polynomial-time algorithm with ratio $\frac{1}{\epsilon} < 2$ for the problem when $\epsilon > 0.5$.

▶ **Theorem 3** (Inapproximability). *For any $\epsilon \leq 0.5$, it is* NP*-hard to approximate the $(1,\epsilon)$-restricted allocation problem within any ratio smaller than* 2*.*

The proof is included in our full version. Our reduction shows that it is NP-hard to estimate the optimal value of the problem within any ratio smaller than 2. The above hardness result implies that the integrality gap of CLP(T) is at least 2 unless P = NP. However, we can remove the P $\neq$ NP assumption by giving a hard instance explicitly (in the full version).

For the restricted allocation problem, the best hardness result on the approximation ratio is 2 while the best upper bound for the integrality gap of CLP(T) is 4. It is not known which bound (or none) is tight. As a step towards closing this gap, we analyze the integrality gap of CLP(T) for the $(1,\epsilon)$-restricted case and show that the upper bound of 4 is not tight in this case (Section 2). Our upper bound implies that in polynomial time we can estimate OPT for the $(1,\epsilon)$-restricted allocation problem within a factor of $3+\delta$, for any $\delta > 0$.

▶ **Theorem 4** (Integrality Gap). *The integrality gap of the configuration-LP of the $(1,\epsilon)$-restricted allocation problem is at most* 3*.*

We also observe that by picking the "closest addable edge", the running time of the local search algorithm can be improved to quasi-polynomial (Section 3). The idea was first used by Polacek and Svensson [15] to obtain the $(4+\delta)$-approximation algorithm for the restricted allocation problem. However, instead of constructing feasible dual solutions for CLP(T), our analysis is based on the assumption of $T \leq$ OPT and is a direct extension of our proof on the integrality gap of CLP(T).

▶ **Theorem 5** (Quasi-Polynomial-Time Approximation). *There exists a $(3+4\epsilon)$-approximation algorithm for the $(1,\epsilon)$-restricted allocation problem that runs in $n^{O(\frac{1}{\epsilon}\log n)}$ time.*

We further extend the quasi-polynomial approximation algorithm by combining the lazy update idea of [1] to obtain a polynomial approximation algorithm (Section 4).

▶ **Theorem 6** (Polynomial-Time Approximation). *For any $\epsilon \in (0, 1)$, there exists a polynomial-time 9-approximation algorithm for the $(1, \epsilon)$-restricted allocation problem. Moreover, the approximation ratio approaches $3 + 2\sqrt{2} \approx 5.83$ as ϵ tends to 0.*

Interestingly, while our quasi-polynomial- and polynomial-time algorithms are extended from the integrality gap analysis by combining ideas on improving the running time of local search, unlike existing techniques, our algorithms and analysis do not directly use the feasibility of $\mathsf{CLP}(T)$. To lead to contradictions, existing results [15, 1] tried to construct feasible dual solutions for $\mathsf{CLP}(T)$ with positive objective values (which implies the infeasibility of $\mathsf{CLP}(T)$). In contrast, our analysis shows that as long as $T \leq \mathsf{OPT}$, our algorithms terminate with the claimed approximation ratios, which simplifies the analysis and is an advantage in some cases when $\mathsf{CLP}(T)$ cannot be applied, e.g., when the utility function is sub-additive [13].

1.3 Other Related Work

Unrestricted Allocation Problem. Based on Bansal and Sviridenko's proof [4] of $O(\sqrt{n})$-integrality gap for the unrestricted allocation problem, Asadpour and Saberi [3] achieved an $\tilde{O}(\sqrt{n})$-approximation algorithm. The current best approximation result for the problem is an $\tilde{O}(n^{\delta})$-approximation algorithm that runs in time $n^{O(\frac{1}{\delta})}$, for any $\delta = \Omega(\frac{\log\log n}{\log n})$, obtained by Chakrabarty et al. [6].

Other Utility Functions. The max-min fair allocation problem with different utility functions has also been considered. Golovin [10] gave an $(m - n + 1)$-approximation algorithm for the problem when the utility functions of agents are submodular. We note that their result can also be extended to sub-additive utility functions. Khot and Ponnuswami [13] also considered the problem with sub-additive utility functions and obtained a $(2n-1)$-approximation algorithm. Later Goemans and Harvey [9] obtained an $\tilde{O}(n^{\frac{1}{2}+\delta})$-approximation for submodular max-min allocation problem in $n^{O(\frac{1}{\delta})}$ time using the $\tilde{O}(n^{\delta})$-approximation algorithm by Chakrabarty et al. [6] as a black box.

2 Integrality Gap for Configuration LP

We show in this section that for the $(1, \epsilon)$-restricted allocation problem, the integrality gap of the CLP is at most 3. Fix $T > 0$ be such that $\mathsf{CLP}(T)$ is feasible.

We show that whenever $\mathsf{CLP}(T)$ is feasible, there exists a $\frac{T}{3}$-allocation (hence $\mathsf{OPT} \geq \frac{T}{3}$), which implies an integrality gap of at most 3. Given any solution x for $\mathsf{CLP}(T)$ and the induced ALP solution $\hat{x}$, for all $\hat{x}_{ij} = 0$, we can remove j from B_i (pretending that i is not interested in j). This operation will preserve the feasibility of x while (possibly) decreasing OPT, which could only enlarge the integrality gap. From now on we assume that a positive fraction of every item in B_i is assigned to agent i.

Assumption on T: To achieve a $\frac{T}{3}$-allocation, we can assume that $T < \frac{3}{2}$; otherwise, we can get a $T - 1 \geq \frac{T}{3}$ allocation by rounding the ALP solution $\hat{x}$ [5]. We can further assume $T \geq 1$ since otherwise we can set all weights $w_j \geq T$ to T (which does not change $\mathsf{CLP}(T)$) and scale all weights so that the maximum weight is 1. From now on, we assume that $T \in [1, \frac{3}{2})$ and $\mathsf{CLP}(T)$ is feasible.

Let $k = \lceil \frac{T}{\epsilon} \rceil$. Note that every bundle consisting solely of light items must contain at least k items to have sufficient utility. For all $i \in A$, let $B_i^1 = \{j \in B_i : w_j = 1\}$ be the set of heavy items and $B_i^\epsilon = \{j \in B_i : w_j = \epsilon\}$ be the set of light items. Our algorithm fixes an integer $r = \lceil \frac{k}{3} \rceil$ and tries to assign items such that each agent i receives either a heavy item $j \in B_i^1$ or r light items in B_i^ϵ. Suppose we are able to find such an allocation, then the integrality gap is $\frac{T}{r\epsilon} \leq 3$.

2.1 Getting a "Minimal" Solution

Let x^* be a solution for $\mathsf{CLP}(T)$. We create another solution x (which might not be feasible) as follows. Initialize $x_{i,S} = 0$ for all $i \in A$ and $S \subseteq B_i$. For all $x^*_{i,S} > 0$, where $S \in C(i,T)$,

1. if $S' = S \cap B_i^1 \neq \emptyset$, then set $x_{i,S'} = x^*_{i,S}$;
2. otherwise, S contains only light items and set $x_{i,S} = x^*_{i,S}$.

Note that for each $i \in A$ we have the following properties on x:

1. (heavy/light configurations) if $x_{i,S} > 0$, then $(S \subseteq B_i^1 \wedge |S| \geq 1)$ or $(S \subseteq B_i^\epsilon \wedge |S| \geq k)$.
2. (covering constraint for agent) $\sum_{S \subseteq B_i} x_{i,S} = \sum_{S \in C(i,T)} x^*_{i,S} \geq 1$.
3. (packing constraint for item) for all $j \in B$: $\sum_{i,S:j\in S} x_{i,S} \leq \sum_{i,S:j \in S \in C(i,T)} x^*_{i,S} \leq 1$.

Now we construct a hypergraph $H(A \cup B, E)$ as follows: for all $x_{i,S} > 0$,

1. if $S \subseteq B_i^1$, then for each $j \in S$, add $\{i,j\}$ to E (we call such an edge **heavy**);
2. otherwise for each $S' \subseteq S$ and $|S'| = r$, add $\{i\} \cup S'$ to E (we call such an edge **light**).

A matching $M \subseteq E$ is a collection of disjoint edges. Note that any perfect matching of H that matches all nodes in A provides an allocation that assigns each $i \in A$ either a heavy item or r light items. For all $F \subseteq E$, let $A(F) = A \cap (\bigcup_{e \in F} e)$ and $B(F) = B \cap (\bigcup_{e\in F} e)$.

2.2 Finding a Perfect Matching

Recall that the existence of a perfect matching in $H(A \cup B, E)$ such that every agent in A is matched implies that the integrality gap of $\mathsf{CLP}(T)$ is at most 3.

▶ **Theorem 7.** *The above hypergraph $H(A \cup B, E)$ has a perfect matching.*

Proof. Given a partial matching $M \subseteq E$, we show how to extend its cardinality by one if $|M| \leq |A| - 1$. Let $i_0 \in A \backslash A(M)$ be an agent not matched by M. For the initial step, suppose X_1 contains an arbitrary edge e_1 with $A(e_1) = \{i_0\}$ and $Y_1 = \mathsf{blocking}(e_1) = \{f \in M : B \cap e_1 \cap f \neq \emptyset\}$ be the blocking edges of e_1. If $\mathsf{blocking}(e_1) = \emptyset$, then we can add edge e_1 to the matching. Assume $\mathsf{blocking}(e_1) \neq \emptyset$.

For the recursive step, suppose we already have edges X_t (where $t = |X_t|$) and Y_t, which together form an alternating tree rooted at i_0. We consider adding the $(t+1)$-st edge to X_t as follows. An edge $e \in E$ is **addable** if (1) $A(e) \in A(X_t \cup Y_t)$; (2) $B(e) \cap B(X_t \cup Y_t) = \emptyset$. If such an edge e_{t+1} exists and $\mathsf{blocking}(e_{t+1}) \neq \emptyset$, let $X_{t+1} = X_t \cup \{e_{t+1}\}$ and $Y_{t+1} = Y_t \cup \mathsf{blocking}(e_{t+1})$. If $\mathsf{blocking}(e_{t+1}) = \emptyset$, then we **contract** X_t by swapping out blocking edges (the details of contraction will be discussed later). The contraction operation guarantees that every addable edge has at least one blocking edge.

▶ **Claim 8** (Always Addable). *There is always an addable edge e_{t+1}.*

Proof. Let $P = A(X_t \cup Y_t)$ be the agents in the tree. Note that $|P| = |Y_t| + 1$ since each agent $i \neq i_0$ in P has an unique blocking edge that **introduces** i.

Let X_t^1 (Y_t^1) be the heavy edges and X_t^ϵ (Y_t^ϵ) be the light edges of X_t (Y_t).

We have $|X_t^1| = |Y_t^1|$ since heavy edges can only be blocked by heavy edges. We have $|X_t^\epsilon| \leq |Y_t^\epsilon|$ since each addable edge has at least one blocking edge.

Let $x_P^1 = \sum_{i\in P}\sum_{S\subseteq B_i^1} x_{i,S}$ be the total units of heavy bundles assigned to P by x, which is a lower bound for the total number of heavy items $B_P^1 = \cup_{i\in P} B_i^1$ agents in P are interested in since

$$x_P^1 = \sum_{i\in P}\sum_{S\subseteq B_i^1} x_{i,S} \le \sum_{i\in P}\sum_{S\subseteq B_i^1}\sum_{j\in S} x_{i,S} = \sum_{j\in B_P^1}\sum_{i,S:j\in S\subseteq B_i^1} x_{i,S} \le |B_P^1|.$$

Let $x_P^\epsilon = \sum_{i\in P}\sum_{S\subseteq B_i^\epsilon} x_{i,S}$ be the total units of light bundles assigned to P by x. By construction of x, we have

$$\sum_{i\in P}\sum_{S\subseteq B_i} x_{i,S} = \sum_{i\in P}\sum_{S\subseteq B_i^1} x_{i,S} + \sum_{i\in P}\sum_{S\subseteq B_i^\epsilon} x_{i,S} = x_P^1 + x_P^\epsilon \ge |P|.$$

Since $|Y_t^1|$ heavy items are already introduced in the tree, if $x_P^1 > |Y_t^1|$, then there must exist an addable heavy edge for some $i \in P$. If $x_P^1 \le |Y_t^1|$, then we have $x_P^\epsilon \ge |P| - x_P^1 \ge |Y_t^\epsilon| + 1 \ge |X_t^\epsilon| + 1$. Note that every light addable edge has at most $r-1$ unblocked items, the total number of light items in the tree is

$$|B^\epsilon(X_t \cup Y_t)| \le (r-1)|X_t^\epsilon| + r|Y_t^\epsilon| \le (2r-1)(x_P^\epsilon - 1) < (2r-1)x_P^\epsilon. \tag{1}$$

For each $i \in P$ and $S \subseteq B_i^\epsilon$, if $x_{i,S} > 0$, then by construction we have $|S| \ge k \ge 3r-2$. If i has no more addable light edges (has at most $r-1$ **unintroduced** light items in H), then at least $\sum_{S\subseteq B_i^\epsilon}(|S| - (r-1))x_{i,S} \ge (2r-1)\sum_{S\subseteq B_i^\epsilon} x_{i,S}$ units of configurations of light items appear in the tree. If there is no more addable light edges for all $i \in P$, then we have

$$|B^\epsilon(X_t \cup Y_t)| \ge \sum_{j\in B^\epsilon(X_t\cup Y_t)}\sum_{i,S:j\in S\subseteq B_i^\epsilon} x_{i,S} \ge (2r-1)\sum_{i\in P}\sum_{S\subseteq B_i^\epsilon} x_{i,S} = (2r-1)x_P^\epsilon,$$

which is a contradiction to (1). ◀

Contraction: If $\mathsf{blocking}(e_{t+1}) = \emptyset$, then we remove the blocking edge f that introduces $A(e_{t+1})$ from the matching and include e_{t+1} into the matching. Both e_{t+1} and f are removed from the tree. We also remove all edges added after f since they can possibly be introduced by $A(f)$. We call this operation a contraction on e_{t+1}. Note that this operation reduces the size of $\mathsf{blocking}(e')$ by one, for the edge e' that is blocked by f. If $\mathsf{blocking}(e') = \emptyset$ after that, then we further contract e' recursively. After all contractions, suppose the remaining addable edges in the tree are $e_1, e_2, \ldots, e_{t'}$ (ordered by the time they are added to the tree), we set $t = t'$, $X_{t'}$ and $Y_{t'}$ be the addable and blocking edges, respectively.

Signature: At any moment before including an addable edge (suppose there are t addable edges in the tree), let $s_i = |\mathsf{blocking}(e_i)|$ for all $i \in [t]$. Let $\mathbf{s} = (s_1, s_2, \ldots, s_t, \infty)$ be the signature of the tree. Then, we have the following.

1. The lexicographical value of $\mathbf{s}$ reduces after each iteration. If there is no contraction in the iteration, then in the signature, the $(t+1)$-st coordinate decreases from ∞ to s_{t+1}, while s_i remains the same for all $i \le t$. Otherwise, let e_i be the edge whose number of blocking edges is reduced by one but remains positive in the contraction phase. Then, we have s_i is reduced by one while s_j remains the same for all $j < i$.
2. There are at most 2^n different signatures since $\sum_{i\in[t]} s_i \le n$ and $t \le n$.

Since an addable edge can be found in polynomial time and the contraction operation stops in polynomial time, a perfect matching can be found in $n \cdot 2^n \cdot \mathsf{poly}(n)$ time. ◀

3 Quasi-Polynomial-Time Approximation Algorithm

We show in this section that a simple modification on the algorithm for finding a perfect matching in Section 2 can dramatically improve the running time from $2^{O(n)}$ to $n^{O(\log n)}$. Assume that $T \leq \mathsf{OPT}$. Note that in this case we can still assume $T \in [1, \frac{3}{2})$.

Note that combining the polynomial time $\frac{1}{\epsilon}$-approximation algorithm, the approximation ratio we obtain in quasi-polynomial time is $\min\{\frac{1}{\epsilon}, 3+4\epsilon\} \leq 4$ for all $\epsilon \in (0,1)$. Moreover, when $\epsilon \to 0$ (in which case the problem is still $(2-\delta)$-inapproximable), our approximation ratio approaches the integrality gap upper bound 3.

Proof of Theorem 5. Let T be a guess of OPT and $k = \lceil \frac{T}{\epsilon} \rceil$. Since the statement trivially holds for $\epsilon \geq \frac{1}{4}$ ($\frac{1}{\epsilon} \leq 3+4\epsilon$). We assume that $\epsilon < \frac{1}{4}$ (hence $k \geq 5$). We show that if $T \leq \mathsf{OPT}$, then we can find in quasi-polynomial time a $\frac{T}{3+4\epsilon}$-allocation; if no such allocation is found after the time limit, then T should be decreased as in binary search. Let $r = \lceil \frac{k}{3+4\epsilon} \rceil$. It suffices to show that a feasible allocation that assigns to each agent i either a heavy item in B_i^1 or r light items in B_i^ϵ can be found in $n^{O(\log n)}$ time, for any $\epsilon < \frac{1}{4}$. We define a heavy edge $\{i,j\}$ for each $j \in B_i^1$ and a light edge $\{i\} \cup S$ for each $S \subseteq B_i^\epsilon$ and $|S| = r$.

As in the proof of Theorem 7, we wish to find a perfect matching for all agents in A. Suppose in some partial matching, there is an unmatched agent i_0 and we construct an alternating tree rooted at i_0. For each addable edge e, we denote by $d(e)$ the number of light edges (including e) in the **path** from i_0 to e in an alternating tree rooted at i_0. Note that a path is a sequence of edges alternating between addable edges and blocking edges. The algorithm we use in this section is the same as previous, except that when there are addable edges, we always pick the one e such that the **distance** $d(e)$ is **minimized**. We show that in this case there is always an addable edge within distance $O(\frac{1}{\epsilon}\log n)$.

Let X_i and Y_i be the set of addable edges and blocking edges at distance i from i_0, respectively. Note that $Y_i = \emptyset$ for all odd i since light blocking edge must be introduced due to light addable edge. Moreover, since on the path from i_0 to every addable edge $e \in X_i$, the light edge (if any) closest to e must be a blocking edge (of even distance), we know that X_{odd} contains only light edges and X_{even} contains only heavy edges. Let Y_i^1 and Y_i^ϵ be the set of heavy edges and light edges in Y_i, respectively.

Let $L = \lceil \log_{1+\frac{\epsilon}{10}} n \rceil$. It suffices to prove Claim 9 below since it implies that

$$|Y^\epsilon_{\leq 2L+2}| > (1+\frac{\epsilon}{10})|Y^\epsilon_{\leq 2L}| > (1+\frac{\epsilon}{10})^L |Y_2^\epsilon| \geq n,$$

which is a contradiction and implies that there is always an addable edge within distance $2L+1$. Note the the last inequality also comes from Claim 9 since otherwise $|Y_2^\epsilon| = 0$ and $|Y_4^\epsilon| = 0 \leq \frac{\epsilon}{10}|Y_2^\epsilon|$ would be a contradiction.

▶ **Claim 9.** *For all $l \in [L]$, when there is no more addable edge within distance $2l+1$, we have $|Y^\epsilon_{2l+2}| > \frac{\epsilon}{10}|Y^\epsilon_{\leq 2l}|$.*

Proof. Let $P = A(X_{\leq 2l} \cup Y_{\leq 2l}) = A(Y_{\leq 2l}) \cup \{i_0\}$. Since there is no more addable edges within distance $2l+1$, we know that every agent $i \in P$ does not admit any addable edges. Hence for each $i \in P$, all heavy items in B_i^1 are already included in $B^1(X^1_{\leq 2l})$ and at most $r-1$ light items in B_i^ϵ are not included in $B^\epsilon(X^\epsilon_{\leq 2l+1} \cup Y^\epsilon_{\leq 2l+2})$.

Since $T \leq \mathsf{OPT}$, we know that at least $|P| - |B^1(X^1_{\leq 2l})| = |Y^\epsilon_{\leq 2l}| + 1$ agents in P are assigned only light items. Hence, out of at least k light items assigned to each of those agents, at least $k-r+1$ items must be included in $B^\epsilon(X^\epsilon_{\leq 2l+1} \cup Y^\epsilon_{\leq 2l+2})$, which means

$$|B^\epsilon(X^\epsilon_{\leq 2l+1} \cup Y^\epsilon_{\leq 2l+2})| \geq (k-r+1)(|Y^\epsilon_{\leq 2l}|+1).$$

Assume $|Y^\epsilon_{2l+2}| \leq \frac{\epsilon}{10}|Y^\epsilon_{\leq 2l}|$, we have $|Y^\epsilon_{\leq 2l+2}| \leq (1+\frac{\epsilon}{10})|Y^\epsilon_{\leq 2l}|$. Since every addable edge contains at most $r-1$ unblocked items (items not used by M), we have the following upper bound for the number of light items in the tree:

$$|B^\epsilon(X^\epsilon_{\leq 2l+1} \cup Y^\epsilon_{\leq 2l+2})| \leq (r-1)|X^\epsilon_{\leq 2l+1}| + r|Y^\epsilon_{\leq 2l+2}| \leq (1+\frac{\epsilon}{10})(2r-1)|Y^\epsilon_{\leq 2l}|.$$

For $\epsilon < \frac{1}{4}$, $T \in [1, \frac{3}{2})$, $k = \lceil \frac{T}{\epsilon} \rceil$ and $r = \lceil \frac{k}{3+4\epsilon} \rceil$, we have $k \geq 3\lceil \frac{k}{3} \rceil - 2 \geq 3r - 2$. Suppose $k = 3r-2$, then we have

$$k = 3\lceil \frac{k}{3+4\epsilon} \rceil - 2 \leq \frac{3k}{3+4\epsilon} + 1 = k - (\frac{4\epsilon k}{3+4\epsilon} - 1),$$

which is a contradiction since $\frac{4\epsilon k}{3+4\epsilon} > 1$. Hence we have $k \geq 3r-1$, which implies $k - r + 1 \geq (3r-1)-(r-1) = 2r \geq (1+\frac{\epsilon}{10})(2r-1)$ since $r \leq \frac{5}{\epsilon}$. Hence we have a contradiction. ◀

At any moment before adding an addable edge, suppose we have constructed $X_{\leq 2l}$ and $Y_{\leq 2l}$. By the above argument we have $2l \leq 2L = O(\frac{1}{\epsilon}\log n)$. Let $a_i = -|X_i|$. Let $|Y^1_i| = b_i$ and $|Y^\epsilon_i| = b_{i-1}$ for all even i. Let $\mathbf{s} = (a_0, b_0, a_1, b_1, \ldots, a_{2l}, b_{2l}, \infty)$ be the **signature** of the alternating tree. We show that $\mathbf{s}$ is lexicographically decreasing accross all iterations.

No contraction: Suppose we added an addable edge e with $\mathsf{blocking}(e) \neq \emptyset$, then e will be included in $X_{\leq 2l}$ or a newly constructed X_{2l+1}, in both cases the lexicographic value of $\mathbf{s}$ decreases since the last modified coordinate decreases.

Contraction: Suppose the newly added edge has no blocking edge, then in the contraction, let $f \in Y^\epsilon_{2i}$, which must be light, be the last blocking edge that is removed. Since b_{2i-1} decreases while a_j (for all $j \leq 2i-1$) and b_j (for all $j \leq 2i-2$) do not change, the lexicographic value of $\mathbf{s}$ decreases.

Since $L = O(\frac{1}{\epsilon}\log n)$, there are $n^{O(\frac{1}{\epsilon}\log n)}$ different signatures. Since an addable edge can be found in polynomial time and the contraction operation stops in polynomial time, the running time of the algorithm is $n \cdot \mathsf{poly}(n) \cdot n^{O(\frac{1}{\epsilon}\log n)} = n^{O(\frac{1}{\epsilon}\log n)}$. ◀

4 Polynomial-Time Approximation Algorithm

We give a polynomial-time approximation algorithm in this section. Based on the previous analysis, to improve the running time from $n^{O(\log n)}$ to $n^{O(1)}$, we need to bound the total number of iterations (signatures) by $\mathsf{poly}(n)$. On a high level, our algorithm is similar to that of Annamalai et al. [1]: we apply the idea of lazy update and greedy player such that after each iteration, either a new layer is constructed or the size of the highest layer changed is reduced by a constant factor. However, instead of constructing feasible dual solutions, we extend the charging argument used in the previous sections on counting the number of light items in the tree to prove the exponential growth property of the alternating tree.

In binary search, let T be a guess of OPT. As explained earlier, we can assume $T \in [1, \frac{3}{2})$. Let $k = \lceil \frac{T}{\epsilon} \rceil$. Our algorithm aims at assigning to each agent either a heavy item or r light items, for some fixed $r \leq \frac{k}{2}$ when $T \leq \mathsf{OPT}$. Such an allocation gives a $\frac{k}{r}$-approximation. Let $p \in (r, k)$ be an integer parameter. Let $0 < \mu \ll 1$ be a very small constant, e.g., $\mu = 10^{-10}$.

As before, for each $i \in A$, we call $\{i, j\}$ a heavy edge for $j \in B^1_i$, and $\{i\} \cup S$ a light edge if $S \subseteq B^\epsilon_i$. However, in this section, we use two types of light edges: either $|S| = p$ (addable edges) or $|S| = r$ (blocking edges). Let M be a maximum matching between A and B^1. We can regard M as a partial allocation that assigns maximum number of heavy items. Let i_0

be an unmatched node in M. We can further assume that every heavy item is interesting to at least 2 agents since otherwise we can assign it to the only agent and remove the item and the agent from the problem instance. We use " $+$ " and " $-$ " to denote the inclusion and exclusion of singletons in a set, respectively.

4.1 Flow Network

Let $G(A \cup B^1, E_M)$ be a **directed** graph uniquely defined by M as follows. For all $i \in A$ and $j \in B^1_i$, if $\{i, j\} \in M$ then $(j, i) \in E_M$, otherwise $(i, j) \in E_M$. We can interpret the digraph as the residual graph of the "interest" network (a digraph with directed edges from each i to $j \in B^1_i$) with current flow M. The digraph G has the following properties

- every $i \in A$ has in-degree ≤ 1, every $j \in B^1$ has out-degree ≤ 1 and in-degree ≥ 1.
- all heavy items reachable from $i \in A$ with in-degree 0 must have out-degree 1 (otherwise we can augment the size of M by one).

Given two sets of light edges Y and X ($A(Y)$ and $A(X)$ do not have to be disjoint), let $f(Y, X)$ denote the maximum number of node-disjoint paths in $G(A \cup B^1, E_M)$ from $A(Y)$ to $A(X)$. Let $F(Y, X)$ be those paths. We will later see that each such path alternates between heavy edges and their blocking edges. Unlike the quasi-polynomial-time algorithm, in our polynomial-time algorithm, the heavy edges do not appear in the alternating tree. Instead, they are used in the flow network $G(A \cup B^1, E_M)$ to play a role of connecting existing addable light edges and blocking light edges.

4.2 Building Phase

▶ **Definition 10** (Layers). For all $i \geq 1$, a layer L_i is a tuple (X_i, Y_i), where X_i is a set of addable edges and Y_i is a set of blocking edges that block edges in X_i.

Initialize $l = 0$, $L_0 = (\emptyset, \{(i_0, \emptyset)\})$. We call an addable edge $e = \{i\} \cup P$ **unblocked** if it contains at least r unblocked light items: $|P \backslash (\bigcup_{e' \in \mathsf{blocking}(e)} B^\epsilon(e'))| \geq r$. Initialize the set of unblocked addable edges be $I = \emptyset$. Throughout the whole algorithm, we maintain a set I of unblocked addable edges and layers $L_i(X_i, Y_i)$ for all $i \leq l$, where X_i contains blocked addable edges. Initialize $X_{l+1} = Y_{l+1} = \emptyset$. We build a new layer as follows.

▶ **Definition 11** (Addable). Given layers $X_{\leq l+1}$ and $Y_{\leq l}$, an edge $e = \{i\} \cup P$ is addable if $|P| = p$ and $P \subseteq B^\epsilon_i \backslash B^\epsilon(X_{\leq l+1} \cup Y_{\leq l})$ such that $f(Y_{\leq l}, X_{\leq l+1} \cup I + e) > f(Y_{\leq l}, X_{\leq l+1} \cup I)$.

Note that such an edge is connected to a blocking edge in $Y_{\leq l}$ by a path in $G(A \cup B^1, E_M)$ that is disjoint from other paths connecting existing blocking edges and addable edges.

Given an addable edge: if it is unblocked, then include it in I; otherwise include it in X_{l+1}. When there is no more addable edges, let $Y_{l+1} = \mathsf{blocking}(X_{l+1}) = \bigcup_{e \in X_{l+1}} \mathsf{blocking}(e)$, set $l = l + 1$ and try to contract L_l. Note that a blocking edge can block multiple addable edges.

4.3 Collapse Phase

Let $W = F(Y_{\leq l}, I)$ be constructed as follows. Initialize $W = \emptyset = F(Y_{\leq 0}, I)$. Recursively for $i = 1, 2, \ldots, l$, let $W = F(Y_{\leq i}, I)$ be augmented from $W = F(Y_{\leq i-1}, I)$. In the final W, let $W_i \subseteq W$ be the paths from $A(Y_i)$ to $A(I)$ and let $I_i \subseteq I$ be those reached by W_i. By the above construction, if $f \in Y_{\leq i}$ have no out-flow in $F(Y_{\leq i}, I)$, then it will not have out-flow in $F(Y_{\leq j}, I)$, for any $j > i$. Hence we have for all $i = 1, 2, \ldots, l$, $|W_i| = |I_i|$ and $|W_{\leq i}| = |I_{\leq i}| = f(Y_{\leq i}, I) = f(Y_{\leq i}, I_{\leq i})$. Note that every path in W_i starts with an agent

$u \in A(Y_i)$ that is assigned a light edge by M and ends at a agent $v \in A(I_i)$ with an unblocked addable edge, which provides a possibility of swapping out a blocking edge in the tree with an unblocked addable edge (by reassigning all heavy items in the path).

▶ **Definition 12** (Collapsible). We call layer L_i collapsible if $|I_i| \geq \mu|Y_i|$.

Intuitively, $|I_i| \geq \mu|Y_i|$ implies that we can swap out a μ fraction of blocking edges in Y_i (which is called a collapse). Let L_t be the earliest collapsible layer, we collapse it as follows.

Step-(1). For each path $P(u, v)$ in W_t from $e_1 := \{u\} \cup R \in Y_t$ to $e_2 := \{v\} \cup P \in I_t$:

1. $M = M - e_1 + e'$, swap out blocking edge e_1 with $e' := \{v\} \cup P'$, where $|P'| = r$ and $P' \subseteq P \backslash \bigcup_{e \in \mathsf{blocking}(e_2)} B^{\epsilon}(e)$,
2. reverse all heavy edges in $P(u, v)$: $M = M \cup \{\{i, j\} : (i, j) \in P(u, v) \cap (A \times B)\} \backslash \{\{i', j'\} : (j', i') \in P(u, v) \cap (B \times A)\}$.

Note that after the above operations, only Y_t and M are changed: the size $|Y_t|$ is reduced by a factor of at least μ and the number of heavy edges in M is not changed.

Step-(2). Set $I = I_{\leq t-1}$. Note that $|W_{\leq t-1}| = f(Y_{\leq t-1}, I) = f(Y_{\leq t-1}, I_{\leq t-1})$ still holds.

Step-(3). Set $l = t$ and repeat the collapse if possible. Remove all unblocked edges in X_t (since $|Y_t|$ decreases). For each removed unblocked edge e, include it in I if $f(Y_{\leq t-1}, X_{\leq t} \cup I + e) > f(Y_{\leq t-1}, X_{\leq t} \cup I)$.

4.4 Invariants and Properties

▶ **Fact 13** (Key Invariant). *Since the construction of L_t (until L_{t-1} is collapsed), $f(Y_{\leq t-1}, X_{\leq t} \cup I)$ does not decrease and is always no less than $|X_{\leq t}|$.*

Proof. We prove by induction on $t \geq 1$. Consider the base case when $t = 1$. The statement trivially holds when L_t is just constructed and when $|X_t \cup I|$ increases. Suppose in some iteration $|X_t \cup I|$ decreases, then it must be because Y_t is collapsed, in which case $f(Y_{\leq t-1}, X_{\leq t} \cup I)$ does not change due to the update rule of step-(3).

Now assume the statement is true for t and consider $t + 1$.

When L_{t+1} is built we have $f(Y_{\leq t}, X_{\leq t+1} \cup I) \geq f(Y_{\leq t-1}, X_{\leq t} \cup I \cup X_{t+1}) = f(Y_{\leq t-1}, X_{\leq t} \cup I) + |X_{t+1}| \geq |X_{\leq t+1}|$. Since $|X_i|$ does not increase afterwards for all $i \leq t + 1$, applying the same argument to L_{t+1} as above yields the fact. ◀

▶ **Lemma 14** (Exponential Growth). *Let $r = \max\{\lceil \frac{k}{9} \rceil, \lceil \frac{k-10}{3+2\sqrt{2}} \rceil\}$, if $T \leq \mathsf{OPT}$, then for all $i \in [l]$ we have $|Y_i| \geq \mu^2 |Y_{\leq i-1}|$, which implies $l = O(\frac{1}{\mu^2} \log n)$.*

Proof Sketch. suppose $|Y_t| < \mu^2 |Y_{\leq t-1}|$, then by Fact 13 we can show that $|X_{\leq t}| \leq f(Y_{\leq t-1}, X_{\leq t} \cup I) < (\frac{r}{p-r+1} + 2\mu)|Y_{\leq t-1}|$ (when μ is sufficiently small). Hence at the moment when there is no more addable edge that can be included into $X_{\leq t}$, we can show that the total number of items agents reachable from $A(Y_{\leq t-1})$ are interested in are not enough to achieve $T \leq \mathsf{OPT}$. Please refer to our full version for the complete proof. ◀

Proof of Theorem 6. For any T and $k = \lceil \frac{T}{\epsilon} \rceil$, the algorithm tries to compute an $r\epsilon$-allocation, for integer r as large as possible, by enumerating all possible values of p between r and k. For any fixed r and p, we try to augment the partial matching M that matches each agent with either a heavy item or r light items. Hence it suffices to show that the algorithm terminates in polynomial time for augmenting the size of M by one. Since each iteration can be done in polynomial time, it suffices to bound the number of iterations by $\mathsf{poly}(n)$. The approximation ratio will be the maximum of $\frac{k}{r}$, over all $T \le \mathsf{OPT}$.

By Lemma 14 and the definition of collapsible, we know that after each iteration, either (if no collapse) a new layer with $|Y_{l+1}| \ge \mu^2 |Y_{\le l}|$ is constructed, or some $|Y_t|$ is reduced to at most $(1-\mu)|Y_t|$ while Y_i are unchanged, for all $i < t$. Let $s_i = \lfloor \log_{\frac{1}{1-\mu}} \frac{|Y_i|}{\mu^{2i}} \rfloor$ and $\mathbf{s} = (s_1, s_2, \ldots, s_l, \infty)$ be the signature, then we have: (1) it is lexicographically decreasing across all iterations: if there is no collapse, then some layer is newly constructed and hence $\mathbf{s}$ decreases; otherwise let L_t be the last layer that is collapsed and $|Y_t|$ be the size of Y_t before it is collapse: we know that at the end of the iteration s_i is not changed for all $i < t$ while $s_t \le \lfloor \log_{\frac{1}{1-\mu}} \frac{(1-\mu)|Y_i|}{\mu^{2i}} \rfloor = \lfloor \log_{\frac{1}{1-\mu}} \frac{|Y_i|}{\mu^{2i}} \rfloor - 1$ is decreased by at least one, which also means $\mathbf{s}$ decreases; (2) its coordinates are not decreasing: for all $i \in [l-1]$ we have $s_{i+1} = \lfloor \log_{\frac{1}{1-\mu}} \frac{|Y_{i+1}|}{\mu^{2i+2}} \rfloor \ge \lfloor \log_{\frac{1}{1-\mu}} \frac{|Y_{\le i}|}{\mu^{2i}} \rfloor \ge s_i$. Since we have $l = O(\log n)$ and $s_i = O(\log n)$ for all $i \in [l]$, the total number of iterations (signatures) is at most $2^{O(\log n)} = \mathsf{poly}(n)$.

Approximation ratio: When $k \le 9$, then a trivial 9-approximation can be achieved by a ϵ-allocation (maximum matching). By the proof of Lemma 14, the approximation ratio $\frac{k}{r}$ is always at most 9 and tends to $3 + 2\sqrt{2} \approx 5.83$ as $\epsilon \to 0$. ◀

References

1 Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. In Piotr Indyk, editor, *SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1357–1372. SIAM, 2015. doi:10.1137/1.9781611973730.90.

2 Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8(3):24, 2012. doi:10.1145/2229163.2229168.

3 Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In David S. Johnson and Uriel Feige, editors, *STOC 2007, San Diego, California, USA, June 11-13, 2007*, pages 114–121. ACM, 2007. doi:10.1145/1250790.1250808.

4 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In Jon M. Kleinberg, editor, *STOC 2006, Seattle, WA, USA, May 21-23, 2006*, pages 31–40. ACM, 2006. doi:10.1145/1132516.1132522.

5 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005. doi:10.1145/1120680.1120683.

6 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 107–116. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.51.

7 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On (1, epsilon)-restricted assignment makespan minimization. In Piotr Indyk, editor, *SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1087–1101. SIAM, 2015. doi:10.1137/1.9781611973730.73.

8 Uriel Feige. On allocations that maximize fairness. In Shang-Hua Teng, editor, *SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 287–293. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347114.

9 Michel X. Goemans, Nicholas J. A. Harvey, Satoru Iwata, and Vahab S. Mirrokni. Approximating submodular functions everywhere. In Claire Mathieu, editor, *SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 535–544. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496829.

10 Daniel Golovin. Max-min fair allocation of indivisible goods. Technical report, Carnegie Mellon University, 2005.

11 Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovasz local lemma. In *FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 397–406. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.45.

12 Penny E. Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11(3):245–248, 1995. doi:10.1007/BF01793010.

13 Subhash Khot and Ashok Kumar Ponnuswami. Approximation algorithms for the max-min allocation problem. In *APPROX-RANDOM*, volume 4627 of *Lecture Notes in Computer Science*, pages 204–217. Springer, 2007.

14 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. doi:10.1007/BF01585745.

15 Lukas Polacek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 726–737. Springer, 2012.

16 Ola Svensson. Santa claus schedules jobs on unrelated machines. In Lance Fortnow and Salil P. Vadhan, editors, *STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 617–626. ACM, 2011. doi:10.1145/1993636.1993718.

All-Pairs Shortest Paths in Unit-Disk Graphs in Slightly Subquadratic Time

Timothy M. Chan[1] and Dimitrios Skrepetos[2]

1 Cheriton School of Computer Science, University of Waterloo, Canada
tmchan@uwaterloo.ca

2 Cheriton School of Computer Science, University of Waterloo, Canada
dskrepet@uwaterloo.ca

Abstract

In this paper we study the all-pairs shortest paths problem in (unweighted) unit-disk graphs. The previous best solution for this problem required $O\left(n^2 \log n\right)$ time, by running the $O\left(n \log n\right)$-time single-source shortest path algorithm of Cabello and Jejčič (2015) from every source vertex, where n is the number of vertices. We not only manage to eliminate the logarithmic factor, but also obtain the first (slightly) subquadratic algorithm for the problem, running in $O\left(n^2 \sqrt{\frac{\log \log n}{\log n}}\right)$ time. Our algorithm computes an implicit representation of all the shortest paths, and, in the same amount of time, can also compute the diameter of the graph.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases unit-disk graphs, all-pairs shortest paths, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.24

1 Introduction

The all-pairs shortest paths (APSP) problem is one of the most known problems in the field of algorithms: given a graph $G = (V, E)$ with n vertices, where each edge has a real weight, we are asked to compute the shortest paths between all pairs of vertices. The classical algorithm of Floyd and Warshall solves the problem in $O\left(n^3\right)$ time. The first improvement for general weighted dense graphs came in 1976 when Fredman [16] obtained an algorithm of $O\left(n^3 \left(\frac{\log \log n}{\log n}\right)^{1/3}\right)$ time. This advancement sparked a number of algorithms that save polylogarithmic factors. For more results for APSP in graphs with arbitrary edge weights see [13, 32, 21, 34, 40, 35, 4, 22, 5, 23]. Recently, Williams [36] in a major breakthrough provided an algorithm that achieves superpolylogarithmic speedup, requiring $\frac{n^3}{2^{\Omega\left(\sqrt{\log n}\right)}}$ randomized time, later derandomized by Chan and Williams [10]. For the case of unweighted undirected graphs, Seidel [30] and Galil and Margalit [18] devised algorithms that solve the problem in matrix multiplication time. For results on directed and unweighted graphs and on graphs whose weights are small integers, see [17, 39, 18, 31, 33].

One very important class of graphs arising from computational geometry is *unit-disk graphs*. A unit-disk graph is the intersection graph of a set of unit disks, which is defined by creating a vertex for each unit disk and an edge between any two unit disks that intersect each other. Equivalently, given a set S of n points in the plane (the disk centers, after rescaling by a half), the unit-disk graph, $\mathrm{UD}(G)$, is defined by setting its vertex set to be S and creating an edge between any two points of S whose Euclidean distance is at most one. The edges are unweighted. The unit-disk graph with n vertices may contain $\Theta(n^2)$ edges, as every unit disk may intersect with every other unit disk. Since we aim for a subquadratic

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 24; pp. 24:1–24:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

solution to the all-pairs shortest path problem, we do not construct the set of the edges explicitly in our algorithms.

Unit-disk graphs are among the most studied families of graphs in geometry, with motivation from communication networks. Unit-disk graphs are related to planar graphs: by the circle packing theorem of Koebe–Andreev–Thurston any planar graph can be represented as a coin disk graph, although the disks may have different radii; on the other hand, planar graphs do not have large cliques unlike unit-disk graphs. Frederickson [15] gave an $O\left(n^2\right)$-time algorithm for solving the APSP problem in weighted planar graphs. Chan [6] improved the bound for unweighted directed planar graphs with an $O\left(n^2\frac{\log\log n}{\log n}\right)$-time algorithm (and also considered general unweighted undirected sparse graphs). Wulff-Nilsen [37] independently developed another $O\left(n^2\frac{\log\log n}{\log n}\right)$-time algorithm for computing the diameter of unweighted undirected planar graphs (and also announced similar results for the weighted case).

In this paper, we provide an algorithm for the APSP problem in unit-disk graphs that requires $O\left(n^2\sqrt{\frac{\log\log n}{\log n}}\right)$ time. The previously fastest solution was to run from each vertex the single-source shortest path algorithm of Cabello and Jejčič [3], which required $O\left(n^2\log n\right)$ total time. (See [29, 19] for other previous results on shortest paths in unit-disk graphs.) Therefore, we not only shave off the extra logarithmic factor of the previous result, but also provide the first (slightly) subquadratic solution to the problem. Our algorithm computes an implicit representation of the shortest paths: we encode the $O\left(n^2\right)$ shortest path distances and predecessors using bit-packing techniques (for use of bit-packing techniques in shortest path algorithms see [5]), so that we are still able to retrieve the shortest path distance of a pair of vertices in $O(1)$ time and the shortest path π itself in time linear in the number of vertices of π. In the same amount of time, we can also compute the diameter of the graph.

In recent years, obtaining polylogarithmic speedup for standard algorithmic problems have received considerable renewed attention. Such problems include 3SUM [20], Fréchet distance [1, 26], combinatorial Boolean matrix multiplication [8, 38], Klee's measure problem [7], CFL reachability [11], and many more. Our result can be seen as another contribution along this line of research.

The polylogarithmic improvement that we obtain for APSP in unit-disk graphs goes beyond standard word RAM tricks. First, in Section 2, we present a new algorithm for the *single-source* shortest path problem for unit-disk graphs, running in linear time after presorting the x- and y-coordinates of the input points. This improves over Cabello and Jejčič's single-source algorithm [3]. Their algorithm started with the Delaunay triangulation and performed repeated nearest neighbor queries, which inherently required $\Omega(n\log n)$ time even excluding preprocessing cost. Our new algorithm is instead based on a simple grid approach and exploits a linear-time Graham-scan-like procedure [28] for computing upper envelopes of presorted unit disks. (According to [3], Efrat has also observed an alternative, grid-based $O(n\log n)$-time algorithm, but his suggested solution seemed a bit more complicated and used a semi-dynamic data structure of Efrat et al. [25], which also inherently required $\Omega(n\log n)$ time even after presorting.)

Second, in Section 3, we extend the single-source algorithm to the case of *multiple* $\left(k = O\left(\sqrt{\frac{\log n}{\log\log n}}\right)\right)$ sources that lie in a *cluster*, i.e., in a common grid cell. In this case, we have to construct not just one but k upper envelopes, one for each source. This leads to a new kind of data structure problem: preprocess a set of unit disks (a "universe") so that given any subset of the universe, we can compute the upper envelope of the subset in slightly *sublinear* time. Our ideas can similarly be applied to the following (even more natural) problem, of independent interest: preprocess a point set (a universe) in 2D so that given any subset of the universe, we can compute its convex hull, again in slightly sublinear time. Note that the input subset and the output can be encoded with linear number of bits, and thus slightly sublinear number of words. Solving problems for "preprocessed universes" is a relatively recent research direction (for example, see [9, 14, 12, 2]); our result with slightly sublinear time provides an unusual addition to this body of work.

Finally, in Section 4, to obtain our subquadratic-time APSP algorithm, we draw inspiration from previous algorithms for planar graphs [6, 37], which use planar separators to decompose the graph into regions of polylogarithmic size, and table-lookup techniques to handle each region. However, this approach does not directly apply to unit-disk graphs, because there could be large cliques and no small separators. On the other hand, when there are many large cliques, intuitively we should be able to exploit the multi-source algorithm from Section 3 to handle such clusters more efficiently. The challenge lies in how to carefully combine these two approaches. For unit-disk graphs, we end up avoiding planar separators and instead adopting a simpler *shifted grid* strategy [24]. This strategy is standard for geometric approximation algorithms, but we use the technique in a new and interesting way to design an *exact* algorithm (with an intricate balancing of parameters).

All of our algorithms operate in a standard unit-cost RAM model of computation, where each word can store either an input point or a $(\log n)$-bit integer. (We do *not* need to assume integer input coordinates from a bounded universe: the input coordinate values may be real numbers if we assume the availability of a few constant-degree algebraic predicates, as in most real-RAM algorithms in computational geometry. We are not cheating when using bit-packing and table-lookup tricks if they are done with $(\log n)$-bit words and without exotic word operations.)

2 Single-source shortest paths in linear time (after presorting)

In this section, we begin with the single-source problem: given a set S of n points, and a source point $s \in S$, we want to compute the shortest paths from s to all points in S in the unit-disk graph $\mathrm{UD}(G)$. We assume that the points in S have been presorted with respect to both x and y. The approach that we follow is a variant of the classical breadth-first search (BFS) algorithm.

The first step is to build a grid composed of square cells with side length $\frac{1}{\sqrt{2}}$. A cell c' is a *neighbor* of a cell c if the minimum distance between c and c' is at most 1. Clearly, the number of neighbors of a cell is constant. For each point p, as is the case in the classical BFS, we have a value for its shortest-path distance from s, $\mathit{dist}[p]$, and another for the predecessor, $\mathit{pred}[p]$, denoting the point before p in that path.

The algorithm iteratively performs the following step $n-1$ times. In each step ℓ we assume that we have already computed the distances and predecessors for all of the points that are at distance no more than $\ell-1$ from s and that we have a list, called the *frontier*, containing all of the points being at distance exactly $\ell-1$ from s. We find all points whose shortest-path distance from s is ℓ by finding all points at distance at most 1 from the points in the frontier, and we filter out the ones whose shortest-path distance from s has been found in earlier steps. For the remaining points, we update their distances and predecessors properly. Finally, we replace the frontier with the set of these points. It is easy to establish the correctness of the algorithm by induction. The concepts of the grid, cells, neighboring cells, and frontier points are depicted in Figure 1.

We say that a cell is *visited* at step ℓ if the algorithm performs an operation to any one of its points during that step. When the algorithm visits a cell, each point of that cell is used in operations at most twice. The following lemma is crucial for the rest of the section.

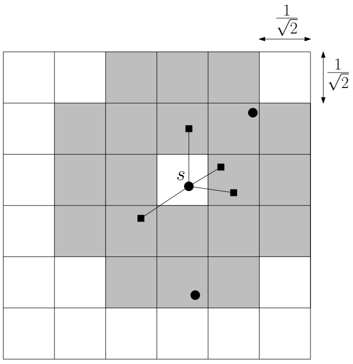


Figure 1 Here we see the grid created for six points depicted as black disks and squares. Each of the thirty-six squares corresponds to a cell. The neighbors of the cell of the third row and fourth column, containing the point s, are shaded gray. In the second step of the algorithm, where s is the source, we see the four points that are at distance one from s, depicted as squares, and the shortest path tree of s so far, depicted with line segments. Those four points compose the frontier for the second step.

▶ **Lemma 1.** *Each cell c is visited only a constant number of times.*

Proof. A cell c is visited either (i) when at least one of its points enters the frontier or (ii) when at least one of the points of a neighboring cell c' of c enters the frontier.

In case (i), we note that once at least one point of c enters the frontier in a step of the algorithm, then the rest of the points of c enter the frontier either in this step or in the next because any two points of c are at distance at most one. Also, a point can be in the frontier in exactly one step. Therefore, c has points in the frontier in at most two steps of the algorithm.

In case (ii), c is visited when one of its neighbors, c', has at least one frontier point. Since the number of the neighbors of c is constant and, by case one, any cell has frontier points at most twice, we conclude that a cell c is visited by its neighbors in a constant number of steps of the algorithm. ◀

To find points that are at distance at most 1 from the points in the frontier, we solve the following subproblem.

▶ **Subproblem 2.** *Given a set of n_r red points below a horizontal line h and another set of n_b blue points above h, both presorted by x, determine for each blue point whether there is a red point at distance at most one from it.*

▶ **Lemma 3.** *We can solve Subproblem 2 in $O(n_r + n_b)$ time.*

Proof. We consider the upper *envelope* of the unit disks centered at the red points; it is composed of the part of the boundary of the union of those unit disks that lies above h.

Notice that the parts of the boundaries of unit disks above h form a *pseudoline* arrangement. For each unit disk we can extend that part to a x-monotone curve, a pseudoline, from $x = -\infty$ to $x = \infty$ such that it intersects only once with the pseudoline of another unit disk. Moreover, we can make the order of the pseudolines at $x = -\infty$ coincide with the order of x-coordinates of the disk centers. An example of the pseudolines can be see in Figure 2. Then we can compute the upper envelope of these pseudolines by adapting the standard *Graham scan* algorithm [28] for computing upper envelopes of lines (i.e., convex hulls of points in the plane by duality). Graham scan takes linear, $O(n_r)$, time after presorting. Notice that we do not

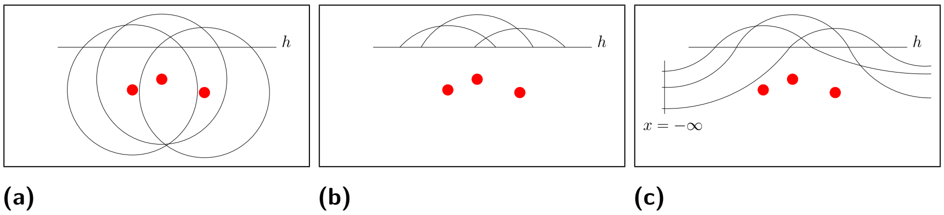


Figure 2 In the first subfigure we see the unit disks of three red points, depicted as disks, and a horizontal line h. In the second subfigure, we see the part of the boundaries of those unit disks lying above h. In the third subfigure, we see the pseudolines corresponding to each unit disk as defined in Lemma 3. Notice that the y-order of the pseudolines at $x = -\infty$ coincides with the x-order of the centers of their unit disks.

have to explicitly compute the pseudolines; the notion of the pseudolines is used only for intuition. The algorithm, assuming that the points are presorted, scans them from left to right, finds intersections between unit disks, and builds step-by-step the upper envelope.

Finally, to solve Subproblem 2, we want to determine for each blue point whether it is below the upper envelope of the red disks. This can done by performing a linear scan over the vertices of the upper envelope and the (presorted) blue points in $O(n_r + n_b)$ time. ◀

Putting everything together, we obtain the following theorem.

▶ **Theorem 4.** *Given a unit-disk graph, we can compute the shortest path tree from a given source in $O(n)$ time, if the input points have been pre-sorted by both x and y.*

Proof. In step ℓ of the BFS algorithm, we find the points at distance exactly ℓ from the source as follows. For each cell c having at least one point in the frontier and each neighbor c' of c, we use the subroutine from Lemma 3 on the input where the red points are the points of c that are in the frontier and the blue points are all the points of c'.

The cells c and c' are either horizontally or vertically separated; without loss of generality, we can assume the former by rotation. By Lemmas 1 and 3, the total time of the BFS is $O(n)$.

Note that to identify the predecessors during the BFS, we need to strengthen Subproblem 2 to report for each blue point a *witness* red point (if exists) that is at distance at most 1; the method in Lemma 1 can easily provide such witnesses. Initially assigning points to grid cells can be done in linear time (without the need for hashing, because of presortedness). ◀

Applying the above theorem n times immediately yields an $O(n^2)$-time algorithm for APSP for unit-disk graphs. In the subsequent sections, we aim for a slightly subquadratic algorithm.

3 Shortest paths for multiple sources in a cluster

In this section, we extend the single-source algorithm of the previous section to compute shortest paths from k source points $s_1, \ldots, s_k \in S$ to all vertices in S, where $k = O\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ and $s_1, \ldots, s_k$ lie in a common grid cell. The approach is to run BFS from the multiple sources simultaneously but avoid a factor-k slowdown by using bit-packing tricks – this approach is inspired by the first APSP algorithm in [6].

First, we extend Lemma 1 to the case of k source points, lying close to each other. We say that a cell is *visited* at step ℓ if at the ℓ-th level of the BFS for at least one of the sources, the algorithm performs an operation to any one of its points.

▶ **Lemma 5.** *If the k source points are in the same cell, each cell is visited only a constant number of times.*

Proof. Let $dist_s[p]$ be the shortest-path distance between a source s and a point p. Observe that for any two sources s and s' in a common cell, $dist_s[p]$ and $dist_{s'}[p]$ can differ by at most one by the triangle inequality. Therefore, the first time that any point of a cell enters the frontier of any source, by the next step of the algorithm that point will enter the frontier of the rest of the sources. The rest of the proof is as in Lemma 1. ◀

If we apply the algorithm of Lemma 3 separately for the k sources, then the cost of this operation alone would be $O(nk)$, which we cannot afford. To overcome the issue, we introduce an extension of Subproblem 2.

▶ **Subproblem 6.** *Preprocess a universe R of n_r red points below a horizontal line h and another universe B of n_b blue points above h, so that given any subset $Q \subseteq R$ of the red universe, we can determine for each blue point whether there is a red point of Q at distance at most one from it, in* sublinear *total time.*

As we have seen in the proof of Lemma 3, the key to solving the subproblem lies in the construction of upper envelopes, so we concentrate on solving the following subproblem.

▶ **Subproblem 7.** *Preprocess a universe R of n_r red points below a horizontal line h, so that given any subset $Q \subseteq R$ of the red universe, we can compute the upper envelope of the unit disks centered at Q (specifically the part above h) in* sublinear *time.*

By sublinear we mean $O\left(\frac{n_r}{\operatorname{polylog} n}\right)$ or $O\left(\frac{n_r+n_b}{\operatorname{polylog} n}\right)$. The reason that sublinear time is feasible at all is that we can represent (i) the input subset Q as an n_r-bit vector, where the i-th bit denotes whether the i-th red point in x-order is in Q, and (ii) the output upper envelope is another n_r-bit vector, where the i-th bit denotes whether the disk defined by the i-th point participates in the upper envelope. We can pack either vector into $O\left(\frac{n_r}{\log n}\right)$ words.

▶ **Lemma 8.** *We can solve Subproblem 7 with $O(n_r \log n_r + n_r 2^g)$ preprocessing time and $O\left(n_r \frac{\log g}{g}\right)$ query time, for any given $g \leq \log n$.*

Proof. During the preprocessing phase, we divide the x-ordered sequence of red points into $O\left(\frac{n_r}{g}\right)$ chunks of at most g consecutive points each; the chunks lie in $O\left(\frac{n_r}{g}\right)$ disjoint slabs. The red points of each chunk compose $O(2^g)$ possible subsets, and we precompute the upper envelope for each such subset (specifically the part above h) in $O(g)$ time. The computation of all these envelopes, which we call *small upper envelopes*, needs $O\left(\frac{n_r}{g} 2^g g\right) = O(n_r 2^g)$ time. We store a lookup table containing all the small upper envelopes, each represented as an ordered array. In Figure 3, we see the chunks and the small upper envelope of each.

To answer a query for a given subset of the red points, we have to merge $O\left(\frac{n_r}{g}\right)$ small upper envelopes for the relevant subsets of the $O\left(\frac{n_r}{g}\right)$ chunks. Observe that the $O\left(\frac{n_r}{g}\right)$ small upper envelopes themselves can be viewed as a pseudoline arrangement (since the chunks are vertically separated). We can therefore apply Graham scan to compute the upper envelope of the $O\left(\frac{n_r}{g}\right)$ small upper envelopes, using a linear $O\left(\frac{n_r}{g}\right)$ number of *primitive operations.* We need two primitive operations: (i) finding the intersection point between two small upper envelopes, and (ii) determining whether a given point is above or below a small upper envelope. Both operations can be done in $O(\log g)$ time by binary searching (see [27]

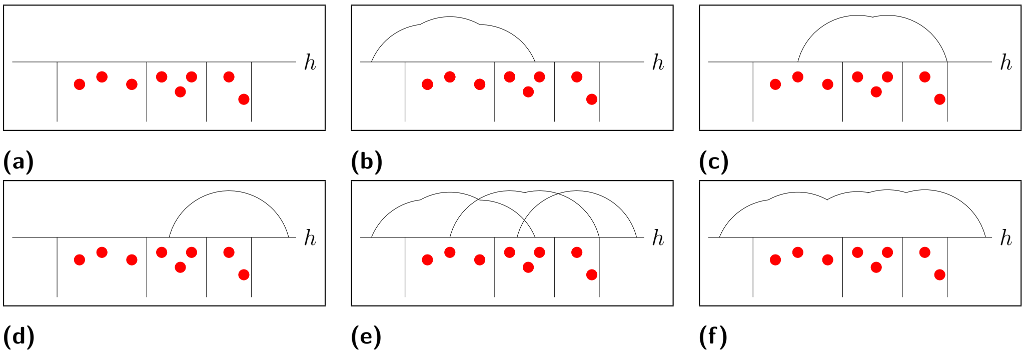


Figure 3 In the first subfigure we see the chunks of eight red points, drawn as disks, for $g = 3$, and a horizontal line h. Notice that the last chunk has only two points. In the next three subfigures, we see the small upper envelope associated with each chunk. The three small envelopes are seen together in the fifth subfigure. Finally, in the sixth subfigure we see the upper envelope of the small upper envelopes, which is the upper envelopes of all red points.

for (i)). Thus, Graham scan takes $O\left(\frac{n_r}{g}\log g\right)$ time. The output is a sequence of $O\left(\frac{n_r}{g}\right)$ pieces of small upper envelopes; we can convert each piece into the bit-vector format, in additional $O\left(\frac{n_r}{\log n}\right)$ total time. In Figure 3, we see the upper envelope. ◀

▶ **Lemma 9.** *We can solve Subproblem 6 with $O\left(n_r \log n_r + n_b \log n_b + n_r 2^g g + n_b g\right)$ preprocessing time and $O\left((n_r + n_b)\frac{\log g}{g}\right)$ query time, for any given $g \leq \log n$.*

Proof. We build on the method from Lemma 8. During the preprocessing phase, we also divide the x-ordered sequence of blue points into $O\left(\frac{n_b}{g}\right)$ chunks of at most g consecutive blue points each; the chunks lie in $O\left(\frac{n_b}{g}\right)$ disjoint slabs (regions each bounded by two vertical lines). We store the following extra structures:

1. For each small upper envelope e and each slab σ that contains at least one vertex of e, we precompute a g-bit vector where the i-th bit denotes whether the i-th blue point in σ is below e. We store all these vectors in a lookup table. There are $O\left(\frac{n_r}{g}2^g\right)$ small upper envelopes, each with $O(g)$ vertices; the total time is $O\left(n_r 2^g g\right)$.
2. For each slab σ, we precompute the arrangement of the $O(g)$ unit disks centered at the blue points in σ; the arrangement has $O\left(g^2\right)$ complexity and we can build a *point location* structure [28] in $O\left(g^2\right)$ time. For each face of the arrangement, we record a g-bit vector where the i-th bit denotes whether the face is inside the disk for the i-th blue point. There are $O\left(\frac{n_b}{g}\right)$ slabs; the total time is $O(n_b g)$.

To answer a query, we first construct the upper envelope E of the $O\left(\frac{n_r}{g}\right)$ small upper envelopes as described in the proof of Lemma 8. We then need to determine for each blue point whether it is below E.

We scan the slabs from left to right. Consider the next slab σ. Consider each small upper envelope e that contributes arcs to E inside σ. We compute a bit vector $\mathbf{v}_{\sigma,e}$ where the i-th bit denotes whether the i-th blue point in σ is below e, as follows:

1. If σ contains at least one vertex of e, then $\mathbf{v}_{\sigma,e}$ has already been precomputed in the lookup table.

2. If σ contains no vertices of e, then only one disk contributes to e inside σ; say the disk is defined by the red point q. We can determine $\mathbf{v}_{\sigma,e}$ by looking up the face containing q in the arrangement of the blue disks for σ, in $O(\log g)$ time by point location.

Finally, we take the bitwise-or of the bit vectors $\mathbf{v}_{\sigma,e}$ over all small upper envelopes e that contributes to E inside σ. The total number of bitwise-or operations and point location queries is $O\left(\frac{n_r+n_b}{g}\right)$, yielding total query time $O\left(\frac{n_r+n_b}{g}\log g\right)$. ◀

▶ **Theorem 10.** *Given a unit-disk graph, after $O\left(n\log n + n2^{O(k\log k)}\right)$-time preprocessing, we can compute an implicit representation of the shortest path trees for any $k \leq \sqrt{\frac{\log n}{\log\log n}}$ source points lying in a unit-diameter square, in $O(n)$ time.*

Proof. For each point p, we maintain k-bit vectors *frontier*$[p]$ (resp. *found*$[p]$), where the i-th bit denotes whether p is in the frontier of the i-th source (resp. whether p has previously appeared in the frontier) in each step ℓ of the BFS algorithm.

In step ℓ of the BFS, we proceed as follows. For each cell c having at least one point in one of the k frontiers and for each neighbor c' of c, we use the subroutine from Lemma 9 on the input where the red universe contains all n_r points of c and the blue points are all n_b points of c'; for each of the k sources, we query with the red subset containing all points in its frontier in c. The total time for the k queries is $O\left(k(n_r+n_b)\frac{\log g}{g}\right) = O(n_r+n_b)$ by setting $g = k\log k$.

One technical issue concerns the generation of the bit-vector representation for these input red subsets (which are ordered by x or y depending on whether c and c' are horizontally or vertically separated). This can be done by taking each chunk of g red points $p_1,\ldots,p_g$, collecting the g k-bit vectors *frontier*$[p_1],\ldots,$*frontier*$[p_g]$, and performing a *transposition* to generate k g-bit vectors, where the j-th bit of the i-th vector is equal to the i-th bit of *frontier*$[p_j]$. The transposition involves simply shuffling bits between words and can be straightforwardly implemented in $O(1)$ time by table lookup if $kg \leq \log n$. The outputs can similarly be converted by transposition to obtain vectors *out*$[p]$ for the blue points p in c', where the i-th bit denotes whether p is at distance at most one from some red point in c with respect to the i-th source.

We can update the k-bit vector *found*$[p]$ by taking the bitwise-or with *out*$[p]$. At the end of step ℓ, we can update the k-bit vector *frontier*$[p]$ by taking the bitwise-difference between the new and old *found*$[p]$ vectors. By Lemmas 5 and 9, the total time of the k simultaneous BFSs is $O(n)$.

Two remaining technical issues concern the encoding of the shortest path distances and of the predecessors. To address the former issue, for each point p, we store $dist_{s_1}[p]$ and work with the vector containing $dist_{s_i}[p] - dist_{s_1}[p]$ over all k sources s_i. Recall that these values are in $\{-1,0,1\}$ by the triangle inequality, and so the vector can be encoded in $O(k)$ bits; the total space over all n points is $O\left(n + \frac{nk}{\log n}\right) = O(n)$ words, and it is easy to update the distances of a point in constant time.

To address the latter issue, we need to strengthen Subproblem 6 so that given any blue point, we can report a witness red point (if exists) that is at distance at most one in constant time. Thus we need to perform a few modifications to the proof of Lemma 9. First in the lookup tables during the preprocessing phase, we record witnesses for the true bits for each g-bit vector, stored in a $(g\log g)$-bit *witness vector.* In the query algorithm, consider each slab σ. The upper envelope E inside σ consists of pieces of small upper envelopes. Divide σ into subslabs by drawing vertical lines at the endpoints of these pieces. If, for each subslab, we created a pointer from each blue point to the small upper envelope in the subslab, we

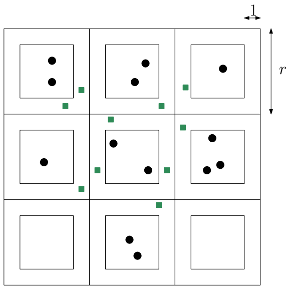


Figure 4 In this figure we see the shifted grid. We assume that the points have already been shifted. The nine big squares correspond to the supercells. The smaller square within each supercell corresponds to the region of the supercell that contains points at distance more than one from its boundary. The boundary points are drawn as green squares and the non-boundary points as black disks.

would need $O(kn)$ total time. To avoid that, for each subslab, we mark its rightmost blue point, which can be found by binary search in $O(\log g)$ time; the total time for that is $O\left(\frac{n_r+n_b}{g} k \log g\right) = O\left(n_r + n_b\right)$. Then we create a pointer from this marked point to the small upper envelope of its subslab. Finally, for each slab, we create a pointer from each blue point q to its successor among the marked blue points; these g pointers require $O\left(g \log g\right)$ bits, can be created in constant time, and can be stored in a $(g \log g)$-bit *pointer vector*, so the total time for this step is $O\left(\frac{n_b}{g} k\right) = O\left(\frac{n_b}{\log k}\right)$. Then, given any blue point q in σ, we can retrieve the pointer vector of σ, look up the marked successor of q, follow its pointer to the small upper envelope e in σ, and then look up q's witness with respect to $\mathbf{v}_{\sigma,e}$, all in constant time. ◀

4 All-pairs shortest paths in slightly subquadratic time

In this final section, we present our APSP algorithm for unit-disk graphs. Let r, a, b be parameters to be chosen later. We build a grid composed of square cells with side length r, where r is a parameter to be specified later $\left(\text{larger than } \frac{1}{\sqrt{2}}\right)$. Call these larger grid cells *supercells*. We say that a point $p \in S$ is a *boundary point* if it is at distance at most one from the boundary of some supercell. We begin with a standard *shifted* grid strategy by Hochbaum and Maass [24]. The supercell and the boundary points are depicted in Figure 4.

▶ **Lemma 11.** *There exists a translation of S such that the number of boundary points is $O\left(\frac{n}{r}\right)$. The translation can be found in linear time.*

Proof. Shift the points of S by a random vector from $\{1, \ldots, r\}^2$. The probability that a point $p \in S$ is a boundary point is at most $\frac{4}{r}$, so the expected number of boundary points is at most $\frac{4n}{r}$. (It is straightforward to derandomize in linear time.) ◀

After applying Lemma 11 to build the grid, our algorithm proceeds in four steps:

1. We first compute the shortest paths between the $O\left(\frac{n}{r}\right)$ boundary points and all points in S. For this step, we can run the single-source algorithm of Section 2 $O\left(\frac{n}{r}\right)$ times, requiring $O\left(\frac{n^2}{r}\right)$ total time.
2. Next, for each supercell γ_i that contains more than a points or more than b boundary points, we compute the shortest paths between all points in $S \cap \gamma_i$ and all points in S. This time, we use the multi-source algorithm of Section 3. The points in γ_i can be grouped

into $O\left(\frac{|S\cap\gamma_i|}{k}+r^2\right)$ clusters of size at most k, since the supercell can be decomposed into $O\left(r^2\right)$ cells of diameter one. The number of supercells with more than a points is $O\left(\frac{n}{a}\right)$ and the number of supercells with more than b boundary points is $O\left(\frac{n}{rb}\right)$. Hence, the total time for this step is $O\left(\sum_i\left(\frac{|S\cap\gamma_i|}{k}+r^2\right)\cdot n\right) = O\left(\left(\frac{n}{k}+\left(\frac{n}{a}+\frac{n}{rb}\right)r^2\right)\cdot n\right) = O\left(\frac{n^2}{k}+\frac{n^2r^2}{a}+\frac{n^2r}{b}\right)$.

3. For each supercell γ_i with at most a points and at most b boundary points, we compute the shortest paths between all pairs of points in $S\cap\gamma_i$. For this step, we can run a naive cubic-time APSP algorithm on $S\cap\gamma_i$, after adding an extra weighted edge between each boundary point u in γ_i and each point $p\in S\cap\gamma_i$, with weight $\mathit{dist}_u[p]$, which we have computed in step 1. (These extra edges take care of the possibility that the shortest paths may not stay inside γ_i.) The total time is $O\left(\sum_i|S\cap\gamma_i|^3\right) = O\left(\sum_i|S\cap\gamma_i|a^2\right) = O\left(na^2\right)$.
4. For each supercell γ_i with at most a points and at most b boundary points, we compute the shortest paths between all points $p\in S\cap\gamma_i$ and all points $q\in S-\gamma_i$. Each such path must pass through a boundary point in γ_i, i.e., we want to find $\min_u\left(\mathit{dist}_u[p]+\mathit{dist}_u[q]\right)$ where the minimum is over all boundary points u in γ_i.
   We describe a table lookup method inspired by the planar-graph APSP algorithm in [6]. For each connected component of the unit-disk graph of $S\cap\gamma_i$, pick a *representative* boundary point (if one exists) among the points of this component. For each point $p\in S\cap\gamma_i$, let $\mathit{rep}(p)$ be the representative boundary point that lies in p's connected component. For each point $q\in S-\gamma_i$, define $\mathit{signature}[q]$ to be the vector containing $\mathit{signature}_u[q]=\mathit{dist}_u[q]-\mathit{dist}_{\mathit{rep}(u)}[q]$ over all boundary points u in γ_i. Observe that these values are bounded by $O\left(r^2\right)$ by the triangle inequality (since the distance between u and $\mathit{rep}(u)$ is at most $O\left(r^2\right)$), and so the vector can be encoded in $O\left(b\log r\right)$ bits. All these values have already been computed in step 1, and so the signatures of all points over all supercells can be generated in $O\left(\frac{n^2}{r}\right)$ total time. (We can ignore empty signatures, i.e., supercells that do not have any boundary points.)
   For each $p\in S\cap\gamma_i$ and each $q\in S-\gamma_i$,
   $$\min_u\left(\mathit{dist}_u[p]+\mathit{dist}_u[q]\right) \;=\; \mathit{dist}_{\mathit{rep}(p)}[q] \;+ \min_{u:\mathit{rep}(u)=\mathit{rep}(p)}\left(\mathit{dist}_u[p]+\mathit{signature}_u[q]\right).$$
   We can precompute the minimum in the right-hand side for each $p\in S\cap\gamma_i$ and each possible signature, and store all these minimums in a lookup table in $|S\cap\gamma_i|2^{O(b\log r)}$ time, for a total of $n2^{O(b\log r)}$ time.

The running time of the entire algorithm is

$$O\left(n\log n+n2^{O(k\log k)}+\frac{n^2}{r}+\frac{n^2}{k}+\frac{n^2r^2}{a}+\frac{n^2r}{b}+na^2+n2^{O(b\log r)}\right).$$

To balance the third and the fourth term, we set $k=r$. To balance the third and the sixth term, we set $r=\sqrt{b}$. To balance the third and the fifth term, we set $a=b^{3/2}$. Finally, we set $b=\frac{\varepsilon\log n}{\log\log n}$ for a sufficiently small constant ε, so that the second and the eighth terms get absorbed by the others. We then obtain the desired $O\left(n^2\sqrt{\frac{\log\log n}{\log n}}\right)$ time bound.

It is straightforward to modify the algorithm to retrieve any shortest-path distance in constant time, and retrieve any shortest path in time proportional to its size. It is also straightforward to modify the algorithm to compute the diameter (in step 4, for each $p\in S\cap\gamma_i$ and each possible signature, we need to find the point $q\in S-\gamma_i$ having this signature that maximizes $\mathit{dist}_{\mathit{rep}(p)}[q]$; all these maximums can be computed in $O\left(n\cdot\frac{n}{r}\right)$ total time by scanning the distance values from all boundary points). We conclude:

▶ **Theorem 12.** *Given a unit-disk graph we can compute an implicit representation of the shortest paths between all pairs, and the diameter of the graph, in* $O\left(n^2\sqrt{\frac{\log\log n}{\log n}}\right)$ *time in the RAM model.*

5 Conclusion

It is an intriguing open problem to compute the diameter of a unit-disk graph in truly subquadratic time, $O(n^{2-\varepsilon})$, for some positive constant ε. Another related problem is APSP in *weighted* unit-disk graphs, where the weight of an edge is defined as the Euclidean distance between the centers of the unit disks it connects.

Acknowledgement. We thank Anna Lubiw for posing the question of APSP in unit-disk graphs, and the participants of the algorithms problem session at Waterloo for discussion.

References

1 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43:429–449, 2014.

2 Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. *Journal of the ACM (JACM)*, 58(2):6, 2011.

3 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48:360–367, 2015.

4 Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

5 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

6 Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8:1–17, 2012.

7 Timothy M. Chan. Klee's measure problem made easy. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 410–419, 2013.

8 Timothy M. Chan. Speeding up the Four Russians algorithm by about one more logarithmic factor. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 212–217, 2015. doi:10.1137/1.9781611973730.16.

9 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2015.

10 Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov–Smolensky. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

11 Swarat Chaudhuri. Subcubic algorithms for recursive state machines. *ACM SIGPLAN Notices*, 43(1):159–169, 2008.

12 Bernard Chazelle and Wolfgang Mulzer. Computing hereditary convex structures. *Discrete & Computational Geometry*, 45(4):796–823, 2011.

13 Wlodzimierz Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International Journal of Computer Mathematics*, 32(1-2):49–60, 1990.

14 Esther Ezra and Wolfgang Mulzer. Convex hull of points lying on lines in $o(n\log n)$ time after preprocessing. *Computational Geometry*, 46(4):417–434, 2013.

15 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

16 Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:83–89, 1976.

17 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

18 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.

19 Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

20 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 621–630, 2014.

21 Yijie Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters*, 91(5):245–250, 2004.

22 Yijie Han. An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51(4):428–434, 2008.

23 Yijie Han and Tadao Takaoka. An $O(n^3\log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *Proceedings of the 13th Scandinavian Symposium on Algorithm Theory (SWAT)*, pages 131–141, 2012.

24 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

25 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

26 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog – with an application to Alt's conjecture. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1399–1413, 2014.

27 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

28 Franco P. Preparata and M. Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

29 Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011.

30 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

31 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 605–614, 1999.

32 Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, 1992. doi:10.1016/0020-0190(92)90200-F.

33 Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998.

34 Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Proceedings of the 10th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 278–289, 2004.

35 Tadao Takaoka. An $O(n^3\log\log n/\log n)$ time algorithm for the all-pairs shortest path problem. *Information Processing Letters*, 96(5):155–161, 2005.

36 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2014.

37 Christian Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time. *Computational Geometry*, 46(7):831–838, 2013.

38 Huacheng Yu. An improved combinatorial algorithm for Boolean matrix multiplication. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 1094–1105, 2015. doi:10.1007/978-3-662-47672-7_89.

39 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.

40 Uri Zwick. A slightly improved sub-cubic algorithm for the all pairsshortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006. doi:10.1007/s00453-005-1199-1.

Sink Evacuation on Trees with Dynamic Confluent Flows[*][†]

Di Chen[1] and Mordecai Golin[2]

1 Hong Kong University of Science and Technology, Hong Kong
2 Hong Kong University of Science and Technology, Hong Kong

Abstract

Let $G = (V, E)$ be a graph modelling a building or road network in which edges have-both travel times (lengths) and *capacities* associated with them. An edge's capacity is the number of people that can enter that edge in a unit of time. In emergencies, people evacuate towards the exits. If too many people try to evacuate through the same edge, *congestion* builds up and slows down the evacuation.

Graphs with both lengths and capacities are known as *Dynamic Flow networks*. An *evacuation plan* for G consists of a choice of exit locations and a partition of the people at the vertices into groups, with each group evacuating to the same exit. The *evacuation time* of a plan is the time it takes until the last person evacuates. The *k-sink evacuation problem* is to provide an evacuation plan with k exit locations that minimizes the evacuation time. It is known that this problem is NP-Hard for general graphs but no polynomial time algorithm was previously known even for the case of G a tree. This paper presents an $O(nk^2 \log^5 n)$ algorithm for the k-sink evacuation problem on trees, which can also be applied to a more general class of problems.

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph Algorithms–Facility location

Keywords and phrases Sink Evacuation, Dynamic Flow, Facility Location, Parametric Search

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.25

1 Introduction

Dynamic flow networks model movement of items on a graph.

Each vertex v is assigned some initial set of supplies w_v. Supplies flow across edges. Each edge e has a length – the time required to traverse it – and a capacity c_e, which limits the rate of the flow of supplies into the edge in one time unit. If all edges have the same capacity $c_e = c$ the network is said to have *uniform capacity*. As supplies move around the graph, *congestion* can occur as supplies back up at a vertex, increasing the time necessary to send a flow.

Dynamic flow networks were introduced by Ford and Fulkerson in [7] and have since been extensively used and analyzed. There are essentially two basic types of problems, with many variants of each. These are the *Max Flow over Time (MFOT)* problem of how much flow can be moved (between specified vertices) in a given time T and the *Quickest Flow Problem (QFP)* of how quickly a given W units of flow can be moved. Good surveys of the area and applications can be found in [19, 1, 6, 17].

[*] Extended abstract; a full version of this paper containing proofs is at `https://arxiv.org/abs/1607.08041`.

[†] Both authors partially supported by Hong Kong RGC GRF grant 16208415.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 25; pp. 25:1–25:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One variant of the QFP that is of interest is the transshipment problem, e.g., [12], in which the graph has several sources and sinks, with the original supplies being the sources and each sink having a specified demand. The problem is to find the minimum time required to satisfy all of the demands. [12] designed the first polynomial time algorithm for that problem, for the case of integral travel times.

Variants of QFP Dynamic flow problems can also model [10] *evacuation problems.* In these, vertex supplies are people in a building(s) and the problem is to find a routing strategy (evacuation plan) that evacuates all of them to specified sinks (exits) in minimum time. Solving this using (integral) dynamic flows, would assign each person an evacuation path with, possibly, two people at the same vertex travelling radically different paths.

A slightly modified version of the problem, the one addressed here, is for the plan to assign to each vertex v exactly one exit or *evacuation edge*, i.e., a sign stating “this way out”. All people starting or arriving at v must evacuate through that edge. After traversing the edge they follow the evacuation edge at the arrival vertex. They continue following the unique evacuation edges until reaching a sink, where they exit. The simpler optimization problem is, given the sinks, to determine a plan minimizing the total time needed to evacuate everyone; we call this the *k-sink assignment problem.* A more complicated version is, given k, to find the (vertex) locations of the k sinks/exits *and* associated evacuation plan that together minimizes the evacuation time. This is the *k-sink location problem*, which we also refer to as ‘the’ k-sink evacuation problem on trees.

Flows with the property that all flows entering a vertex leave along the same edge are known as *confluent*[1]; even in the static case constructing an optimal confluent flow is known to be very difficult. i.e., if P $\neq$ NP, then it is even impossible to construct a constant-factor approximate optimal confluent flow in polynomial time on a general graph [4, 5, 3, 18].

Note that if the capacities are “large enough” then no congestion can occur and every person follows the shortest path to some exit with the cost of the plan being the length of the maximum shortest path. This is exactly the k-center problem on graphs which is already known to be NP-Hard [9]. Unlike k-center, which is polynomial-time solvable for fixed k, Kamiyama *et al.* [13] proves by reduction from the `Partition` problem, that, even for fixed $k = 1$ finding the min-time evacuation protocol is still NP-Hard for general graphs.

The only known solvable case for general k is for G a path. For paths with uniform capacities, [11] gives an $O(kn)$ algorithm.[2]

When G is a tree the 1-sink location problem can be solved [16] in $O(n \log^2 n)$ time. This can be reduced [10] down to $O(n \log n)$ for the uniform capacity version, i.e., all the c_e are identical. For the assignment problem, [15] gives an $O(n^2 k \log^2 n)$ algorithm for finding the minimum time evacuation protocol. i.e., a partitioning of the tree into subtrees that evacuate to each sink. The best previous known time for solving the k-sink location problem was $O(n(c \log n)^{k+1})$, where c is some constant [14].

1.1 Our contributions

This paper gives the first polynomial time algorithm for solving the k-sink location problem on trees. Our result uses the $O(n \log^2 n)$ algorithm of [15], for calculating the evacuation time of a tree given the location of a sink, as an oracle.

▶ **Theorem 1.** *The k-sink location problem for evacuation can be solved in time $O(nk^2 \log^5 n)$.*

[1] Confluent flows occur naturally in other problems e.g. packet forwarding and railway scheduling [5].

[2] This is generalized to the general capacity path to $O(kn \log^2 n)$ in the unpublished [2].

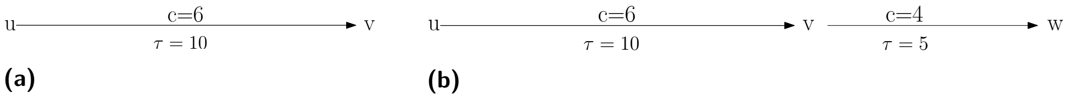


Figure 1 In (a), if $w_u = 20$ the last person leaving u arrives at v at time $t = 13$. In (b) Assume people at u, v are all evacuating to w and $w_u = 20$ and $w_v > 0$. The first person from u arrives at v at time $t = 11$. If $w_v \leq 40$ all of the people on v enter (v, w) *before or at* $t = 10$, so there will be no congestion when the first people from u arrive at v and they just sail through v without stopping. The last people from u reach w at time $t = 20$.; but if $w_v > 40$, some people who started at v will still be waiting at v when the first people from u arrive there. In this case, there is congestion and the people from u will have to wait. After waiting, the last person from u will finally arrive at w at time $14 + \lfloor (20 + w_v)/4 \rfloor$.

It is instructive to compare our approach to Frederickson's [8] $O(n)$ algorithm for solving the k-center problem on trees, which was built from the following two ingredients.

1. An $O(n)$ time previously known algorithm for checking *feasibility*, i.e., given $\alpha > 0$, testing whether a k-center solution with cost $\leq \alpha$ *exists*
2. A clever *parametric search* method to filter the $O(n^2)$ pairwise distances between nodes, one of which is the optimal cost, via the feasibility test.

Section 3, is devoted to constructing a first polynomial time feasibility test for k-sink evacuation on trees. It starts with a simple version that makes a polynomial number of oracle calls and then is extensively refined so as to make only $O(k \log n)$ (amortized) calls.

On the other hand, there is no small set of easily defined cost values known to contain the optimal solution. We sidestep this issue by doing parametric searching *within* our feasibility testing algorithm, Section 4, which leads to Theorem 1.

As a side result, a slight modification to our algorithm allows improving, for almost all k, the best previously known algorithm for the k-sink assignment problem, from $O(n^2 k \log^2 n)$ [15] down to $O(nk^2 \log^4 n)$, as justified in the full paper.

2 Formal definition of the sink evacuation problem

Let $G = (V, E)$ be an undirected graph. Each edge $e = (u, v)$ has a travel time τ_e; flow leaving u at time $t = t_0$ arrives at v at time $t = t_0 + \tau_e$. Each edge also has a *capacity* $c_e \geq 0$. This restricts at most c_e units of resource to enter edge e in one unit of time. For our version of the problem we restrict c to be integral; the capacity can then be visualized as the *width* of the edge with only c_e people being allowed to travel in parallel along the edge.

Consider w_u people waiting at vertex u at time $t = 0$ to travel the edge $e = (u, v)$. Only c_e people enter the edge in one time unit, so the items travel in $\lceil w_u / c_e \rceil$ packets, each of size c_e, except possibly for the last one. The first packet enters e at time $t = 0$, the second at time $t = 1$, etc.. The first packet therefore reaches v at time $t = \tau_e$ time, the second at $t = \tau_e + 1$ and the last one at time $t = \tau_e + \lceil w_u / c_e \rceil - 1$. Figure 1(a) illustrates this process. In the diagram people get moved along (u, v) in groups of size at most 6. If $w_u = 20$, there are 4 groups; the first one reaches v at time $t = 10$, the second at time $t = 11$, the third at $t = 12$ and the last one (with only 2 people) at $t = 13$.

Now suppose that items are travelling along a path $\ldots u \to v \to w \to \ldots$ where $e_1 = (u, v)$ and $e_2 = (v, w)$. Items arriving at v can't enter e_2 until the items already there have left. This waiting causes *congestion* which is one of the major complications involved in constructing good evacuation paths. Figure 1(b) illustrates how congestion can build up.

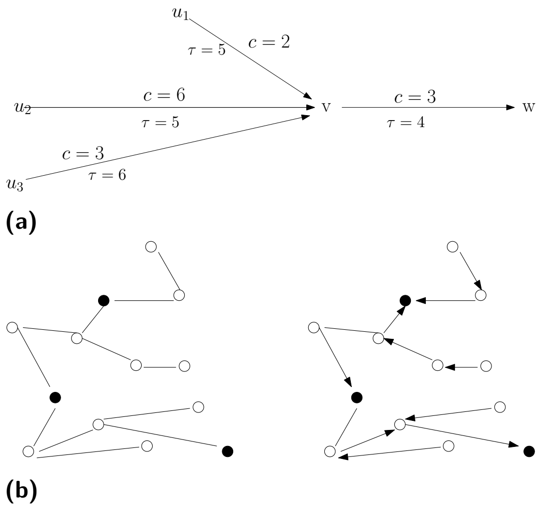


Figure 2 (a) evacuation problem with 4 vertices and sink at w. If $w_v = 8$, $w_{u_1} = 4$, $w_{u_2} = 10$, $w_{u_3} = 11$ and sink at w, the last person arrives w at time 15. In (b) the left figure is a tree with the $k = 3$ black vertices as sinks. The right figure provides an evacuation plan. Each non-sink vertex v has exactly one outgoing edge and, following the directed edges from each such v leads to a sink.

As another example, consider Figure 2(a) with every node evacuating to w. When the first people from u_1 arrive at v, some of the original people still remain there, leading to congestion. Calculation shows that the last people from u_1 leave v at time 4 so when the first people from u_2 arrive at v at time 5, no one is waiting at v. But, when the first people from u_3 arrive at v some people from u_2 are waiting there, causing congestion. After that, people arrive from both u_2 and u_3 at the same time, with many having to wait. The last person finally reaches w at time 15, so the evacuation protocol takes time 15.

Given a graph G, distinguish a subset $S \subseteq V$ with $|S| = k$ as sinks (exits). An evacuation plan provides, for each vertex $v \notin S$, the unique edge along which all people starting at or passing through v evacuate. Furthermore, starting at any v and following the edges will lead from v to one of the S (if $v \in S$, people at v evacuate immediately through the exit at v). Figure 2(b) provides an example.

Note that the evacuation plan defines a confluent flow. The evacuation edges form a directed forest; the sink of each tree in the forest is one of the designated sinks in S.

Given the evacuation plan and the values w_v specifying the initial number of people located at each node, one can calculate, for each vertex, the time (with congestion) it takes for all of its people to evacuate. The maximum of this over all v is the minimum time required to evacuate *all* people to some exit using the rules above. Call this the *cost for S associated with the evacuation plan.* The *cost* for S will be the minimum cost over all evacuation plans using that set S as sinks.

The *k-sink location* problem is to find a subset S of size k with minimum cost. Recall that [15] provides an $O(n \log^2 n)$ problem for solving this problem if for tree G with $k = 1$. We will use this algorithm as an oracle for solving the general k-location problem on trees.

Given the hardness results, there may not exist an efficient algorithm for general graphs, but our algorithms can serve as fast subroutines for exhaustive search or heuristic methods.

2.1 General problem formulation

The input to our algorithm(s) will be a tree $T_{\text{in}} = (V_{\text{in}}, E_{\text{in}})$, and a positive integer k. Let $n = |V_{\text{in}}| = |E_{\text{in}}| + 1$. Our goal will be to find a subset $S \subseteq V_{\text{in}}$ with cardinality at most k that can minimize the cost denoted by $F(S)$. This will essentially involve partitioning the V_{in} into $\leq k$ subtrees that minimizes their individual max costs.

In the main part of the paper, we denote by $f(U, s)$, where $s \in U$, the time taken by all people from nodes in U to evacuate to s. So given a set of sinks S, and a partition $\mathcal{P}$ of V_{in} where $|S \cap P| = 1$ for all $P \in \mathcal{P}$, the total evacuation time is given by $\max_{P \in \mathcal{P}} f(P, S \cap P)$.

Our solution will actually work for any $f(U, v)$ that is a *monotone min-max cost* function. This is a clean abstraction of evacuation functions that allows us to cleanly formulate and understand proofs. Formulated this way, our techniques permit solving other related problems. See the full paper for details.

We also write our proofs in the full paper under such framework. Our algorithms are designed to make calls directly to an oracle $\mathcal{A}$ that, given any U that induces a connected component of T_{in} and any $v \in U$, computes a monotone min-max cost function $f(U, v)$. In general such a polynomial time oracle must exist for the problem to even be in NP. As we fully account for the time used to call the oracle in any way, there is no material difference whether the oracle is considered a part of the algorithm or given in the input.

3 Bounded cost k-sink (feasibility test)

To tackle the more complicated general formulation, we first consider a simplification, which is a decision problem whether all nodes can be evacuated given a time limit $\mathcal{T}$, with k sinks. We call the general version of this problem "bounded cost minmax k-sink". We will use an algorithm solving this problem as a subroutine for solving the full problem; we measure the time complexity by the number of calls to the oracle $\mathcal{A}$, as follows.

▶ **Theorem 2.** *If $\mathcal{A}$ runs in time $t_{\mathcal{A}}(n)$, the bounded cost minmax k-sink problem can be solved in time $O(k \max(t_{\mathcal{A}}(n), n) \log n)$.*

▶ **Definition 3.** A *feasible configuration* is a set of sinks $S \subseteq V$ with a partition $\mathcal{P} \in \Lambda(S)$ where $F_S(\mathcal{P}) \leq \mathcal{T}$; S is also separately called a *feasible sink placement*, and $\mathcal{P}$ is *a partition witnessing the feasibility of* S. An optimal feasible configuration is a *feasible sink placement $S^* \subseteq V$ with minimum cardinality*; we write $k^* := |S^*|$.

If $k^* > k$ then the algorithm returns 'No'. Otherwise, it returns a feasible configuration $(S_{\text{out}}, \mathcal{P}_{\text{out}})$ such that $|S_{\text{out}}| \leq k$.

▶ **Definition 4.** Suppose U induces a subtree of T_{in} and $S \subseteq U$. We say U can be *served by* S if, for some partition $\mathcal{P}$ of U, for each $P \in \mathcal{P}$ there exists $s \in S$ such that $f(P, s) \leq \mathcal{T}$.

▶ **Definition 5.** Let U be the nodes of a connected component of G and $v \in V$ (not necessarily in U). We say that v *supports* U if one of the following holds:

- If $v \in U$, then $f(U, v) \leq \mathcal{T}$.
- If $v \notin U$, let Π be the set of nodes on the path from v to U. Then $f(U \cup \{v\} \cup \Pi, v) \leq \mathcal{T}$.

If U can be served by S, then any node in U is supported by some $s \in S$. The converse is not necessarily true.

3.1 Greedy construction

Our algorithms greedily build S_{out} and $\mathcal{P}_{\text{out}}$ on-the-fly, making irrevocable decisions on what should be in the output. S_{out} is initialized to be empty. In each step, we add elements to S_{out} but never remove them, and once $|S_{\text{out}}| > k$ we immediately terminate with a 'No'. If, at termination, $|S_{\text{out}}| \leq k$, we output S_{out}.

Similarly, $\mathcal{P}_{\text{out}}$ is initially empty, and the algorithm performs irrevocable updates to $\mathcal{P}_{\text{out}}$ while running. An update to $\mathcal{P}_{\text{out}}$ is a *commit.* When set $P_{\text{new}} \subseteq V_{\text{in}}$ is committed it is associated with some some sink in S_{out} (which might have to be added to S_{out} at the same time). If P_{new} shares its sink with an existing block $P \in \mathcal{P}_{\text{out}}$, we merge P_{new} into P. Another way to view this operation is that either a new sink is added, or unassigned nodes are assigned to an existing sink.

In essence, we avoid backtracking so that S_{out} does not lose elements, and blocks added to $\mathcal{P}_{\text{out}}$ can only grow. For this to work, we must require, throughout the algorithm:

(C1) An optimal feasible sink placement S^* exists where $S_{\text{out}} \subseteq S^*$.

(C2) For any $P \in \mathcal{P}_{\text{out}}$ there exists a unique $s \in S_{\text{out}}$ such that $|P \cap S_{\text{out}}| = 1$, and $f(P, s) \leq \mathcal{T}$.

Additionally, $\mathcal{P}_{\text{out}}$ will be a partition of V_{in} upon termination with 'yes'. When these conditions all hold, then $|\mathcal{P}_{\text{out}}| \leq k$ and $(S_{\text{out}}, \mathcal{P}_{\text{out}})$ is feasible and output by the algorithm.

3.1.1 A separation argument

As the algorithm progresses, it removes nodes from the remaining graph (the *working tree*), simplifying the combinatorial structure. We will need the definitions below:

► **Definition 6** (Self sufficiency and $T_{-v}(u)$)**.** A subtree $T' = (V', E')$ of T_{in} is *self-sufficient* if V' can be served by $S_{\text{out}} \cap V'$.

Given a tree $T = (V, E)$, consider an internal node $v \in V$ and one of its neighbors $u \in V$. Removing v from T leaves a forest $\mathcal{F}_{-v}$ of disjoint subtrees of T. Then there is a unique tree $T' = (V', E') \in \mathcal{F}_{-v}$ such that $u \in V'$, denoted by $T_{-v}(u) = (V_{-v}(u), E_{-v}(u))$. The concept of self sufficiency is introduced for subtree of this form.

Roughly speaking, if $T_{-v}(u)$ is self-sufficient, and u is a sink,

there is no need to add any other sinks to $T_{-v}(u)$, also no node outside $T_{-v}(u)$ will be routed to any sink in $T_{-v}(u)$ other than u. This means all nodes in $V_{-v}(u)$ except u can be removed from consideration; a more formal statement of this fact is given in the full version.

Throughout the algorithm, we maintain a 'working' tree $T = (V, E)$ as well as a working set of sinks $S = S_{\text{out}} \cap V$. Initially, $T = T_{\text{in}}$. As the algorithm progresses, T is maintained to be a subtree of T_{in} by peeling off self-sufficient subtrees, which ensures that solving the bounded problem on T is equivalent to solving the bounded problem on T_{in}.

For this to work, we enforce that sink s is added to S_{out} and S only when, for some neighbor u of s, the tree induced by $V_{-s}(u) \cup \{s\}$ is self-sufficient with respective to the sink set $S \cup \{s\}$. This permits removing $V_{-s}(u)$ from T after adding s to S_{out} and S. So in the algorithm we can assume that sinks exist only at the leaves of the working tree T.

3.2 Subroutine: Peaking Criterion

We now describe a convenient mechanism that allows us to greedily add sinks.

▶ **Definition 7** (Peaking criterion)**.** Given $T = (V, E)$, the ordered pair of points $(u, v) \in V \times V$ satisfies the *peaking criterion* (abbreviated *PC*) if and only if u and v are neighbors, $V_{-v}(u) \cap S$ is empty, and finally $f(V_{-v}(u), u) \leq \mathcal{T}$ but $f(V_{-v}(u) \cup \{v\}, v) > \mathcal{T}$.

▶ **Lemma 8.** *Let S be a feasible sink placement for T, and let $u, v \in V$ be neighbors. If (u, v) satisfies the peaking criterion, then $S' := (S \backslash V_{-v}(u)) \cup \{u\}$ is also a feasible sink placement. In particular, if S is an optimal feasible sink placement, then so is S'.*

If (u, v) satisfies the peaking criterion, we can immediately place a sink at u and then commit $V_{-v}(u)$. The following demonstrates that, whenever $S = \emptyset$, at least 1 sink can be found using the peaking criterion, unless a single node can $s \in V$ support the entire graph.

▶ **Lemma 9.** *Suppose for some v, u, $f(V_{-v}(u) \cup \{v\}, v) > \mathcal{T}$, and $S \cap V_{-v}(u) = \emptyset$. Then there exists a pair of nodes $s, t \in V_{-v}(u) \cup \{v\}$ such that (s, t) satisfies the peaking criterion.*

▶ **Corollary 10.** *Given $S = \emptyset$, either one of the following occurs:*
1. *For any $s \in V$ we have $f(V, s) \leq \mathcal{T}$, or*
2. *There exist a pair of nodes $u, v \in V$ that satisfies the peaking criterion.*

At stages where it is applicable, for each ordered pair (u, v) that satisfies the peaking criterion we place a sink at u and remove nodes in $V_{-v}(u)$. If instead the first case of the above corollary occurs, we can add an arbitrary $s \in V$ to S and S_{out} and terminate.

3.3 Subroutine: Reaching Criterion

Corollary 10 provides two ways to add sinks to S_{out}. The peaking criterion is a way to add sinks to T and remove certain nodes from T. On the other hand, the reaching criterion (RC) is a way to *remove* sinks from T and S, while keeping them in S_{out}. Roughly speaking, the reaching criterion finalizes all nodes that should be assigned to certain sinks, and then removes all these nodes from consideration. To forumlate RC, we first introduce the hub tree, which has convenient properties that arise from applying the peaking criterion.

▶ **Definition 11** (Hubs)**.** Let $L \subseteq V$ be the leaves of the rooted tree $T = (V, E)$. Let $S \subseteq L$, be a set of sinks, with no sink in $V \backslash L$. Let $H(S) \subseteq V$ be the set of lowest common ancestors of all pairs of sinks in T. The nodes in $H(S)$ are the *hubs* associated with S.

The *hub tree* $T_{H(S)} = (V_{H(S)}, E_{H(S)})$ is the subgraph of T that includes all vertices and edges along all possible simple paths among nodes $H(S) \cup S$.

▶ **Definition 12** (Outstanding branches)**.** Given $T = (V, E)$ and S, we say that a node $w \in V$ branches out to η if η is a neighbor of w in T that does not exist in $V_{H(S)}$. The subtree $T' := T_{-w}(\eta)$ is called an *outstanding branch*; we say that T' is *attached* to w.

▶ **Definition 13** (Bulk path)**.** Given two distinct $u, v \in V_{H(S)}$, the *bulk path* $\text{BP}(u, v)$ is the union of nodes along the unique path Π between u, v (inclusive), along with all the nodes in all outstanding branches that are attached to any node in Π.

Given T and S, we say a node $v \in E_{H(S)}$ *can evacuate* to $s \in S$ if $f(\text{BP}(v, s), s) \leq \mathcal{T}$; when such $s \in S$ exists for v, we say that v *can evacuate.* Given this we can formulate an ‘opposite’ to the peaking criterion, which allows us to remove nodes, including sinks, from T.

▶ **Definition 14** (Reaching criterion)**.** Given $T = (V, E)$ and a set of sinks S, placed at the leaves of T. Let T be RC-viable with respect to S and $(u, v) \in V \times V$ be an ordered pair of nodes. u, v satisfy the *reaching criterion (RC)* if and only if they are neighbors in T, and $T_{-v}(u)$ is self-sufficient while the tree induced by $\text{BP}(v, u) \cup V_{-v}(u)$ is not.

A crucial property arises after an exhaustive application of the peaking criterion.

▶ **Definition 15** (RC-viable)**.** Given T and sinks S, we say that T is *RC-viable* if:

1. all sinks S occur at the leaves of T
2. if $T' = (V', E')$ is an outstanding branch attached to $w \in V_{H(S)}$, then $f(V' \cup \{w\}, w) \leq \mathcal{T}$

▶ **Lemma 16.** *Given $T = (V, E)$ and sinks S, where S is a subset of leaves of T. Suppose no ordered pair $(u, v) \in V \times V$ satisfy the peaking criterion. Then T is RC-viable.*

Now when T is RC-viable w.r.t. S, there is no need to place sinks within outstanding branches; this is because if an outstanding branch is attached to a node w, then a sink at w can already serve the entire outstanding branch.

▶ **Theorem 17.** *Suppose $T = (V, E)$ is RC-viable with respect to $S \subseteq V$. If $u, v \in V$ satisfies the reaching criterion, then we can remove $T_{-v}(u)$ from T, and also commit all blocks in the partitioning of $T_{-v}(u)$ that witnesses the self-sufficiency of $T_{-v}(u)$. By definition, $T_{-v}(u)$ includes at least one sink from S.*

After removing $T_{-v}(u)$ by the reaching criterion, we need to run the peaking criterion again on T, in order to preserve RC-viability. Then, we can apply RC again. In this way, we interleave the invocations of PC and RC to gradually eliminate nodes from the working tree.

3.3.1 Testing for self-sufficiency

In order to make use of the reaching criterion, we require efficient tests for self-sufficiency. Note that [15] readily gives such test albeit at a higher time complexity. In our algorithm, we perform self-sufficiency tests on a rooted subtree T' only if it satisfies some special conditions, allowing us to exploit RC-viability and reuse past computations. By our arrangements, when such T' passes our test we know it demonstrates a stronger form of self-sufficiency.

▶ **Definition 18** (Recursive self-sufficiency)**.** Given a rooted subtree $T' = (V', E')$ of T, $V' \cap S \neq \emptyset$, we say that T' is *recursively self-sufficient* if for all $u \in V_{H(S)} \cap V'$, the subtree of T' rooted at u is self-sufficient.

A bottom-up approach can be used to test for recursive self-sufficiency, which in turn implies 'plain' self-sufficiency.

▶ **Lemma 19.** *Given a RC-viable rooted subtree $T' = (V', E')$ of T, $V' \cap S \neq \emptyset$, where v is the root. Suppose there exists a child u of v in $V_{H(S)} \cap V'$ such that $T_{-v}(u)$ is recursively self-sufficient, and there is a sink $s \in S \cap V_{-v}(u)$ such that v can evacuate to s.*

Then $BP(v, s) \cup V_{-v}(u)$ is recursively self-sufficient. If, additionally, for every child u' of v in $V_{H(S)} \cap V'$, $T_{-v}(u')$ is recursively self-sufficient, then T' is recursively self-sufficient.

We say that s is a witness to Lemma 19 for T' and v; we store this witness, as well as the witness for every subtree of T' rooted at some $v \in V'$. One can retrieve a partition $\mathcal{P}'$ of T' that witnesses the self-sufficiency of T', in $O(|V'|)$ time. For this to be useful, note that only recursive self-sufficiency will be relevant. When a RC-viable tree is self-sufficient but not recursively self-sufficient, if we process bottom-up, we can always cut off part of the tree using the reaching criterion, so that the remainder is recursively self-sufficient. This is demonstrated in the detailed algorithm, in the full paper.

3.4 Combining the Pieces

The main ingredients of our algorithm are the peaking and reaching criteria along with ideas to test self-sufficiency. We use the peaking criterion to add sinks to T, and then the reaching criterion to remove sinks and nodes from T, until either T is empty or T can be served by a single sink. In the following we describe a full algorithm that makes use of these ideas.

3.4.1 Simpler, iterative approach ('Tree Climbing')

Essentially, in this algorithm we iteratively check and apply the PC and RC bottom-up from the leaves. We do not specify a root here; the root can be arbitrary, and changed whenever necessary. As we go up from the leaves, for each pair (u, v) that forms an edge of the tree, we would call the oracle $\mathcal{A}$ for $f(V_{-v}(u), u)$, $f(V_{-v}(u) \cup \{v\}, v)$ or $f(\mathrm{BP}(v, s), s)$ for some sink s, and apply either the peaking criterion or the reaching criterion. By design RC is checked whenever the tree is RC-viable, and PC is checked whenever the tree is not RC-viable, and we do not need to test both on the same pair (u, v).

▶ **Lemma 20.** *The bounded-cost tree-climbing algorithm makes $O(n)$ calls to $\mathcal{A}$.*

Proof. We only make $O(1)$ calls to evaluate $f(\cdot, \cdot)$ for each pair $(u, v) \in V_{\mathrm{in}}$. ◀

After seeing the iterative approach, it is easier to understand the more advanced algorithm, which uses divide-and-conquer and binary search to replace the iterative processes.

3.4.2 Peaking criterion by recursion

Macroscopically, we replace plain iteration with a fully recursive process. We do this once in the beginning, as well as every time we remove a sink. Overall the algorithm makes $O(k \log n)$ 'amortized' calls to the oracle. Recall that the main purpose of the peaking criterion is to place sinks and make the tree T RC-viable.

A localized view

We start with a more intuitive, localized view of the recursion. We evaluate $f(\cdot, \cdot)$ on sets of nodes of the form $V_{-v}(u)$ or $V_{-v}(u) \cup \{v\}$. If $f(V_{-v}(u), u) \leq \mathcal{T}$ then we mark all nodes in $V_{-v}(u)$; note that if $f(V_{-v}(u) \cup \{v\}, u) > \mathcal{T}$ but all nodes in $V_{-v}(u)$ are marked, then PC can be invoked and $V_{-v}(u)$ is cut off. Sometimes we also mark the node v, if all but one of its neighbors (that have not been cut off) are marked.

Over the course of the algorithm, we are given a node $v \in V$ (along with other information including which other nodes are marked), and for each neighbor u of v we decide whether to evaluate $a_u := f(V_{-v}(u) \cup \{v\}, v)$. As a basic principle to save costs, we do not wish to call the oracle if all nodes in $V_{-v}(u) \cup \{v\}$ are marked, or if $V_{-v}(u) \cup \{v\}$ contains a sink.

When we do get $a_u \leq \mathcal{T}$, mark all nodes in $V_{-v}(u)$. Moreover, if no more than 1 neighbor of v is unmarked, and by this time $v \notin S_{\mathrm{out}}$, we also mark v. This part is the same in tree-climbing, and maintains *an important invariant* regarding the set of marked nodes: if u is marked but a neighbor v is not, then all nodes in $V_{-v}(u)$ are marked, and $f(V_{-v}(u) \cup \{v\}, v) \leq \mathcal{T}$.

On the other hand, if in fact we find that $a_u > \mathcal{T}$, we would wish to recurse into $T_{-v}(u)$, because one sink must be placed in it. Now we return to a more global view.

A global view

To maintain RC-viability we need to apply the oracle on various parts of T. In the iterative algorithm, this process is extremely repetitive. Now we wish to segregate different sets of nodes on the tree, so the oracle is only applied to separate parts.

▶ **Definition 21** (Compartments and Boundaries)**.** Let $T' = (V', E')$ be a subtree of $T = (V, E)$. The boundary $\delta T'$ of T' is the set of all nodes in T' that is a neighbor of some node in $V \backslash V'$.

Now given a set of nodes W of a tree $T = (V, E)$, the set of compartments $\mathcal{C}_T(W)$ is a set of subtrees of T, where the union of all nodes is V, and for each $T' = (V', E') \in \mathcal{C}_T(W)$, V' is a maximal set of nodes that induces a subtree T'' of T such that $\delta T'' \subseteq W$.

Intuitively, the set of compartments is induced by first removing W, so that T is broken up into a forest of smaller trees, and for each of the small trees we re-add nodes in W that were attached to it, where the reattached nodes are called the boundary. As opposed to partitioning, two compartments may share nodes at their boundaries.

In the algorithm, we generate a sequence of sets $W_0 \subseteq W_1, ..., W_t \subseteq V$ in the following manner: W_0 contains the tree median of T, and then to create W_i from W_{i-1} we simply add to W_i the tree medians of every compartment in $\mathcal{C}_T(W_i)$.

For each i, we only make oracle calls that take the form $f(V_{-v}(u), s)$ or $f(V_{-v}(u) \cup \{v\}, s)$. We avoid choices of (u, v) that will cause evaluation on overlapping sets, based on information gained on processing W_{i-1} in the same way. In this way we only make essentially $O(1)$ 'amortized' calls to the oracle for each i. For details see the full paper.

After removing nodes via the reaching criterion, we only need to do this on a subtree of T, which we can assume takes the same time as on the full tree. One can see that $t = O(\log n)$ thus the peaking criterion takes at most $O(k \log n)$ amortized oracle calls.

3.4.3 Reaching criterion by Binary Search

Intuitively, with the reaching criterion we look for an edge (u, v) in $T_{H(S)}$ so that $T_{-v}(u)$, which contains at least one sink, can be removed. Now given *adjacent* hubs h_1 and h_2, i.e. hubs where there are no other hubs along the path from h_1 to h_2, consider any subtree of T rooted at h_1, in which h_2 is a descendent of h_1. Then exactly one of the following is true:

(P1) There is an edge (u, v) in the path $\Pi(h_1, h_2)$ between h_1 and h_2, where u is a child of $v \neq h_1$, such that $T_{-v}(u)$ is recursively self-sufficient, but the subtree rooted at v is not.

(P2) Let u be the child of h_1 that is on the path between h_1 and h_2. Then the subtree rooted at u, i.e. $T_{-h_1}(u)$, is recursively self-sufficient.

As h_1 and h_2 are adjacent hubs, for any edge (u, v) along the path, where $v \neq h_1$ is the parent of u, the subtree rooted at v is recursively self sufficient only if the subtree rooted at u is. Suppose we know that the subtree rooted at h_2 is recursively self-sufficient.

In the iterative algorithm we move upwards from h_2 to h_1 gradually until we find such an edge, or upon reaching h_1; this can be replaced by a binary search. This idea will let us only use $O(k^2 \log n)$ calls; proper amortization with pruning can reduce this to $O(k \log n)$ oracle calls. See full paper. Theorem 2 follows from the above faster algorithm.

4 Full problem: cost minimization

Given an algorithm for the bounded cost problem, we may perform a binary search over possible values of $\mathcal{T}$ for the minimal $\mathcal{T}^*$ allowing evacuation with k sinks. To produce a *strongly* polynomial time algorithm, at a higher level, we wish to search among a finite, discrete set of possible values for $\mathcal{T}^*$. This can be done by a *parametric searching* technique.

4.1 Iterative approach

We start by modifying the iterative algorithm for bounded cost. In that algorithm, the specific value of $\mathcal{T}$ dictates the contents of T, S, S_{out} etc., as well as which node pairs satisfy either of PC and RC, at each step of that depends upon the outcome of a comparison of the form $f(\cdot,\cdot) \leq \mathcal{T}$, obtained from calling the oracle.

The idea is to run a parametric search version of the bounded cost algorithm. $\mathcal{T}$ will no longer be a constant; we *interfere* the normal course of the algorithm by changing $\mathcal{T}$ during runtime. The decision to interfere is based on a *threshold margin* $(\mathcal{T}^L, \mathcal{T}^H]$ that we maintain, to keep track of candidate values of $\mathcal{T}^*$. Initially, $(\mathcal{T}^L, \mathcal{T}^H] = (-\infty. + \infty]$, and $\mathcal{T} = 0$.

The following process terminates with some $\mathcal{T} \in (\mathcal{T}^L, \mathcal{T}^H]$; call this the 'bounded cost algorithm with interference'. Every time we evaluate $a = f(\cdot,\cdot)$, we set $\mathcal{T}$ based on the following, before making the comparison $a \leq \mathcal{T}$ and proceeding with the relevant if-clause.

1. If $a \leq \mathcal{T}^L$, set $\mathcal{T} = \mathcal{T}^L$, so the if-clause always resolves as $f(\cdot,\cdot) \leq \mathcal{T}$.
2. If $a > \mathcal{T}^H$, set $\mathcal{T} = \mathcal{T}^H$, so the if-clause always resolves as $f(\cdot,\cdot) > \mathcal{T}$.
3. If $a \in (\mathcal{T}^L, \mathcal{T}^H]$, run a separate clean, non-interfered instance of the bounded cost algorithm with threshold value $\mathcal{T} := a$, and observe the output.
   - Output is 'No': set $\mathcal{T}^L := a$, and $\mathcal{T} := a$, resolving the if-clause as $a = f(\cdot,\cdot) \leq \mathcal{T} = a$.
   - Otherwise, set $\mathcal{T}^H := a$, and $\mathcal{T} := \mathcal{T}^L$.

▶ **Lemma 22.** *Let $(\mathcal{T}_<, \mathcal{T}_>]$ be the threshold margin at the end of the bounded cost algorithm with interference. Then $\mathcal{T}_> = \mathcal{T}^*$. In particular, we can then run the bounded cost algorithm (non-interfered) on $\mathcal{T} := \mathcal{T}_>$ to retrieve the optimal feasible configuration.*

▶ **Theorem 23.** *Minmax tree facility location can be solved in $O(n^2)$ calls to $\mathcal{A}$.*

Proof. We always allow the interfered algorithm to make progress, albeit with changing values of $\mathcal{T}$, so Lemma 20 still applies; $f(\cdot,\cdot)$ is evaluated at most $O(n)$ times in the interfered algorithm, thus we also launch a separate instance of the feasibility test $O(n)$ times. ◀

4.2 Using divide-and-conquer and binary search

The above idea still works for applying RC, that we can use the same ideas of calling the feasibility test and interfering as we evaluate $f(\cdot,\cdot)$. Thus we only interfere $O(k \log n)$ times, making $O(k^2 \log^2 n)$ total calls to the oracle.

But it does not work well with the peaking criterion; that the divide-and-conquer algorithm for the peaking criterion relies very strongly on amortization, and a naive application of interference will perform $O(n)$ feasibility tests, while we aim for $O(k \log n)$.

The basic idea is to filter through values of $f(\cdot,\cdot)$ where we decide to interfere. Intuitively, the divide and conquer algorithm can be organized in t layers in reference to $W_1, ..., W_t$ where $t = O(\log n)$, for each we evaluate $f(\cdot,\cdot)$ on certain pairs of nodes and sets. Each evaluation of $f(\cdot,\cdot)$ can be identified with an edge of T, thus in each layer we have at most $O(n)$ evaluations, producing a list of $O(n)$ values.

Thus, at each layer we evaluate $f(\cdot,\cdot)$, and binary search for a pair of values $a_<, a_>$ such that $a_< \leq \mathcal{T}^* < a_>$, making $O(\log n)$ calls to the bounded-cost algorithm, and then set $\mathcal{T} = a_<$ when proceeding to mark nodes and place sinks, before moving to the next layer.

This gives $O(\log^2 n)$ calls for a single application of the peaking criterion. As we only need to apply the peaking criterion $O(k)$ times, the resulting number of calls to the feasibility test is $O(k \log^2 n)$. As each call takes $O(nk \log^3 n)$ time, Theorem 1 then follows.

5 Conclusion

Given a Dynamic flow network on a tree $T_{\text{in}} = (V, E)$ we derive an algorithm for finding the locations of k sinks that minimize the maximum time needed to evacuate the entire graph. Evacuation is modelled using dynamic confluent flows. Only an $O(n \log^2 n)$ time algorithm for solving the one-sink ($k = 1$) case was previously known.

This paper gives the first polynomial time algorithm for solving the arbitrary k-sink problem, developed in two parts. Section 3 gives an $O(nk \log^3 n)$ algorithm to test the feasibility of completing evacuation in time $\mathcal{T}$ with k sinks. Section 4 showed how to modify this to an $O(nk^2 \log^5 n)$ algorithm for finding the minimum such $\mathcal{T}$ that permits evacuation.

References

1 J. E. Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20(1):1–66, 1989. doi:10.1007/BF02216922.

2 G. P. Arumugam, J. Augustine, M. J. Golin, and P. Srikanthan. Optimal evacuation on dynamic paths with general capacities of edges. *Unpublished Manuscript*, 2015.

3 J. Chen, R. Rajaraman, and R. Sundaram. Meet and merge: Approximation algorithms for confluent flows. *Journal of Computer and System Sciences*, 72(3):468–489, 2006.

4 Jiangzhuo Chen, Robert D berg, László Lovász, Rajmohan Rajaraman, Ravi Sundaram, and Adrian Vetta. (Almost) Tight bounds and existence theorems for single-commodity confluent flows. *Journal of the ACM*, 54(4), jul 2007.

5 Daniel Dressler and Martin Strehler. Capacitated Confluent Flows: Complexity and Algorithms. In *7th International Conference on Algorithms and Complexity (CIAC'10)*, pages 347–358, 2010.

6 Lisa Fleischer and Martin Skutella. Quickest Flows Over Time. *SIAM Journal on Computing*, 36(6):1600–1630, January 2007.

7 L. R. Ford and D. R. Fulkerson. Constructing Maximal Dynamic Flows from Static Flows. *Operations Research*, 6(3):419–433, June 1958.

8 Greg N Frederickson. Parametric search and locating supply centers in trees. In *Proceedings of the Second Workshop on Algorithms and Data Structures (WADS'91)*, pages 299–319. Springer, 1991.

9 Michael R. Garey and David S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

10 Yuya Higashikawa, M. J. Golin, and Naoki Katoh. Minimax Regret Sink Location Problem in Dynamic Tree Networks with Uniform Capacity. In *The 8th International Workshop on Algorithms and Computation (WALCOM'2014)*, pages 125–137, 2014. doi:10.1007/978-3-319-04657-0_14.

11 Yuya Higashikawa, Mordecai J Golin, and Naoki Katoh. Multiple sink location problems in dynamic path networks. *Theoretical Computer Science*, 607:2–15, 2015.

12 B Hoppe and É Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000.

13 N. Kamiyama, N. Katoh, and A. Takizawa. Theoretical and Practical Issues of Evacuation Planning in Urban Areas. In *The Eighth Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA2007)*, pages 49–50, 2007.

14 Satoko Mamada and Kazuhisa Makino. An Evacuation Problem in Tree Dynamic Networks with Multiple Exits. In Tatsuo Arai, Shigeru Yamamoto, and Kazuhi Makino, editors, *Systems & Human Science-For Safety, Security, and Dependability; Selected Papers of the 1st International Symposium SSR2003*, pages 517–526. Elsevier B.V, 2005.

15 Satoko Mamada, Takeaki Uno, Kazuhisa Makino, and Satoru Fujishige. A tree partitioning problem arising from an evacuation problem in tree dynamic networks. *Journal of the Operations Research Society of Japan*, 48(3):196–206, 2005.

16 Satoko Mamada, Takeaki Uno, Kazuhisa Makino, and Satoru Fujishige. An $O(n \log^2 n)$algorithm for the optimal sink location problem in dynamic tree networks. *Discrete Applied Mathematics*, 154(2387-2401):251–264, 2006.

17 M. M. B. Pascoal, M. Eugénia V. Captivo, and J. C. N. Clímaco. A comprehensive survey on the quickest path problem. *Annals of Operations Research*, 147(1):5–21, August 2006.

18 F. Bruce Shepherd and Adrian Vetta. The Inapproximability of Maximum Single-Sink Unsplittable, Priority and Confluent Flow Problems, 2015. `arXiv:1504.0627`.

19 Martin Skutella. An introduction to network flows over time. In William Cook, László Lovász, and Jens Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, 2009.

Adaptivity vs. Postselection, and Hardness Amplification for Polynomial Approximation[*]

Lijie Chen

Tsinghua University, Beijing, China
wjmzbmr@gmail.com

Abstract

We study the following problem: with the power of postselection (classically or quantumly), what is your ability to answer adaptive queries to certain languages? More specifically, for what kind of computational classes $\mathcal{C}$, we have $\mathsf{P}^{\mathcal{C}}$ belongs to $\mathsf{PostBPP}$ or $\mathsf{PostBQP}$? While a complete answer to the above question seems impossible given the development of present computational complexity theory. We study the analogous question in *query complexity*, which sheds light on the limitation of *relativized* methods (the relativization barrier) to the above question.

Informally, we show that, for a partial function f, if there is no efficient[1] *small bounded-error* algorithm for f classically or quantumly, then there is no efficient *postselection bounded-error* algorithm to answer adaptive queries to f classically or quantumly. Our results imply a new proof for the classical oracle separation $\mathsf{P}^{\mathsf{NP}^{\mathcal{O}}} \not\subset \mathsf{PP}^{\mathcal{O}}$, which is arguably more elegant. They also lead to a new oracle separation $\mathsf{P}^{\mathsf{SZK}^{\mathcal{O}}} \not\subset \mathsf{PP}^{\mathcal{O}}$, which is close to an oracle separation between SZK and PP – an open problem in the field of oracle separations.

Our result also implies a hardness amplification construction for polynomial approximation: given a function f on n bits, we construct an adaptive-version of f, denoted by F, on $O(m \cdot n)$ bits, such that if f requires large degree to approximate to error $2/3$ in a certain one-sided sense, then F requires large degree to approximate even to error $1/2 - 2^{-m}$. Our construction achieves the same amplification in the work of Thaler (ICALP, 2016), by composing a function with $O(\log n)$ *deterministic query complexity*, which is in sharp contrast to all the previous results where the composing *amplifiers* are all hard functions in a certain sense.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases approximate degree, postselection, hardness amplification, adaptivity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.26

1 Introduction

1.1 Background

The idea of postselection has been surprisingly fruitful in theoretical computer science and quantum computing [3, 11, 6]. Philosophically, it addresses the following question: if you believe in the Many-worlds interpretation[2] and can condition on a rare event (implemented by killing yourself after observing the undesired outcomes), then what would you be able to compute in a reasonable amount of time? The complexity classes $\mathsf{PostBPP}$ [12] and $\mathsf{PostBQP}$ [1] are defined to represent the computational problems you can solve with the ability of postselection in a classical world or a quantum world.

[*] The full version is available at http://arxiv.org/abs/1606.04016.

[1] In the world of query complexity, being efficient means using $O(\text{polylog}(n))$ time.

[2] https://en.wikipedia.org/wiki/Many-worlds_interpretation

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 26; pp. 26:1–26:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, even with that seemingly omnipotent power of postselection, your computational power is still bounded. It is known that $\mathsf{PostBPP} \subseteq \mathsf{PH}$ [12], and (surprisingly) $\mathsf{PostBQP} = \mathsf{PP}$ [1]. Hence, it seems quite plausible that even with the postselection power, you are still not able to solve a PSPACE-complete problem, as it is widely believed that PH and PP are strictly contained in PSPACE.

Another more non-trivial (and perhaps unexpected) weakness of those postselection computation classes, is their inability to simulate *adaptive queries* to certain languages. For example, it is known that $\mathsf{P}^{\mathsf{NP}[O(\log n)]3}$ is contained in PostBPP [12], and this result relativizes. But there is an oracle separation between $\mathsf{P}^{\mathsf{NP}[\omega(\log n)]}$ and PostBQP [4]. In other words, there is no relativized PostBQP algorithm that can simulate $\omega(\log n)$ adaptive queries to a certain language in NP. In contrast, we know that $\mathsf{P}^{\|\mathsf{NP}} \subseteq \mathsf{PostBPP} \subseteq \mathsf{PP}$ [12], hence they are capable of simulating *non-adaptive* queries to NP.

Then a natural question follows:

► **Question 1.1.** *What is the limit of the abilities of these postselection classes on simulating adaptive queries to certain languages? More specifically, is there any characterization of the complexity class $\mathcal{C}$ such that $\mathsf{P}^{\mathcal{C}}$ is contained in PostBPP or PostBQP?*

Arguably, a complete answer to this problem seems not possible at the present time: even determining whether $\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{PP}$ is already extremely hard, as showing $\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{PP}$ probably requires some new non-relativized techniques, and proving $\mathsf{P}^{\mathsf{NP}} \not\subset \mathsf{PP}$ implies $\mathsf{PH} \not\subset \mathsf{PP}$, which is a long-standing open problem.

1.2 Relativization and the analogous question in query complexity

So in this paper, inspired by the oracle separation in [4], we study this problem from a relativization point of view. *Relativization*, or *oracle separations* are ultimately about the *query complexity*. Given a complexity class $\mathcal{C}$, there is a canonical way to define its analogue in query complexity: partial functions which are computable by a *non-uniform* $\mathcal{C}$ machine with polylog(n) queries to the input. For convenience, we will use $\mathcal{C}^{\mathsf{dt}}$ to denote the query complexity version of $\mathcal{C}$. We adopt the convention that $\mathcal{C}^{\mathsf{dt}}$ denotes the query analogue of $\mathcal{C}$, while $\mathcal{C}^{\mathsf{dt}}(f)$ denotes the $\mathcal{C}^{\mathsf{dt}}$ complexity of the partial function f.

For a partial function f, we use $\mathsf{len}(f)$ to denote its input length. We say a family of partial functions $\mathbf{f} \in \mathcal{C}^{\mathsf{dt}}$, if $\mathcal{C}^{\mathsf{dt}}(f) = O(\text{polylog}(\mathsf{len}(f)))$ for all $f \in \mathbf{f}$.

In order to study this question in the query complexity setting, given a partial function f, we need to define its adaptive version.

► **Definition 1.2** (Adaptive Construction). Given a function $f : D \to \{0,1\}$ with $D \subseteq \{0,1\}^M$ and an integer d, we define $\mathsf{Ada}_{f,d}$, its depth d adaptive version, as follows:

$$\mathsf{Ada}_{f,0} := f \quad \text{and} \quad \begin{aligned} &\mathsf{Ada}_{f,d} : D \times D_{d-1} \times D_{d-1} \to \{0,1\} \\ &\mathsf{Ada}_{f,d}(w,x,y) := \begin{cases} \mathsf{Ada}_{f,d-1}(x) & f(w) = 0 \\ \mathsf{Ada}_{f,d-1}(y) & f(w) = 1 \end{cases} \end{aligned}$$

where D_{d-1} denotes the domain of $\mathsf{Ada}_{f,d-1}$.

The input to $\mathsf{Ada}_{f,d}$ can be encoded as a string of length $(2^{d+1}-1) \cdot M$. Thus, $\mathsf{Ada}_{f,d}$ is a partial function from $D^{(2^{d+1}-1)} \to \{0,1\}$.

[3] $O(\log n)$ stands for the P algorithm can only make $O(\log n)$ queries to the oracle.

Then, given a family of partial function $\mathbf{f}$, we define $\mathsf{Ada}_{\mathbf{f}} := \{\mathsf{Ada}_{f,d} \mid f \in \mathbf{f}, d \in \mathbb{N}\}$.

Notice that when you have the ability to adaptively solve $d+1$ queries to f (or with high probability), then it is easy to solve $\mathsf{Ada}_{f,d}$. Conversely, in order to solve $\mathsf{Ada}_{f,d}$, you need to be able to adaptively answer $d+1$ questions to f, as even *knowing what is the right i^{th} question to answer* requires you to correctly answer all the previous $i-1$ questions.

Now, everything is ready for us to state the analogous question in query complexity.

▶ **Question 1.3.** *What is the characterization of the partial functions family* $\mathbf{f}$ *such that* $\mathsf{Ada}_{\mathbf{f}} \in \mathsf{PostBPP}^{dt}$ *(*$\mathsf{PostBQP}^{dt}$*)?*

There are at least two reasons to study Question 1.3. First, it is an interesting question itself in query complexity. Second, an answer to Question 1.3 also completely characterizes the limitation on the *relativized techniques* for answering Question 1.1, i.e., the limitation of relativized methods for simulating *adaptive queries* to certain complexity classes with the power of postselection.

This paper provides some interesting results toward resolving Question 1.3.

1.3 Our results

Despite that we are not able to give a complete answer to Question 1.3. We provide some interesting lower bounds showing that certain functions' adaptive versions are hard for these postselection classes.

Formally, we prove the following two theorems.

▶ **Theorem 1.4** (Quantum Case). *For a family of partial function* $\mathbf{f}$*,* $\mathsf{Ada}_{\mathbf{f}} \notin \mathsf{PostBQP}^{dt}(\mathsf{PP}^{dt})$ *if* $\mathbf{f} \notin \mathsf{SBQP}^{dt} \cap \mathsf{coSBQP}^{dt}$.

▶ **Theorem 1.5** (Classical Case). *For a family of partial function* $\mathbf{f}$*,* $\mathsf{Ada}_{\mathbf{f}} \notin \mathsf{PostBPP}^{dt}$ *if* $\mathbf{f} \notin \mathsf{SBP}^{dt} \cap \mathsf{coSBP}^{dt}$.

Roughly speaking, SBP is a relaxation of BPP, it is the set of languages L such that there exists a BPP machine M, which accepts x with probability $\geq 2\alpha$ if $x \in L$; and with probability $\leq \alpha$ if $x \notin L$ for a positive real number α. And SBQP is the quantum analogue of SBP, where you are allowed to use a polynomial time quantum algorithm instead.[4]

Our theorems show that, for a partial function f, if there is no efficient classical (quantum) algorithm which accepts all the 1-inputs with a slightly better chance than all the 0-inputs, then there is no efficient PostBPP (PostBQP) algorithm that can answer adaptive queries to f.

In fact, we prove the following two quantitatively tighter theorems, from which Theorem 1.4 and Theorem 1.5 follows easily.

▶ **Theorem 1.6.** *Let f be a partial function and T be a non-negative integer. Suppose* $\widehat{\deg}_+(f) > T$ *or* $\widehat{\deg}_-(f) > T$*, then we have*

$$\mathsf{PP}^{dt}(\mathsf{Ada}_{f,d}) > \min(T/4, 2^{d-1}).^{5}$$

▶ **Theorem 1.7.** *Let* $f : D \to \{0,1\}$ *with* $D \subseteq \{0,1\}^M$ *be a partial function and d be a non-negative integer. Suppose* $\mathsf{SBP}^{dt}(f) > T$ *or* $\mathsf{coSBP}^{dt}(f) > T$*, then we have*

$$\mathsf{PostBPP}^{dt}(\mathsf{Ada}_{f,d}) > \min(T/5, (2^d - 1)/5).$$

[4] For the formal definitions of SBP, PostBPP, PostBQP, SBQP and their equivalents in query complexity, see the preliminaries.

1.4 Applications in oracle separations

Our results have several applications in oracle separations.

- A new proof for $\mathsf{P}^{\mathsf{NP}^{\mathcal{O}}} \not\subset \mathsf{PP}^{\mathcal{O}}$:
  We prove that $\mathsf{SBQP}^{\mathsf{dt}}(f)$ is indeed equivalent to *one-sided low-weight approximate degree*, denoted by $\widehat{\deg}_{+}(f)$ (cf. Definition 2.8), which is lower bounded by one-sided approximate degree $\deg_{+}(f)$ (cf. Definition 1.8).
  Using the fact that $\deg_{+}(\mathsf{AND}_n) \geq \Omega(\sqrt{n})$, Theorem 1.4 implies that $\mathsf{Ada}_{\mathsf{AND}} \not\subset \mathsf{PP}^{\mathsf{dt}}$, yielding a simpler proof for the classical oracle separation between P^{NP} and PP in [4].
  Our proof is arguably simpler and more elegant. Also, unlike the seemingly artificial problem ODD-MAX-BIT[6] in [4], $\mathsf{Ada}_{\mathsf{AND}}$ looks like a more natural hard problem in P^{NP}.
- The new oracle separation $\mathsf{P}^{\mathsf{SZK}^{\mathcal{O}}} \not\subset \mathsf{PP}^{\mathcal{O}}$:
  Since the *Permutation Testing Problem*, denoted by PTP_n (see Problem 2.12 for a formal definition), satisfies $\deg_{+}(\mathsf{PTP}_n) \geq \Omega(n^{1/3})$ and has a $\log(n)$-time SZK protocol. Theorem 1.4 implies that $\mathsf{Ada}_{\mathsf{PTP}} \not\subset \mathsf{PP}^{\mathsf{dt}}$, which in turn shows an oracle separation between $\mathsf{P}^{\mathsf{SZK}}$ and PP.
  It has been an open problem [2] that whether there exists an oracle separation between SZK and PP, our result is pretty close to an affirmative answer to that.[7]
  Also, note that $\mathsf{P}^{\mathsf{SZK}} \subseteq \mathsf{P}^{\mathsf{AM} \cap \mathsf{coAM}} = \mathsf{AM} \cap \mathsf{coAM}$, so our result improves on the oracle separation between $\mathsf{AM} \cap \mathsf{coAM}$ and PP by Vereschchagin [18].

1.5 Applications in hardness amplification for polynomial approximation

Our construction also leads to a hardness amplification theorem for polynomial approximation. In order to state our result, we need to introduce the definition of two approximate degrees first.

▶ **Definition 1.8.** The ϵ-approximate degree of a partial function of $f : D \to \{0,1\}$, denoted as $\widetilde{\deg}_{\epsilon}(f)$, is the least degree of a real polynomial p such that $|p(x) - f(x)| \leq \epsilon$ when $x \in D$, and $|p(x)| \leq 1 + \epsilon$ when $x \notin D$.

We say a polynomial p one-sided ϵ-approximates a partial Boolean function f, if $p(x) \in [0, \epsilon]$ when $f(x) = 0$, and $p(x) \geq 1$ when $f(x) = 1$.[8] Then the one-sided ϵ-approximate degree of a partial function f, denoted by $\deg_{+}^{\epsilon}(f)$, is the minimum degree of a polynomial one-sided ϵ-approximating f.

Now we are in a position to state our amplification theorem.

▶ **Theorem 1.9.** *Let f be a partial function such that $\deg_{+}^{2/3}(f) > T$ and d be a positive integer, we have $\widetilde{\deg}_{\epsilon}(\mathsf{Ada}_{f,d}) > T$ for $\epsilon = 0.5 - 2^{-2^{d}+1}$.*

That is, given a function with high one-sided approximate degree for an error constant bounded away from 1, it can be transformed to a function with high approximate degree even for ϵ *doubly exponentially* close to $1/2$ in d.[9]

[6] Given a binary input x, it asks whether the rightest 1 in x is in an odd position.

[7] Partially inspired by this work, an oracle separation between SZK and PP (in fact, UPP) has been constructed in a very recent work of Bouland, Chen, Holden, Thaler and Vasudevan [5], thus resolved this open problem.

[8] Our definition of one-sided approximation is slightly different from the standard one [15, 8, 16], but it greatly simplifies several discussions in our paper, and they are clearly equivalent up to a linear transformation in ϵ.

[9] Which is *single exponential* in the input length of the *amplifier* AdaQ, see the discussion below.

Comparison with previous amplification results

There have been a lot of research interest in hardness amplification for polynomial approximation, many amplification results are achieved through *function composition* [9, 15, 17]. We use $f \circ g$ to denote the block composition of f and g, i.e. $f(g, g, \ldots, g)$.

Our result can also be viewed as one of them. Let $\mathsf{AdaQ}_d := \mathsf{Ada}_{\mathsf{id},d}$, where id is just the identity function from $\{0,1\}$ to $\{0,1\}$. Then we can see that in fact $\mathsf{Ada}_{f,d}$ is equivalent to $\mathsf{AdaQ}_d \circ f$. Let $n = 2^{d+1} - 1$, which is the input length of AdaQ_d.

However, all the previous amplification results are achieved by letting the *amplifier* f to be a *hard* function. We list all these results for an easy comparison.

- In the work of Bun and Tahler [9], they showed that for a function g such that $\deg_+(g) > T$, $\widetilde{\deg}_\epsilon(\mathsf{OR}_n \circ g) > T$ for $\epsilon = 1/2 - 2^{-\Omega(n)}$. This is further improved by Sherstov [15] to that $\deg_\pm(\mathsf{OR}_n \circ g) = \Omega(\min(n, T))$. Here, the amplifier OR_n is a hard function in the sense that $\deg_+(\mathsf{OR}_n) \ge \Omega(\sqrt{n})$ [14].
- In [17], Thaler showed that for a function g such that $\deg_+(g) > T$, $\widetilde{\deg}_\epsilon(\mathsf{ODD\text{-}MAX\text{-}BIT}_n \circ g) > T$ for $\epsilon = 1/2 - 2^{-\Omega(n)}$.[10] In this case, the amplifier $\mathsf{ODD\text{-}MAX\text{-}BIT}_n$ is even harder in the sense that it has a $\mathsf{PP}^{\mathsf{dt}}$ query complexity of $\Omega(\sqrt[3]{n})$ [4].
- Moreover, it is easy to see that the *randomized query complexity* of both OR_n and $\mathsf{ODD\text{-}MAX\text{-}BIT}_n$ is the maximum possible $\Omega(n)$.

In contrast, our amplifier AdaQ, is *extremely* simple – it has a *deterministic query complexity* of $O(\log n)$![11]

This is a rather surprising feature of our result. That means AdaQ also has an *exact degree* of $O(\log n)$. Intuitively, composing with such a simple and innocent function seems would not affect the hardness of the resulting function. Our result severely contradicts this intuition. But from the view point of Theorem 1.4, composing with AdaQ indeed "adaptivize" the function, makes it hard for $\mathsf{PostBQP}$ algorithms, which is in turn closely connected to PP algorithms and therefore polynomial approximate degree. So this result is arguably natural under that perspective, which illustrates a recurring theme in TCS: a new perspective can lead to some unexpected results.

1.6 Paper organization

In Section 2 we introduce some preliminaries, due to the space constraints, some of the formal definitions of those partial function classes in query complexity can be found in the full version. We prove Theorem 1.4 and Theorem 1.6 in Section 3, and defer the proof for Theorem 1.5 and Theorem 1.7 to the full version. Theorem 1.9 is proved in Section 3.4. And we provide formal proofs for the two oracle separation results in the full version.

2 Preliminaries

2.1 Decision trees and quantum query algorithms

A (randomized) decision tree is the analogue of a deterministic (randomized) algorithm in the query complexity world, and a quantum query algorithm is the analogue of a quantum algorithm. See [7] for a nice survey on query complexity.

[10] This construction is further improved in a very recent work [10] by Bun and Thaler, with a more sophisticated construction which does not follow the composition paradigm.

[11] A simple $O(\log n)$-query algorithm just follows from the definition.

Let $\mathcal{T}$ be a randomized decision tree, we use $\mathcal{C}(\mathcal{T})$ to denote the maximum number of queries incurred by $\mathcal{T}$ in the worst case[12]. Let $\mathcal{Q}$ be a quantum query algorithm, we use $\mathcal{C}(\mathcal{Q})$ to denote the number of queries taken by $\mathcal{Q}$.

We assume a randomized decision tree $\mathcal{T}$ (or a quantum query algorithm $\mathcal{Q}$) outputs a result in $\{0,1\}$, and we use $\mathcal{T}(x)$ ($\mathcal{Q}(x)$) to denote the (random) output of $\mathcal{T}$ ($\mathcal{Q}$) given an input x.

2.2 Complexity classes and their query complexity analogues

We assume familiarity with some standard complexity classes like PP. Due to space constraint, we only introduce the most relevant classes $\mathsf{A0PP}^{\mathsf{dt}}$ and $\mathsf{PP}^{\mathsf{dt}}$ here, and defer the formal definitions of the partial function complexity classes $\mathsf{SBP}^{\mathsf{dt}}$, $\mathsf{SBQP}^{\mathsf{dt}}$, $\mathsf{PostBPP}^{\mathsf{dt}}$ and $\mathsf{PostBQP}^{\mathsf{dt}}$ to the full version.

Recall that $\mathcal{C}^{\mathsf{dt}}$ is the set of the partial function family $\mathbf{f}$ with $\mathcal{C}^{\mathsf{dt}}(f) = O(\text{polylog}(\mathsf{len}(f)))$ for all $f \in \mathbf{f}$, hence we only need to define $\mathcal{C}^{\mathsf{dt}}(f)$ for a partial function f.

$\mathsf{PP}^{\mathsf{dt}}$

We first define $\mathsf{PP}^{\mathsf{dt}}(f)$.

▶ **Definition 2.1.** Let $f : D \to \{0,1\}$ with $D \subseteq \{0,1\}^M$ be a partial function. Let $\mathcal{T}$ be a randomized decision tree which computes f with a probability better than 1/2. Let α be the maximum real number such that

$$\Pr[\mathcal{T}(x) = f(x)] \geq \frac{1}{2} + \alpha$$

for all $x \in D$.

Then we define $\mathsf{PP}^{\mathsf{dt}}(\mathcal{T}; f) := C(\mathcal{T}) + \log_2(1/\alpha)$, and $\mathsf{PP}^{\mathsf{dt}}(f)$ as the minimum of $\mathsf{PP}^{\mathsf{dt}}(\mathcal{T}; f)$ over all $\mathcal{T}$ computing f with a probability better than 1/2.

A0PP and $\mathsf{A0PP}^{\mathsf{dt}}$

In this subsection we review the definition of $\mathsf{A0PP}$, and define its analogue in query complexity. There are several equivalent definitions for $\mathsf{A0PP}$, we choose the most convenient one here.

▶ **Definition 2.2.** $\mathsf{A0PP}$ (defined by Vyalyi [19]) is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a BPP machine M and a polynomial p, such that for all inputs x:

(i) $x \in L \Longrightarrow \Pr[M(x) \text{ accepts}] \geq \frac{1}{2} + 2^{-p(|x|)}$.

(ii) $x \notin L \Longrightarrow \Pr[M(x) \text{accepts}] \in \left[\frac{1}{2}, \frac{1}{2} + 2^{-p(|x|)-1}\right]$.

▶ **Definition 2.3.** Let $f : D \to \{0,1\}$ with $D \subseteq \{0,1\}^M$ be a partial function. We say a randomized decision tree $\mathcal{T}$ $\mathsf{A0PP}$-computes f if there is a real number $\alpha > 0$ such that

- $\Pr[\mathcal{T}(x) = 1] \geq 1/2 + 2\alpha$ when $f(x) = 1$.
- $\Pr[\mathcal{T}(x) = 1] \in [1/2, 1/2 + \alpha]$ when $f(x) = 0$.

Fix a $\mathcal{T}$ $\mathsf{A0PP}$-computing f, let α be the maximum real number satisfying above conditions. Then we define $\mathsf{A0PP}^{\mathsf{dt}}(\mathcal{T}; f) = C(\mathcal{T}) + \log_2(1/\alpha)$ for $\mathcal{T}$ $\mathsf{A0PP}$-computing f and $\mathsf{A0PP}^{\mathsf{dt}}(f)$ as the minimum of $\mathsf{A0PP}^{\mathsf{dt}}(\mathcal{T}; f)$ over all $\mathcal{T}$ $\mathsf{A0PP}^{\mathsf{dt}}$-computing f. And we simply let $\mathsf{coA0PP}^{\mathsf{dt}}(f) := \mathsf{A0PP}^{\mathsf{dt}}(\neg f)$.

[12] i.e. the maximum height of a decision tree in the support of $\mathcal{T}$

Two relativized facts

We also introduce two important relativized results here. In [1], Aaronson showed that PostBQP is indeed PP in disguise.

▶ **Theorem 2.4** ([1]). *PostBQP = PP.*

And in [13], Kuperberg showed that SBQP is in fact equal to A0PP.

▶ **Theorem 2.5** ([13]). *SBQP = A0PP.*

These two theorems relativize, hence we have the following corollaries.

▶ **Corollary 2.6.** $\mathit{SBQP}^{\mathit{dt}} = \mathit{A0PP}^{\mathit{dt}}$.

▶ **Corollary 2.7.** $\mathit{PostBQP}^{\mathit{dt}} = \mathit{PP}^{\mathit{dt}}$.

2.3 Low-weighted one-sided approximate degree

In this subsection, we introduce a new notion of one-sided approximate degree, which is closely connected to $\mathsf{A0PP}^{\mathsf{dt}}(f)$.

▶ **Definition 2.8.** Write a polynomial $p(x) := \sum_{i=1}^{m} a_i \cdot M_i(x)$ as a sum of monomials, we define $\mathsf{weight}(p) := \sum_{i=1}^{m} |a_i|$. The one-sided low-weight ϵ-approximate degree of a partial function f denoted by $\widehat{\deg}_+^{\epsilon}(f)$, is defined by

$$\widehat{\deg}_+^{\epsilon}(f) := \min_p \max\{\deg(p), \log_2(\mathsf{weight}(p))\},$$

where p goes over all polynomials which one-sided ϵ-approximates f.[13]

We simply let $\widehat{\deg}_-^{\epsilon}(f) := \widehat{\deg}_+^{\epsilon}(\neg f)$. We also define $\widehat{\deg}_+(f)$ as $\widehat{\deg}_+^{1/2}(f)$. $\widehat{\deg}_-$ is defined similarly.

Clearly $\widehat{\deg}_+^{\epsilon}(f) \geq \deg_+^{\epsilon}(f)$. And the choice of constant 1/2 is arbitrary, as we can reduce the approximation error by the following lemma.

▶ **Lemma 2.9.** *For any* $0 < \epsilon_1 < \epsilon_2 < 1$, $\widehat{\deg}_+^{\epsilon_1}(f) \leq \left\lceil \frac{\ln \epsilon_1^{-1}}{\ln \epsilon_2^{-1}} \right\rceil \cdot \widehat{\deg}_+^{\epsilon_2}(f)$.

Proof. We can just take the $\left\lceil \frac{\ln \epsilon_1^{-1}}{\ln \epsilon_2^{-1}} \right\rceil^{th}$ power of the polynomial corresponding to $\widehat{\deg}_+^{\epsilon_2}(f)$. ◀

We show that $\widehat{\deg}_+(f)$ is in fact equivalent to $\mathsf{A0PP}^{\mathsf{dt}}(f)$ up to a constant factor.

▶ **Theorem 2.10.** *Let f be a partial function, then*

$$\widehat{\deg}_+(f) \leq 2 \cdot \mathit{A0PP}^{\mathit{dt}}(f) \text{ and } \mathit{A0PP}^{\mathit{dt}}(f) \leq 2 \cdot \widehat{\deg}_+(f) + 2.$$

The proof is based on a simple transformation between a decision tree and the polynomial representing it, we defer the details to the full version.

And the following corollary follows from the definitions.

▶ **Corollary 2.11.** *Let f be a partial function, then*

$$\widehat{\deg}_-(f) \leq 2 \cdot \mathit{coA0PP}^{\mathit{dt}}(f) \text{ and } \mathit{coA0PP}^{\mathit{dt}}(f) \leq 2 \cdot \widehat{\deg}_-(f) + 2.$$

[13] Recall that a polynomial p one-sided ϵ-approximates a partial Boolean function f, if $p(x) \in [0, \epsilon]$ when $f(x) = 0$, and $p(x) \geq 1$ when $f(x) = 1$ as in Definition 1.8.

2.4 The permutation testing problem

Finally, we introduce the permutation testing problem.

▶ **Problem 2.12** (Permutation Testing Problem or PTP). *Given black-box access to a function $f : [n] \to [n]$, and promised that either*

(i) *f is a permutation (i.e., is one-to-one), or*

(ii) *f differs from every permutation on at least $n/8$ coordinates.*

The problem is to accept if (i) holds and reject if (ii) holds.

Assume n is a power of 2, we use PTP_n to denote the Permutation Testing Problem on functions from $[n] \to [n]$. PTP_n can be viewed as a partial function $D \to \{0,1\}$ with $D \subseteq \{0,1\}^{n \cdot \log_2 n}$.

3 Proof for the quantum case

In this section we prove Theorem 1.4 and Theorem 1.6.

Let $f : D \to \{0,1\}$ with $D \subseteq \{0,1\}^M$ be a partial function, we say a polynomial p on M variables *computes* f, if $p(x) \geq 1$ whenever $f(x) = 1$, and $p(x) \leq -1$ whenever $f(x) = 0$.

3.1 Existence of the hard distributions

In this subsection we show that if $\widetilde{\deg}_+(f)$ is large, there must exist some input distributions witness this fact in a certain sense.

▶ **Lemma 3.1.** *Let f be a partial function and T be a non-negative integer. For convenience, we say a polynomial p is* valid*, if it is of degree at most T, and satisfies $\mathsf{weight}(p) \leq 2^T$.*

If $\widetilde{\deg}_+^{2/3}(f) > T$, there exist two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ supported on $f^{-1}(0)$ and $f^{-1}(1)$ respectively, such that

$$-p(\mathcal{D}_0) > 2 \cdot p(\mathcal{D}_1),$$

where $p(\mathcal{D}) = \mathbb{E}_{x \sim \mathcal{D}}[p(x)]$, for all valid polynomial p computing f.

In order to establish the above lemma, we need the following simple lemma.

▶ **Lemma 3.2.** *For any valid polynomial p computing f, if $\widetilde{\deg}_+^{2/3}(f) > T$, then there exist $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$ such that $-p(x) > 2 \cdot p(y)$.*

The proof is based on a simple calculation, the details can be found in the full version.

Then we prove Lemma 3.1.

Proof of Lemma 3.1. By Lemma 3.2, we have

$$\min_{p} \max_{(x,y) \in f^0 \times f^1} -p(x) - 2 \cdot p(y) > 0,$$

where p is a valid polynomial which computes f, $f^0 := f^{-1}(0)$ and $f^1 := f^{-1}(1)$. By the minimax theorem, and note that all the valid polynomials form a *compact convex set*, there exists a distribution $\mathcal{D}_{xy}$ on $f^0 \times f^1$ such that for any valid polynomial p computing f, we have

$$\mathbb{E}_{(x,y) \sim \mathcal{D}_{xy}}[-p(x) - 2 \cdot p(y)] > 0.$$

Then we simply let $\mathcal{D}_0$ ($\mathcal{D}_1$) be the marginal distribution of $\mathcal{D}_{xy}$ on f^0 (f^1), which completes the proof. ◀

And the following corollary follows by the definition of $\widehat{\deg}_-$.

▶ **Corollary 3.3.** *Let f be a partial function and T be a non-negative integer, if $\widehat{\deg}_-^{2/3}(f) > T$, then there exist two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ supported on $f^{-1}(0)$ and $f^{-1}(1)$ respectively, such that for all valid polynomial p computing f,*

$$p(\mathcal{D}_1) > -2 \cdot p(\mathcal{D}_0).$$

3.2 Proof for Theorem 1.4 and Theorem 1.6

We first show Theorem 1.6 implies Theorem 1.4.

Proof of Theorem 1.4. Suppose $\mathbf{f} \notin \mathsf{SBQP}^{\mathsf{dt}}$, the case that $\mathbf{f} \notin \mathsf{coSBQP}^{\mathsf{dt}}$ is similar.

By Corollary 2.6 and Theorem 2.10, there exists a sequence of function $\{f_i\}_{i=1}^{\infty} \subseteq \mathbf{f}$ such that $\widehat{\deg}_+(f_i) > \log(\mathsf{len}(f_i))^i$. Then we consider the partial function sequence $\{\mathsf{Ada}_{f_i,\lceil\log(\mathsf{len}(f_i))\rceil}\}_{i=1}^{\infty} \subseteq \mathsf{Ada}_{\mathbf{f}}$.

By Theorem 1.6, we have

$$\mathsf{PP}^{\mathsf{dt}}(\mathsf{Ada}_{f_i,\lceil\log(\mathsf{len}(f_i))\rceil}) > \min(\log(\mathsf{len}(f_i))^i/4, \mathsf{len}(f_i)/2).$$

Note that $\mathsf{len}(\mathsf{Ada}_{f_i,\lceil\log(\mathsf{len}(f_i))\rceil}) \le 2 \cdot \mathsf{len}(f_i)^2$, we can see $\mathsf{Ada}_{\mathbf{f}} \notin \mathsf{PP}^{\mathsf{dt}}$ due to the above partial function sequence. ◀

Now, we are going to prove Theorem 1.6. We begin by introducing some consequences of a function having low $\mathsf{PP}^{\mathsf{dt}}$ complexity.

▶ **Lemma 3.4.** *Let f be a partial function, T be a positive integer. Suppose $\mathsf{PP}^{\mathsf{dt}}(f) \le T$, then there exists a degree T-polynomial p computing f and satisfying $\mathsf{weight}(p) \le 2^{2T}$.*

The proof is based on a direct analysis of the polynomial representing the decision tree for $\mathsf{PP}^{\mathsf{dt}}(f)$, we defer the details to the full version.

Our proof relies on the following two key lemmas.

▶ **Lemma 3.5.** *Let f be a partial function with $\widehat{\deg}_+^{2/3}(f) > T$. Then for each integer d, there exist two distributions $\mathcal{D}_1^d$ and $\mathcal{D}_0^d$ supported on $\mathsf{Ada}_{f,d}^{-1}(1)$ and $\mathsf{Ada}_{f,d}^{-1}(0)$ respectively, such that $-p(\mathcal{D}_0) > 2^{2^d} \cdot p(\mathcal{D}_1)$ for any degree-T polynomial p computing $\mathsf{Ada}_{f,d}$ and satisfying $\mathsf{weight}(p) \le 2^T$.*

▶ **Lemma 3.6.** *Let f be a partial function with $\widehat{\deg}_-^{2/3}(f) > T$. Then for each integer d, there exist two distributions $\mathcal{D}_1^d$ and $\mathcal{D}_0^d$ supported on $\mathsf{Ada}_{f,d}^{-1}(1)$ and $\mathsf{Ada}_{f,d}^{-1}(0)$ respectively, such that $p(\mathcal{D}_1) > -2^{2^d} \cdot p(\mathcal{D}_0)$ for any degree-T polynomial p computing $\mathsf{Ada}_{f,d}$ and satisfying $\mathsf{weight}(p) \le 2^T$.*

We first show these two lemmas imply Theorem 1.6 in a straightforward way.

Proof of Theorem 1.6. We prove the case when $\widehat{\deg}_+(f) > T$ first.

Otherwise, suppose $\mathsf{PP}^{\mathsf{dt}}(\mathsf{Ada}_{f,d}) \le \min(T/4, 2^{d-1})$. By Lemma 3.4, we have a degree-$T/4$ polynomial p computing $\mathsf{Ada}_{f,d}$ with $\mathsf{weight}(p) \le \min(2^{T/2}, 2^{2^d})$. From Lemma 2.9, $\widehat{\deg}_+(f) = \widehat{\deg}_+^{1/2}(f) \le 2 \cdot \widehat{\deg}_+^{2/3}(f)$, hence $\widehat{\deg}_+^{2/3}(f) > T/2$. Then by Lemma 3.5, there exist two distributions $\mathcal{D}_1^d$ and $\mathcal{D}_0^d$ supported on $\mathsf{Ada}_{f,d}^{-1}(1)$ and $\mathsf{Ada}_{f,d}^{-1}(0)$ respectively, such that $-p(\mathcal{D}_0) > 2^{2^d} \cdot p(\mathcal{D}_1)$ as p is of degree at most $T/4$ and satisfies $\mathsf{weight}(p) \le 2^{T/2}$.

But this means that $-p(\mathcal{D}_0) > 2^{2^d}$, which implies there exists an x such that $p(x) < -2^{2^d}$, therefore $\mathsf{weight}(p) > 2^{2^d}$, contradiction.

The case when $\widehat{\deg}_-(f) > T$ follows exactly in the same way by using Lemma 3.6 instead of Lemma 3.5. ◀

3.3 Proof for Lemma 3.5

Finally we prove Lemma 3.5. The proof for Lemma 3.6 is completely symmetric using Corollary 3.3 instead of Lemma 3.1.

Proof of Lemma 3.5. Recall that a polynomial p is valid, if it is of degree at most T, and satisfies $\mathsf{weight}(p) \leq 2^T$. Let $f_d := \mathsf{Ada}_{f,d}$ and D_d be the domain of f_d. We are going to construct these distributions $\mathcal{D}_0^d$'s and $\mathcal{D}_1^d$'s by an elegant induction.

Construction of $\mathcal{D}_0$ and $\mathcal{D}_1$ from Lemma 3.1. By Lemma 3.1 there exist two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ supported on $f^{-1}(0)$ and $f^{-1}(1)$ respectively, such that $-p(\mathcal{D}_0) > 2 \cdot p(\mathcal{D}_1)$ for all valid polynomial p computing f.

The base case: construction of $\mathcal{D}_0^0$ and $\mathcal{D}_1^0$. For the base case $d = 0$, as f_0 is just f, we simply set $\mathcal{D}_0^0 = \mathcal{D}_0$ and $\mathcal{D}_1^0 = \mathcal{D}_1$. Then for all valid polynomial p computing f_0, we have $-p(\mathcal{D}_0^0) > 2 \cdot p(\mathcal{D}_1^0) = 2^{2^0} \cdot p(\mathcal{D}_1^0)$.

Construction of $\mathcal{D}_0^d$ and $\mathcal{D}_1^d$ for $d > 0$. When $d > 0$, suppose that we have already constructed the required distributions $\mathcal{D}_0^{d-1}$ and $\mathcal{D}_1^{d-1}$ for f_{d-1}. Decompose the input to f_d as $(w, x, y) \in D \times D_{d-1} \times D_{d-1}$ as in the definition, we claim that

$$\mathcal{D}_0^d = (\mathcal{D}_0, \mathcal{D}_0^{d-1}, \mathcal{D}_0^{d-1})^{14} \text{ and } \mathcal{D}_1^d = (\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_1^{d-1})$$

satisfy our conditions.

Analysis of $\mathcal{D}_0^d$ and $\mathcal{D}_1^d$. Note that D_i^d is supported on $f_d^{-1}(i)$ for $i \in \{0, 1\}$ from the definition. Let $p(w, x, y)$ be a valid polynomial computing f_d. We set

$$p(\mathcal{D}_w, \mathcal{D}_x, \mathcal{D}_y) := \mathbb{E}_{w \sim \mathcal{D}_w, x \sim \mathcal{D}_x, y \sim \mathcal{D}_y}[p(w, x, y)]$$

for simplicity, where $\mathcal{D}_w, \mathcal{D}_x, \mathcal{D}_y$ are distributions over D, D_{d-1}, D_{d-1} respectively.

Then we have to verify that for all valid polynomial p computing f_d,

$$-p(\mathcal{D}_0^d) = -p(\mathcal{D}_0, \mathcal{D}_0^{d-1}, \mathcal{D}_0^{d-1}) > 2^{2^d} \cdot p(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_1^{d-1}) = 2^{2^d} \cdot p(\mathcal{D}_1^d).$$

We proceed by incrementally changing $(\mathcal{D}_0, \mathcal{D}_0^{d-1}, \mathcal{D}_0^{d-1})$ into $(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_1^{d-1})$, and establish inequalities along the way.

Step 1: $(\mathcal{D}_0, \mathcal{D}_0^{d-1}, \mathcal{D}_0^{d-1}) \Rightarrow (\mathcal{D}_0, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1})$. By the definition, we can see that for any fixed $W \in \mathbf{support}(\mathcal{D}_0)$ and $Y \in \mathbf{support}(\mathcal{D}_0^{d-1})$, the polynomial in x defined by $p_L(x) := p(W, x, Y)$ is a valid polynomial computing f_{d-1}, hence $-p_L(\mathcal{D}_0^{d-1}) > 2^{2^{d-1}} \cdot p_L(\mathcal{D}_1^{d-1})$. By linearity, we have

$$-p(\mathcal{D}_0, \mathcal{D}_0^{d-1}, \mathcal{D}_0^{d-1}) > 2^{2^{d-1}} \cdot p(\mathcal{D}_0, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}).$$

Step 2: $(\mathcal{D}_0, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}) \Rightarrow (\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1})$. Similarly, for any fixed $X \in \mathbf{support}(\mathcal{D}_1^{d-1})$ and $Y \in \mathbf{support}(\mathcal{D}_0^{d-1})$, by the definition, we can see that the polynomial in w defined by $p_M(w) := -p(w, X, Y)$ is a valid polynomial computing f, hence $-p_M(\mathcal{D}_0) > 2 \cdot p_M(\mathcal{D}_1)$. Again by linearity, we have

$$p(\mathcal{D}_0, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}) > -2 \cdot p(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}) > -p(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}).$$

Step 3: $(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}) \Rightarrow (\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_1^{d-1})$. Finally, for any fixed $W \in \mathbf{support}(\mathcal{D}_1)$ and $X \in \mathbf{support}(\mathcal{D}_1^{d-1})$, the polynomial in y defined by $p_R(y) := p(W, X, y)$ is a polynomial computing f_{d-1}, hence $-p_R(\mathcal{D}_0^{d-1}) > 2^{2^{d-1}} \cdot p_R(\mathcal{D}_1^{d-1})$. By linearity, we have

$$-p(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_0^{d-1}) > 2^{2^{d-1}} \cdot p(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_1^{d-1}).$$

Putting the above three inequalities together, we have

$$-p(\mathcal{D}_0^d) = -p(\mathcal{D}_0, \mathcal{D}_0^{d-1}, \mathcal{D}_0^{d-1}) > 2^{2^d} \cdot p(\mathcal{D}_1, \mathcal{D}_1^{d-1}, \mathcal{D}_1^{d-1}) = 2^{2^d} \cdot p(\mathcal{D}_1^d).$$

This completes the proof. ◀

3.4 Application in hardness amplification for polynomial approximation

In this subsection, we slightly adapt the above proof in order to show Theorem 1.9.

For a polynomial p on n variables, let $\|p\|_\infty := \max_{x \in \{0,1\}^n} |p(x)|$. Lemma 3.5 shows that, fix a partial function f with $\widehat{\deg}_+(f) > T$, then for any polynomial computing $\mathsf{Ada}_{f,d}$ with $\mathsf{weight}(p) \le 2^T$, we must have $\|p\|_\infty > 2^{2^d}$. The restriction on $\mathsf{weight}(p)$ is essential for us to establish the connection between $\mathsf{A0PP}^{\mathsf{dt}}$ and $\widehat{\deg}_+$, but it becomes troublesome when it comes to proving a hardness amplification result.

Luckily, we can get rid of the restriction on $\mathsf{weight}(p)$ by making a stronger assumption that $\deg_+(f) > T$. Formally, we have the following analogous lemma for Lemma 3.5.

▶ **Lemma 3.7.** *Let f be a partial function with $\deg_+^{2/3}(f) > T$. Then for each integer d, there exist two distributions $\mathcal{D}_1^d$ and $\mathcal{D}_0^d$ supported on $\mathsf{Ada}_{f,d}^{-1}(1)$ and $\mathsf{Ada}_{f,d}^{-1}(0)$ respectively, such that for any degree-T polynomial p computing $\mathsf{Ada}_{f,d}$, $-p(\mathcal{D}_0^d) > 2^{2^d} \cdot p(\mathcal{D}_1^d)$ and consequently $\|p\|_{+\infty} > 2^{2^d}$.*

Proof. Using nearly the same proof for Lemma 3.1, we can show that for a partial function f, if $\deg_+^{2/3}(f) > T$, there exist two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ supported on $f^{-1}(0)$ and $f^{-1}(1)$ respectively, such that $-p(\mathcal{D}_0) > 2 \cdot p(\mathcal{D}_1)$ for all degree-T polynomial p computing f. Then we can proceed exactly as in the proof for Lemma 3.5 to get the desired distributions. ◀

Finally, we are ready to prove Theorem 1.9.

Proof of Theorem 1.9. Let $F := \mathsf{Ada}_{f,d}$. Suppose otherwise $\widetilde{\deg}_\epsilon(F) \le T$ for $\epsilon = 0.5 - 2^{-2^d+1}$. Then there exists a polynomial p such that $\|p\|_\infty \le 1 + \epsilon$, $p(x) \le 0.5 - 2^{-2^d+1}$ when $F(x) = 0$, and $p(x) \ge 0.5 + 2^{-2^d+1}$ when $F(x) = 1$.

Then we define polynomial $q(x) := (p(x) - 0.5) \cdot 2^{2^d-1}$. It is easy to see $q(x)$ computes F. Also, we have $\|q\|_\infty \le (\|p\|_\infty + 0.5) \cdot 2^{2^d-1} < 2^{2^d}$, which contradicts Lemma 3.7, and this completes the proof. ◀

Acknowledgment. I would like to thank Scott Aaronson, Adam Bouland, Dhiraj Holden and Prashant Vasudevan for several helpful discussions during this work, Ruosong Wang for many comments on an early draft of this paper, Justin Thaler for the suggestion on the application in hardness amplification for polynomial approximation, and Mika Göös and Thomas Watson for pointing out an issue in the proof of Theorem 1.7.

References

1 Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 461(2063):3473–3482, 2005.

2 Scott Aaronson. Impossibility of succinct quantum proofs for collision-freeness. *Quantum Information & Computation*, 12(1-2):21–28, 2012.

3 Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2011.

4 Richard Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational Complexity*, 4(4):339–349, 1994.

5 Adam Bouland, Lijie Chen, Dhiraj Holden, Justin Thaler, and Prashant Nalini Vasudevan. On SZK and PP. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 140, 2016.

6 Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 467(2126):459–472, 2011.

7 Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

8 Mark Bun and Justin Thaler. Dual polynomials for collision and element distinctness. *arXiv preprint arXiv:1503.07261*, 2015.

9 Mark Bun and Justin Thaler. Hardness amplification and the approximate degree of constant-depth circuits. In *International Colloquium on Automata, Languages, and Programming*, pages 268–280. Springer, 2015.

10 Mark Bun and Justin Thaler. Approximate degree and the complexity of depth three circuits. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 121, 2016.

11 Andrew Drucker and Ronald de Wolf. Quantum proofs for classical theorems. *arXiv preprint arXiv:0910.3376*, 2009.

12 Yenjo Han, Lane A Hemaspaandra, and Thomas Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.

13 Greg Kuperberg. How hard is it to approximate the Jones polynomial? *arXiv preprint arXiv:0908.0512*, 2009.

14 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational complexity*, 4(4):301–313, 1994.

15 Alexander A. Sherstov. Breaking the Minsky-Papert barrier for constant-depth circuits. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 223–232. ACM, 2014.

16 Alexander A Sherstov. The power of asymmetry in constant-depth circuits. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 431–450. IEEE, 2015.

17 Justin Thaler. Lower bounds for the approximate degree of block-composed functions. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 21, page 150, 2014.

18 NK Vereschchagin. On the power of pp. In *Structure in Complexity Theory Conference, 1992., Proceedings of the Seventh Annual*, pages 138–143. IEEE, 1992.

19 Mikhail N. Vyalyi. QMA = PP implies that PP contains PH. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 2003. URL: `http://eccc.hpi-web.de/report/2003/021/`.

Search on a Line by Byzantine Robots*

Jurek Czyzowicz[1], Konstantinos Georgiou[2], Evangelos Kranakis[3], Danny Krizanc[4], Lata Narayanan[5], Jaroslav Opatrny[6], and Sunil Shende[7]

1 **Département d'informatique, Université du Québec en Outaouais, Gatineau, QC, Canada**
Jurek.Czyzowicz@uqo.ca

2 **Department of Mathematics, Ryerson University, Toronto, Canada**
konstantinos@ryerson.ca

3 **School of Computer Science, Carleton University, Ottawa, Canada**
kranakis@scs.carleton.ca

4 **Department of Mathematics and Computer Science, Wesleyan University, Middletown CT, USA**
dkrizanc@wesleyan.edu

5 **Department of Computer Science and Software Engineering, Concordia University, Montreal, QC, Canada**
lata@cs.concordia.ca

6 **Department of Computer Science and Software Engineering, Concordia University, Montreal, QC, Canada**
opatrny@cs.concordia.ca

7 **Department of Computer Science, Rutgers University, Camden, USA**
sunil.shende@rutgers.edu

Abstract

We consider the problem of fault-tolerant parallel search on an infinite line by n robots. Starting from the origin, the robots are required to find a target at an unknown location. The robots can move with maximum speed 1 and can communicate in wireless mode among themselves. However, among the n robots, there are f robots that exhibit *byzantine faults*. A faulty robot can fail to report the target even after reaching it, or it can make malicious claims about having found the target when in fact it has not. Given the presence of such faulty robots, the search for the target can only be concluded when the non-faulty robots have sufficient verification that the target has been found. We aim to design algorithms that minimize the value of $S_d(n, f)$, the time to find a target at a distance d from the origin by n robots among which f are faulty. We give several different algorithms whose running time depends on the ratio f/n, the density of faulty robots, and also prove lower bounds. Our algorithms are optimal for some densities of faulty robots.

1998 ACM Subject Classification F.1.2 Modes of Computation, Parallelism and concurrency, B.8 Performance and Reliability, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Cow path problem, Parallel search, Mobile robots, Wireless communication, Byzantine faults

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.27

* A full version of the paper is available at http://arxiv.org/abs/1611.08209.

The 27th International Symposium on Algorithms and Computation 2016.
Editor: Seok-Hee Hong; Article No. 27; pp. 27:1–27:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Searching on a line (also known as a *single-lane cow-path* or a *linear search*) problem is concerned with a robot looking for a target placed at an unknown location on an infinite line; the robot moves with uniform (constant) speed and can change direction (without any loss in time) along this line. The ultimate goal is to find the target in optimal time [5]. Searching is central to many areas of computer science including data structures, computational geometry, and artificial intelligence. A version of the problem was first posed in 1963 by Bellman [12] and independently considered in 1964 by Beck [7], where the target was placed according to a known probability distribution on the real line, the robot was moving with uniform speed, and the goal was to find the target in minimum expected time.

In this paper, we consider the problem of *parallel, co-operative* search on the infinite line by n mobile robots at most f of which are faulty. The target is placed at a distance unknown to the robots. The robots start at the same time and location and can communicate instantaneously in wireless mode at any distance on the real line. While searching, the robots may co-operate by exchanging (broadcasting) messages; however, the search may be impeded by some of the robots (at most f) which may exhibit byzantine faults. The ultimate goal is to minimize the time it takes all non-faulty robots to be certain that the correct location of the target has been found.

1.1 Motion and communication model

To begin, we describe the robots' locomotive and communication models used in a search algorithm.

Robots and their trajectories. Robots are assumed to start at a common location, considered to be the origin of the line. They can move at maximum unit speed either along the positive direction (described as moving *right*) or along the negative direction (described as moving *left*); any robot can change direction arbitrarily often (by *turning*) without any loss in time. An algorithm for parallel search specifies a *trajectory* unique to each robot that is given by its turning points, and the speed(s) to follow between turning points. Since each robot has a distinct identity, it may also follow a distinct trajectory. Robots are assumed to have full knowledge of all trajectories, and moreover can communicate instantaneously with each other in wireless mode at any distance. Since robots know all the trajectories, the only kind of message broadcast by a robot R is whether or not it has found the target at some location; if R stays silent while visiting some location, the implicit assumption made by the other robots is that R did not detect the target there. Thus R follows its predefined trajectory until either it finds the target, in which case it announces that it has found the target, or it hears some other robot R' announce that it has found the target, at which point R may change its trajectory to participate in a verification protocol in regard to the announcement.

Messages and communication. All n robots know that f of the robots are faulty but they cannot differentiate in advance which among them are faulty; instead they must distinguish faulty from non-faulty ones based on conflict resolution and verification of messages received throughout the communication exchanges taking place during the execution of the search protocol. To this end, robots are equipped with pairwise distinct identities which they cannot alter at any time (in that respect our model is similar to the weakly Byzantine agent in [22]). In addition to the *correct* identity, the current location of a robot is automatically included in any broadcast message sent by the robot. Consequently, a faulty robot that does not follow its assigned trajectory can be immediately detected as faulty by the other robots, if it chooses to broadcast at some stage. In all other ways, faulty robots are indistinguishable from non-faulty (*reliable*) robots, except that the former can make *deliberate* positive and negative detection *errors* as follows. A non-faulty or *reliable* robot never lies when it has to confirm or deny the existence of the target at some location. Contrast this with *a faulty robot that may stay silent even when it detects or visits the target, or may claim that it has found the target when, in fact, it has not found it.* Thus, a reliable robot *cannot necessarily trust* an announcement that the target has been found, nor can it be certain that a location - visited silently by another robot - does not contain the target. In other words, the search for a target can terminate only after at least one robot that is *provably reliable* has visited the target and announced that it has been found. This requirement is critical to all our algorithms: if at some time, multiple robots make conflicting announcements at a location then the resulting (conflicting) *votes* can only be resolved if something is known about the number of reliable robots that participated in the vote. For instance, if three robots vote and it is known that two of them are reliable, then the majority vote would be the truth.

1.2 Preliminaries and notation

Consider a parallel search algorithm for a target located at distance d from the origin. First we define the search time of the algorithm and its corresponding competitive ratio.

▶ **Definition 1** (Search Time). Let $S_d(n, f)$ denote the time it takes for a search algorithm using a collection of n robots at most f of which are faulty, to find in parallel the location of a target placed at a distance d (unknown to the robots) from the starting position (the origin) of the robots on the line.

▶ **Definition 2** (Competitive Ratio). The corresponding *competitive ratio* is defined as $S_d(n, f)/d$, which is the ratio of the algorithm's search time and the lower bound d on the time taken by any algorithm for the problem.

For larger values of n and f, it will be more convenient to express our results in terms of the *density*, $\beta = \frac{f}{n}$, of faulty robots. This leads to the following definition.

▶ **Definition 3** (Asymptotic Competitive Search Ratio). Extend the definition of $S_d(n, f)$ above to non-integer values of n by replacing n with $\lceil n \rceil$ while the parameter f remains integral. Let $\beta = \frac{f}{n}$. Then

$$\hat{S}(\beta) = \min \ \{\alpha \mid \exists \text{ constant } c_\beta \text{ such that } \forall f > 0, \ S_d\left(f/\beta + c_\beta, f\right) \leq \alpha d\} \tag{1}$$

denotes the asymptotic competitive search ratio of any algorithm with search time $S_d(n, f)$.

Note that if $n \geq 4f + 2$, then in any partition of the robots into two groups each of size at least $2f + 1$, we will always have at least $f + 1$ reliable (non-faulty) robots per group. Therefore, an algorithm that sends the corresponding robots in the two groups in opposite directions is guaranteed to find the target in time d, because when the target is visited by one of the groups, a straightforward majority vote in the group confirms its presence reliably. Hence, $S_d(4f + 2, f) = d$, which is optimal. On the other hand, if $n \leq 2f$, there is no algorithm to complete the search: the f faulty robots may always completely disagree with the reliable ones, making it impossible to be certain of the location of the target. Therefore, in the sequel, we examine the interesting case where $2f + 1 \leq n \leq 4f + 1$.

Table 1 Upper and lower bounds on the search time $S_d(n, f)$ for a given number $n \leq 6$ of robots and faults $f = 1, 2$. Byz UB and Byz LB denote the known upper and lower bound for byzantine faults while Crash UB and Crash LB denote the known upper and lower bound for crash faults.

n, f	Byz. UB	Byz. LB	Crash-UB	Crash-LB
$3, 1$	$9d$	$3.93d$	$5.24d$	$3.76d$
$4, 1$	$3d$	$3d$	d	d
$5, 1$	$2d$	$2d$	d	d
$6, 1$	d	d	d	d
$5, 2$	$9d$	$3.57d$	$4.43d$	$3.57d$
$6, 2$	$4d$	$3d$	d	d

Table 2 Upper and lower bounds on the asymptotic competitive search ratio $\hat{S}(\beta)$ for various ranges of the density β. Note that for $\beta > \frac{1}{2}$ the search problem is impossible to solve.

β	$\leq \frac{1}{4}$	$(\frac{1}{4}, \frac{3}{10}]$	$(\frac{3}{10}, \frac{1}{3}]$	$(\frac{1}{3}, \frac{5}{14}]$	$(\frac{5}{14}, \frac{13}{34}]$	$(\frac{13}{34}, \frac{19}{46}]$	$(\frac{19}{46}, \frac{47}{110}]$	$(\frac{47}{110}, \frac{65}{146}]$	$(\frac{65}{146}, \frac{157}{396}]$	$(\frac{157}{396}, \frac{1}{2}]$
UB	1	2	3	3	4	5	6	7	8	9
LB	1	2	2	3	3	3	3	3	3	3

1.3 Our results

In Section 2, we are concerned mostly with upper bounds. Subsection 2.1 establishes the guiding principles for the design of algorithms.

We begin our study of upper bounds in Subsection 2.2 by establishing bounds for $S_d(n, f)$ for specific small values of n and f. These results are summarized in Table 1. For a comparison, we include in Table 1 known results on the search time for algorithms on the line with faulty robots that exhibit only *crash* faults [19], i.e., when the faulty robots never send any messages.

For larger values of n and f we express our results in terms of the density $\beta = \frac{f}{n}$ and show how to extend our algorithms from small values of n and f to this setting. Table 2 summarizes our results from Subsection 2.3.

Subsection 2.4 concludes Section 2 with several intriguing algorithms in that for densities $\frac{f}{n}$ between $\frac{3}{10}$ and $\frac{1}{3}$ the resulting search time is between $2d$ and $3d$. In Section 3, we derive two lower bounds on the search time. All missing proofs are found in the full version of the paper.

1.4 Related work

A search problem is usually seen as localization of a hidden target using searchers capable to move in the environment. It is an optimization question, usually attempting to minimize the time needed to complete the search. The question has been studied in numerous variations involving static or moving targets, one or many searchers, known or unknown environment, synchronous or asynchronous settings, different speed agents and many others (cf. [23]).

In several studies, when the environment is not known in advance, search implies exploration, often involving mapping and localizing searchers within the environment [2, 3, 21, 24, 26, 30]. However, even for the case of a known, simple environment like a line, there were several interesting studies attempting to optimize the search time. They started with the independent works of Bellman [12] and Beck [7], in which the authors attempted to minimize the competitive ratio in a stochastic setting. More exactly, they proved that time $9d$ is needed to guarantee finding the target situated at a (*a priori* unknown) distance d from the origin. Several other works on linear search followed (e.g. see [4, 7, 8, 9, 10, 11, 12]). More recently the search by a single searcher was studied for different models, e.g., when the turn cost was considered [20], when the bounds on the distance to the target are known in advance [14], and when the target was moving or for more general linear cost functions [13].

Most recently variants of linear search were studied for collections of collaborating searchers (robots). [16] considered linear group search, when the process is completed when the target is reached by the last robot visiting it. The robots collaborate attempting to minimize the group search time. However, [16] shows that having many robots does not help and the optimal search time is still bounded from below by $9d$. Group search using a pair of robots having distinct maximal speeds was studied in [6], in which techniques producing optimal search time were designed.

Fault tolerance was studied in distributed computing in various settings in the past (e.g., see [25, 28, 29]). However, the subject of unreliability was mainly for static components of the environment (e.g. network nodes or links), which was sometimes modelled by dynamically evolving environments (cf. [15, 27]). The malfunctions arising to mobile robots were investigated for various problems of gathering or pattern forming [1, 17, 22, 31] or patrolling [18]. Recently [19] investigated crash faults of robots performing linear search, where the time of finding the target by the first reliable robot was optimized. However, dealing with Byzantine agents is in general more tricky, requiring to identify and to refute the most malicious adversarial behavior (e.g., see [22]).

2 Upper Bounds

As already observed, if $n \geq 4f+2$, linear search can be performed optimally in time d, and no algorithm exists if $n \leq 2f$. Therefore, we consider below the case when $2f+1 \leq n \leq 4f+1$. Clearly, the robots can always stay together as a group, and perform the doubling zig-zag strategy that is optimal for a single robot and that has competitive ratio 9 [5, 7]. Since the reliable robots (at least $f+1$) are always in a majority, we are guaranteed to find the target. This yields the following upper bound:

▶ **Theorem 4.** *$S_d(n, f) \leq 9d$.*

In the remainder of this section, we provide upper bounds that, in general, are better than those suggested by Theorem 4 for the search problem. We do so by identifying and using some *guiding* principles to design search algorithms in the presence of faulty robots.

2.1 Principles for the design of algorithms

The general framework of our algorithms involves five basic principles, namely *Partition into Groups, Symmetry of Algorithms, Resolution of Conflicts, Simultaneous Announcements*, and *Computations by the Robots*, which we describe below in detail.

Partition into Groups. Depending on the ratio of faulty robots, we partition the robots into a certain number of groups. Two of the groups lead the exploration in opposite direction from the origin of the line. Further, each of these two groups will have at least $f+1$ robots so that at least some of the robots would announce the target when it is reached.

Symmetry of Algorithms. The algorithms are symmetric as far as left and right part of the line is concerned. We therefore typically discuss the behavior of the algorithm with respect to one side of the line only.

Resolution of Conflicts. If at any time there is an announcement of a target, the robots in the search groups stop until the claim is resolved. In the meantime, robots from some other group(s) move to resolve the claim. Once the claim is resolved, either the target is found and the robots stop, or a certain number of faulty robots is identified. From this time onward, the algorithm disregards any message from these faulty robots, effectively reducing the number of faulty robots to contend with, and the groups continue the search. Thus, each such announcement exposes more of the faulty robots, until eventually, we can be certain of a majority of robots in each search group being reliable, in which case the remaining search can be easily finished.

Simultaneous Announcements. When two announcements are being made at the same time, as usual with wireless transmissions, the algorithm deals only with one of them at a time, chosen arbitrarily. After the resolution of the first announcement is done and the search is possibly restarted, the robots redo their observation, and then the announcement is repeated if needed, thus taking into consideration the situation after the resolution of the first announcement. We show it does not influence the search time.

Computations by the Robots We assume that the time spent on calculations is negligible in comparison with the time spent in moving. Thus, we count only the time needed in movements of the robots until the target is found.

As indicated above, throughout the execution of the algorithms, conflicts will be resolved by voting. More precisely, we define $V(x,t)$ to be the *vote* of the robots about position x at time t. If y robots have claimed that the target is at x at or before time t, while z have claimed (by visiting and keeping silent) that it is not at x, then we say $V(x,t) = (y,z)$.

▶ **Definition 5** (Conflict). We say there is a *conflict* at position x at time t if $V(x,t) = (y,z)$, with $0 < y, z \leq f$.

The following two simple observations are used extensively in the proofs in this section.

▶ **Lemma 6.** *Let $V(x,t) = (y,z)$, and let f be the number of faulty robots before time t. Then*

1. *If $y > f$ then the target is at position x and the search is concluded.*
2. *If $z > f$ then the target is not at position x and y new faulty robots have been identified at time t.*

▶ **Lemma 7.** *Suppose at time t, there are f' faulty robots remaining, and there are at least $2f'+1$ robots at positions $\geq x$ and at least $2f'+1$ robots at positions $\leq -x$. Then any target that is distance d from the origin can be found in time $t + (d-x)$.*

To build intuition, we start with giving algorithms with at most 2 faulty robots, and later show how to use these techniques to give algorithms with asymptotic ratios for general values of n and f.

2.2 Algorithms for $n \leq 6$

Since $2f+1 \leq n < 4f+2$, there are only two kinds of possible combinations of values with $n \leq 6$: either $f = 1$ and $3 \leq n \leq 5$, or $f = 2$ and $5 \leq n \leq 6$.

▶ **Proposition 8.** $S_d(4,1) \leq 3d$

Proof of Proposition 8. Partition all robots into two search groups, L, and R, with two robots in each group. Each robot in R (L) moves right (left resp.) at speed 1 until it finds the target or hears an announcement that the target has been found. Suppose now that there is an announcement at time x from position $x > 0$. If $V(x,x) = (2,0)$, by Lemma 6, the target has been found at x and the algorithm terminates. Suppose that $V(x,x) = (1,1)$. Then one of the robots in L, say A, travels to x to resolve the conflict, taking additional time $2x$, while all other robots remain stationary. At time $3x$, the robot A reaches x. If $V(x,3x) = (2,1)$, by Lemma 6, the target has been found, and the algorithm terminates. If instead $V(x,3x) = (1,2)$, then by Lemma 6, the faulty robot is identified, and all other robots can be inferred to be reliable. Now the search continues with the groups moving in opposite directions with only the reliable robots being considered, until the target is found. Notice that an announcement at $-x$, simultaneous with that at x, would be resolved at time $3x$ with reliable robots. Therefore, if the target is at d or $-d$, the time taken to find it is $\leq 3x + d - x = 2x + d \leq 3d$ since $d > x$. Thus in all cases, $S_d(4,1) \leq 3d$. ◀

If the number of robots increases to $n = 5$ (while f still equals 1), then it is possible to send two groups of size 2 in opposite directions as in the algorithm above, but keep one *spare* robot at the origin for conflict resolution. This improves the search time to at most $2d$ since the spare is always at a distance d from a conflicting vote, and moreover, the spare is definitely reliable since the faulty robot is part of the conflicting vote..

▶ **Proposition 9.** $S_d(5,1) \leq 2d$

Note that the cases, $(n,f) = (5,2)$ or $(n,f) = (3,1)$, satisfy $n = 2f+1$, the bare minimum of robots necessary to guarantee termination. For these cases, it seems very difficult to improve upon the upper bound on $S_d(n,f) \leq 9d$ from Theorem 4. In fact, we conjecture that this best possible for the pairs $(5,2)$ and $(3,1)$ stated above.

By ensuring an appropriate *redistribution* of robots past the announcement of a conflict, we can show the following result:

▶ **Proposition 10.** *$S_d(6,2) \leq 4d$.*

2.3 Algorithms for large n

We now consider the case of large n, with different values of the density, $\beta = f/n$, of faulty robots. We start with generalizing the results from the previous subsection, then build recursive techniques that allow us to deal with larger densities of faulty robots, while paying a price in terms of the search time.

▶ **Theorem 11.** *$S_d\left(\frac{10f+4}{3}, f\right) \leq 2d$, provided that $f \equiv 2 \mod 3$.*

Using the fact that $S_d(n+k,f) \leq S_d(n,f)$ for any $k \geq 0$ and that $\hat{S}(\beta) \leq \hat{S}(\beta')$ if $\beta \leq \beta'$ we can easily derive the following corollary:

▶ **Corollary 12.** *If $\beta \leq \frac{3}{10}$ then $\hat{S}(\beta) \leq 2$.*

▶ **Theorem 13.** *$S_d\left(\frac{14f+4}{5}, f\right) \leq 3d$ provided $f \equiv 4 \mod 5$.*

Proof of Theorem 13. Partition the robots into two search groups L and R each containing $\frac{7f+2}{5}$ robots. The robots in L move left and those in R move right at speed 1. Without loss of generality, assume there is an announcement at x at time x. Let $V(x,x) = (y,z)$. Then if $\max\{y,z\} > f$, the announcement is resolved using Lemma 6. Suppose instead that $\max\{y,z\} \leq f$. Then $\min\{y,z\} \geq \frac{2f+2}{5}$ and at least $\frac{2f+2}{5}$ robots at x are faulty. In this case, $\frac{3f+3}{5}$ robots from L move from $-x$ to x, and at the same time $\frac{2f+2}{5}$ robots that voted yes and $\frac{2f+2}{5}$ that voted no are sent from x to $-x$. At time $3x$, in total $2f+1$ robots have voted at x, and by Lemma 6, either the target is identified, or at least $\frac{2f+2}{5}$ faulty robots are identified at $-x$ and may be disregarded from now on. There are at most $\frac{3f-2}{5}$ faulty robots unidentified. After the exchange of robots and elimination of the faulty robots in the worst case there are $\frac{6f+1}{5}$ robots in L and in R, i.e., a majority of reliable robots in both search groups. Therefore by Lemma 7, search for a target at distance d can be finished in time $3x + d - x \leq 3d$ as claimed. Note that all quantities are integral if $f \equiv 4 \mod 5$. ◀

As above, the following corollary is immediate:

▶ **Corollary 14.** *If $\beta \leq \frac{5}{14}$ then $\hat{S}(\beta) \leq 3$.*

As illustrated in the proofs of Theorems 11 and 13, when an announcement of a target is made, either the target can be confirmed, or the number of unidentified faulty robots can be reduced by an exchange of robots between the two search groups. For higher densities of faulty robots this technique can be repeated, for which we pay by an increase in the search time. This is the motivation for the recurrence formulas below that are used to obtain search algorithms for higher densities of robots.

▶ **Definition 15.** Let $T_x(l,s,r,f)$ be the minimum search time required by the robots to find the target given that initially, l robots are located at $-x$, s robots are at the origin 0, r robots are at $+x$, and f robots are faulty.

Since, as in the algorithms described so far, one way to solve our search problem is to send two equal-sized groups of robots to positions x and $-x$, we get the following upper bound.

▶ **Lemma 16.** *$\forall d \geq x > 0$ $S_d(n,f) \leq x + T_x(n/2, 0, n/2, f)$. Furthermore, if $n/2 \geq 2f+1$ then $T_x(n/2,0,n/2,f) = d - x$.*

If there is an announcement at x, we can identify some of the faulty robots, and by paying a price in terms of additional time, we can reduce it to a new problem with a smaller number of faulty robots. This can be encapsulated in the following lemma:

▶ **Lemma 17.** *Let $k > 0$ be even. Suppose there is an announcement at distance x from the origin. Then for all $a \geq x$, $T_x(f+k, 0, f+k, f) \leq 2x + T_a(f+k/2, 0, f+k/2, f-k)$.*

Proof of Lemma 17. Assume there are $f+k$ robots each at x and $-x$, with at most f faulty robots in all, and that a conflict occurs at $x > 0$ at some time t. Let $V(x,x) = (y,z)$. Then $k \leq \min\{y,z\} \leq \max\{y,z\} \leq f$. Now the robots move as follows:

1. All $f+k$ robots at position x move to $-x$.
2. $f + k/2$ of the robots at $-x$ move to x.

Note that these movements take time $2x$, and there are now $f+k/2$ robots at x and $f+k/2+k$ robots at $-x$. Since $2f + 3k/2$ robots have now visited x, the vote $V(x,3x)$ is enough to resolve the conflict, and there remain at most $f-k$ faulty robots among the total $2f+k$ robots. This proves the lemma. ◀

Lemma 17 along with Theorem 13 can be used to obtain slower algorithms for higher densities. We have:

▶ **Theorem 18.**
1. *$\hat{S}(\beta) \leq 5$ for $\beta \leq 19/46$.*
2. *$\hat{S}(\beta) \leq 7$ for $\beta \leq 65/146$.*

Similar to Lemma 17 the following lemma establishes a recurrence that can be used to extend Theorem 11 to higher densities (at a cost of a higher competitive ratio).

▶ **Lemma 19.** *Suppose there is an announcement at distance x from the origin. Then for all $a \geq x$ and $k \geq f/4$: $T_x(f+k, 0, f+k, f) \leq 2x + T_a\left(\frac{4(f-k)}{3}, \frac{2(f-k)}{3}, 3k, f-k\right)$.*

Using a similar argument to that used in Theorem 18 we can apply Lemma 19 and Theorem 11 to get:

▶ **Theorem 20.**
1. *$\hat{S}(\beta) \leq 4$ for $\beta \leq 13/34$.*
2. *$\hat{S}(\beta) \leq 6$ for $\beta \leq 47/110$.*
3. *$\hat{S}(\beta) \leq 8$ for $\beta \leq 157/396$.*

2.4 Algorithms for $\frac{3}{10} \leq \beta < \frac{1}{3}$

Finally we discuss a new class of algorithms for densities of $\frac{f}{n}$ between $\frac{3}{10}$ and $\frac{1}{3}$ whose search time is between $2d$ and $3d$.

Informally, in any of these algorithms, the robots are partitioned into two search groups, that move in opposite directions at speed 1, and i middle groups, i odd, $i \geq 3$, positioned at regular intervals between the search groups. These i groups are used to solve any conflict reached by the search groups. The positioning of the middle groups between the search groups is achieved by them moving at a fraction of the maximal speed.

When a vote arises that cannot be resolved using Lemma 6, the middle groups are moved to the point of conflict in sequence at speed 1 until a resolution of the conflict is obtained. The middle groups not used in the resolution of a conflict on one side can be used to resolve a conflict on the other side. This approach allows a fine-grain resolution of a conflict by taking into account the result of the vote each time a group arrives to the conflict point.

▶ **Lemma 21.** *Let i be an odd integer, $i \geq 3$.*
$S_d(\frac{(3i+2)f}{i+1} + 2, f)) \leq \left(3 - \frac{2}{i+1}\right) d$, provided $f \equiv 0 \mod (i+1)$.

▶ **Corollary 22.**
1. *$\hat{S}(\beta) \leq 2.5$ for $\beta \leq 4/13$.*
2. *$\hat{S}(\beta) \leq 2.67$ for $\beta \leq 6/19$.*
3. *$\hat{S}(\beta) \leq 2.75$ for $\beta \leq 8/25$.*
4. *$\hat{S}(\beta) \leq 2.8$ for $\beta \leq 10/31$.*

3 Lower Bounds

It is straightforward to see that to achieve search time d, $4f + 2$ robots are necessary; with $4f + 1$ or fewer robots, at time d, either d or $-d$ can be visited by at most $2f$ robots. The adversary can make f of these $2f$ robots faulty, and it is impossible to be certain about the answer. Formally we can prove the following result.

▶ **Lemma 23.** $S_d(5,1) \geq 2d$.

Proof of Lemma 23. At time $d-\epsilon$ no one has visited d or $-d$. Consider where the robots are at this time. It must be the case that one of the intervals $(-d,0)$ or $(0,d)$ contains at most 2 robots. Without loss of generality say it is $(0,d)$. Put the target at d. Sort the robots by distance to d (ties broken arbitrarily) and make the robot closest to d faulty and silent. Then at least one robot from $(-d,0]$ must also reach d so that two non-faulty robots can identify the target at d. Thus, the search time is at least $d-\epsilon+d=2d-\epsilon$. ◀

The next theorem shows that the density $f/n = \frac{3}{10}$ in Theorem 11 is also a lower bound on this ratio if we want to maintain the search time to be at most $2d$.

▶ **Theorem 24.** *If $S_d(n,f) \leq 2d$ then $\frac{f}{n} \leq \frac{3}{10}$.*

Proof of Theorem 24. Assume on the contrary that $\frac{n}{f} \leq \frac{10}{3} - \epsilon$ and that there is an algorithm for solving the search problem in time $2d$. Observe the intervals $[-d,0)$, $\{0\}$, $(0,+d]$ at time d and let us denote by l,r the number of robots within $[-d,0)$, $(0,+d]$, and by s the number of robots at the origin 0, respectively. By assumption $l+r+s=(10/3-\epsilon)f$. Observe that robots which are located at points different from $-d,0,d$ at time d may not be helpful in reducing the $2d$ search time. Thus, without loss of generality we may assume that at time d only the points $-d,0,+d$ are occupied by robots. Without loss of generality assume that $r \leq l$. We derive a contradiction by considering two cases.

1. Either l or $r \geq \frac{4}{3}f$. In this case we have that $r+s=\frac{10}{3}f-\epsilon-l \leq \frac{10}{3}f-\epsilon-\frac{4}{3}f=2f-\epsilon$. Thus, $s+r$ robots are not sufficient to resolve conflicts on the right possibly involving f faulty robots within time $2d$.
2. Assume that there exists $\epsilon > 0$ such that both, $l,r \leq (\frac{4}{3}-\epsilon)f$. In particular, consider $r \leq (\frac{4}{3}-\epsilon)f$. Consider time d and suppose that up to $min\{r, \frac{1}{3}f\}$ of robots at d claim to find the target. For the algorithm to attain time $2d$, robots must be send from the start position 0 at time d to position d so as to verify the claim. Since among the robots sent to $+d$ from 0 we could have all remaining faulty robots, the number of robots sent from 0 must be at least $2f+1-r$ so that we a decision at time $2d$ can be made. However, if the target is not at $+d$ then the adversary could make it so that only $\frac{1}{3}f$ robots are faulty at $+d$ from among $2f+1$ robots. However, now we have at most $\frac{10}{3}f-\epsilon-2f-1=\frac{4}{3}f-\epsilon-1$ robots at 0 or to the left of 0 and still $\frac{2}{3}f$ faulty robots remain among them. Thus, any claim of target at $-d'$ to the left of $-d$ cannot be verified in time $2d'$ by the available robots.

This proves the theorem. ◀

▶ **Lemma 25.** $S_d(3f+1,f)=3d$.

Proof of Lemma 25. The upper bound $S_d(3f+1,f) \leq 3d$ has been proved in Theorem 13. To prove the lower bound $S_d(3f+1,f) \geq 3d$ we argue as follows. Consider visits to the set of symmetric positions $\{-d,+d\}$ by the robots. In particular, consider the first time t that at least $f+1$ robots complete visits to the *second* of the positions in the set. For instance, without loss of generality, assume that position $-d$ is visited first by at least $f+1$ robots and later (or instantaneously) by at least $f+1$ robots. Clearly the time t is at least d. The adversary arranges for a conflict at position $+d$. Note that unless $t \geq 3d$, the sets of robots visiting the two positions must be disjoint, and hence, the conflict at position $+d$ involves at most $2f$ robots participating in a vote, i.e. to resolve the conflict, at least one of the robots that visited $-d$ must move to $+d$. It follows that the total time required is at least $t+2d \geq 3d$. ◀

Note that Lemma 25 implies a lower bound for densities β in the range $1/3 > \beta > 3/10$. In case $n = 3,\ f = 1$, we can show the following lower bound on the search time.

▶ **Lemma 26.** *$S_d(3,1) \geq 3.93d$.*

Proof. (Lemma 26) We start by considering three positive real numbers x, y, α such that

$$\frac{\alpha-1}{2} \leq x < y \leq \frac{2}{\alpha-3} \text{ and } \frac{\alpha-1}{2} \leq \frac{y}{x} \leq \frac{2}{\alpha-3}. \tag{2}$$

We will show that an α satisfying Inequalities (2) above is the competitive ratio of all search algorithms for three robots with one Byzantine fault. Moreover, using Mathematica it can be shown that the maximum value of α that satisfies (2) is 3.93.

Consider numbers $-y,\ -x,\ -1,\ 0,\ 1,\ x,\ y$ on the real line and the movement of the three robots with respect to these points. Assume on the contrary the competitive ratio is some value ρ such that $\rho < \alpha$. Throughout the arguments below we are using Inequalities (2).

Observe that two robots must visit the points $-1, 1$ before time α, otherwise we get a contradiction to the competitive ratio because of Inequality (2). Therefore there exists a robot, say A, that visited both of these points before time α. Same argument applies for points $-x, x$. There exist a robot that visits both points $-x, x$ before time αx. Observe that this robot cannot be A. Indeed, otherwise it takes either time $2x + 1$ to reach point -1 or time $2 + 3x$ to reach point x. Let B be the robot that visits both points $-x, x$ before time αx. Because of the time constraints in Inequalities (2) the robot B must have either positive trajectory (i.e., visiting x before $-x$) or negative trajectory (i.e., visiting $-x$ before x). However, it is easy to see that B cannot have a positive trajectory because it would be too far to confirm an target placed at -1. This proves the lemma ◀

4 Discussion

In this paper, we considered a generalization of the well-known cow-path problem by having the search done in parallel with a group of n robots, with up to f of them being byzantine faulty. We presented optimal search algorithms for several ranges of values for $\beta = f/n$, the fraction of faulty robots, and gave non-trivial upper and lower bounds in many cases. Several interesting problems in the setting remain open, the most interesting one being to give tight upper and lower bounds in the case $n = 2f + 1$.

References

1 N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

2 S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

3 S. Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32(1):123–143, 2002.

4 S. Alpern and S. Gal. *The theory of search games and rendezvous*, volume 55. Kluwer Academic Publishers, 2002.

5 R. Baeza Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

6 E. Bampas, J. Czyzowicz, L. Gasieniec, D. Ilcinkas, R. Klasing, T. Kociumaka, and D. Pajak. Linear search by a pair of distinct-speed robots. In *SIROCCO*. to appear, 2016.

7 A. Beck. On the linear search problem. *Israel J. of Mathematics*, 2(4):221–228, 1964.

8 A. Beck. More on the linear search problem. *Israel J. of Mathematics*, 3(2):61–70, 1965.

9 A. Beck and M. Beck. Son of the linear search problem. *Israel Journal of Mathematics*, 48(2-3):109–122, 1984.

10 A. Beck and D. Newman. Yet more on the linear search problem. *Israel J. of Mathematics*, 8(4):419–429, 1970.

11 A. Beck and P. Warren. The return of the linear search problem. *Israel J. of Mathematics*, 14(2):169–183, 1973.

12 R. Bellman. An optimal search. *SIAM Review*, 5(3):274–274, 1963.

13 P. Bose and J.-L. De Carufel. A general framework for searching on a line. In *WALCOM: Algorithms and Computation – 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings*, pages 143–153, 2016.

14 P. Bose, J.-L. De Carufel, and S. Durocher. Revisiting the problem of searching on a line. In *Algorithms – ESA 2013 – 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 205–216, 2013.

15 A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. In *Ad-hoc, mobile, and wireless networks, LNCS*, volume 6811, pages 346–359. Springer, 2011.

16 M. Chrobak, L. Gasieniec, Gorry T., and R. Martin. Group search on the line. In *SOFSEM 2015*. Springer, 2015.

17 R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal of Computing*, 41(1):1516–1528, 2005.

18 J. Czyzowicz, L. Gasieniec, A. Kosowski, E. Kranakis, D. Krizanc, and N. Taleb. When patrolmen become corrupted: Monitoring a graph using faulty mobile robots. In *Algorithms and Computation – Proceedings of 26th ISAAC 2015*, pages 343–354, 2015.

19 J. Czyzowicz, E. Kranakis, D. Krizanc, L. Narayanan, and Opatrny J. Search on a line with faulty robots. In *PODC*. ACM, 2016.

20 E. D. Demaine, S. P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361(2):342–355, 2006.

21 X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *FOCS*, pages 298–303. IEEE, 1991.

22 Y. Dieudonné, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Transactions on Algorithms (TALG)*, 11(1):1, 2014.

23 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

24 F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001.

25 J. Hromkovič, R. Klasing, B. Monien, and R. Peine. Dissemination of information in interconnection networks (broadcasting & gossiping). In *Combinatorial network theory*, pages 125–212. Springer, 1996.

26 J. Kleinberg. On-line search in a simple polygon. In *SODA*, page 8. SIAM, 1994.

27 F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522. ACM, 2010.

28 L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

29 N. A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

30 C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *ICALP*, pages 610–620. Springer, 1989.

31 S. Souissi, X. Défago, and M. Yamashita. Gathering asynchronous mobile robots with inaccurate compasses. *Principles of Distributed Systems*, pages 333–349, 2006.

Bipartite Matching with Linear Edge Weights*

Nevzat Onur Domaniç[1], Chi-Kit Lam[2], and C. Gregory Plaxton[3]

1 **University of Texas at Austin, Dept. of Computer Science, Austin, TX, USA**
onur@cs.utexas.edu
2 **University of Texas at Austin, Dept. of Computer Science, Austin, TX, USA**
geocklam@cs.utexas.edu
3 **University of Texas at Austin, Dept. of Computer Science, Austin, TX, USA**
plaxton@cs.utexas.edu

Abstract

Consider a complete weighted bipartite graph G in which each left vertex u has two real numbers *intercept* and *slope*, each right vertex v has a real number *quality*, and the weight of any edge (u, v) is defined as the intercept of u plus the slope of u times the quality of v. Let m (resp., n) denote the number of left (resp., right) vertices, and assume that $m \geq n$. We develop a fast algorithm for computing a maximum weight matching (MWM) of such a graph. Our algorithm begins by computing an MWM of the subgraph induced by the n right vertices and an arbitrary subset of n left vertices; this step is straightforward to perform in $O(n \log n)$ time. The remaining $m - n$ left vertices are then inserted into the graph one at a time, in arbitrary order. As each left vertex is inserted, the MWM is updated. It is relatively straightforward to process each such insertion in $O(n)$ time; our main technical contribution is to improve this time bound to $O(\sqrt{n} \log^2 n)$. This result has an application related to unit-demand auctions. It is well known that the VCG mechanism yields a suitable solution (allocation and prices) for any unit-demand auction. The graph G may be viewed as encoding a special kind of unit-demand auction in which each left vertex u represents a unit-demand bid, each right vertex v represents an item, and the weight of an edge (u, v) represents the offer of bid u on item v. In this context, our fast insertion algorithm immediately provides an $O(\sqrt{n} \log^2 n)$-time algorithm for updating a VCG allocation when a new bid is received. We show how to generalize the insertion algorithm to update (an efficient representation of) the VCG prices within the same time bound.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Weighted bipartite matching, Unit-demand auctions, VCG allocation and pricing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.28

1 Introduction

Given an undirected graph $G = (V, E)$, a *matching* of G is a subset M of E such that no two edges in M share an endpoint. If G is a weighted graph, we define the weight of a matching as the sum of the weights of its constituent edges. The problem of finding a maximum weight matching (MWM) of a weighted bipartite graph, also known as the “assignment problem” in operations research, is a basic and well-studied problem in combinatorial optimization. A classic algorithm for the assignment problem is the Hungarian method [11], which admits an $O(|V|^3)$-time implementation. For dense graphs with arbitrary edge weights, this time bound remains the fastest known. Fredman and Tarjan [5] introduce Fibonacci heaps,

* This research was supported by NSF Grant CCF–1217980.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 28; pp. 28:1–28:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and by utilizing this data structure to speed up shortest path computations, they obtain a running time of $O(|V|^2 \log |V| + |E| \cdot |V|)$ for the maximum weight bipartite matching problem. When the edge weights are integers in $\{0, \ldots, N\}$, Duan and Su [4] give a scaling algorithm with running time $O(|E|\sqrt{|V|} \log N)$. In this paper, we consider a restricted class of complete weighted bipartite graphs where the edge weights have a special structure. Both unweighted and weighted matching problems in restricted classes of bipartite graphs have been studied extensively. Glover [7], Lipski and Preparata [13], Gabow and Tarjan [6], Steiner and Yeomans [15], and Katriel [10] study matching problems in convex bipartite graphs, the graphs in which the right vertices can be ordered in such a way that the neighbors of each left vertex are consecutive. Plaxton [14] studies vertex-weighted matchings in two-directional orthogonal ray graphs, which generalize convex bipartite graphs.

In the present paper, we develop a fast algorithm for computing an MWM of a complete weighted bipartite graph with the following special structure: there are m left vertices, each of which has two associated real values, a "slope" and an "intercept"; there are n right vertices, each of which has an associated real "quality"; for each left vertex u and right vertex v, the weight of edge (u, v) is given by the slope of u times the quality of v plus the intercept of u. Since the weight of any edge (u, v) is determined by evaluating the linear function specified by u (via the slope and intercept) on the quality of v, we refer to this problem as *bipartite matching with linear edge weights.* Assuming that $m \geq n$, we solve this problem in $O(m\sqrt{n} \log^2 n)$ time. We begin by solving the problem on a subgraph induced by the n right vertices and an arbitrary subset of n left vertices; this turns out to be easy to accomplish in $O(n \log n)$ time via sorting. We then insert the remaining left vertices one at a time, in arbitrary order. As each left vertex is inserted, we update the solution in $O(\sqrt{n} \log^2 n)$ time. It is relatively straightforward to process each such insertion in $O(n)$ time, yielding an overall $O(mn)$ time bound. Our algorithm provides a significant improvement over the latter bound, which is the fastest previous result that we are aware of.

In recent work that is closely related to the current paper, Domaniç and Plaxton [3] present a fast algorithm for bipartite matching with linear edge weights in the special case where the qualities of the right vertices form an arithmetic sequence. Assuming that $m \geq n$, their algorithm runs in $O(m \log m)$ time. Applying that algorithm to the scheduling domain directly solves the problem of scheduling unit jobs on a single machine with a common deadline where each job has a weight and a profit, and the objective is to minimize the sum of the weighted completion times of the scheduled jobs plus the sum of the profits of the rejected jobs. Domaniç and Plaxton [3] also provide an extension that preserves the $O(m \log m)$ time bound for the special case where the qualities correspond to the concatenation of two arithmetic sequences. This extension solves a more general scheduling problem that incorporates weighted tardiness penalties with respect to a common due date into the objective.

By removing the technical restrictions on the qualities imposed in [3], the algorithm of the present paper supports a richer edge weight structure, while continuing to admit a compact graph representation that uses space linear in the number of vertices. In terms of scheduling, the present algorithm addresses a broader class of problems than [3]; for example, it can handle symmetric earliness and tardiness penalties with respect to a common due date, and allows certain time slots to be marked as unavailable. Below we discuss another motivation for the present work, which is based on its connection to unit-demand auctions.

In a unit-demand auction of a collection of items, each bidder submits a bid that specifies a separate offer on each item, which may or may not be equal to the private valuation that the bidder has for that item [1]. The outcome of the unit-demand auction is a pricing of the items and an allocation of each bidder to at most one item. In mechanism design, it is known that the VCG mechanism is the only mechanism for unit-demand auctions that achieves the desired properties of being efficient, strategyproof, and envy-free [8, 12]. Such an auction can be modeled as a bipartite graph in which each left vertex represents a bid, each right vertex represents an item, and the weight of the edge from a bid u to an item v represents the offer of the bid u on item v. Then, a VCG allocation corresponds to an MWM of such bipartite graph, and the VCG prices correspond to the dual variables computed by the Hungarian method, i.e., they correspond to the prices having the minimum sum among the ones that are the solutions to the dual of the linear program that solves the assignment problem encoding the auction.

The main motivation for our interest in the problem we consider in this paper, given the aforementioned desirable properties of the VCG mechanism, is to find frameworks to encode unit-demand auctions that are expressive enough to have suitable applications while being restrictive enough to yield efficient algorithms for finding VCG outcomes. For instance, consider a unit-demand auction for last-minute vacation packages in which some trusted third party (e.g., TripAdvisor) assigns a "quality" rating for each package and each bidder formulates a unit-demand bid for every package by simply declaring a linear function of the qualities of packages, i.e., determining the intercept and slope of this linear function. Within this context, we can formulate an auction as a complete weighted bipartite graph in the family that we consider in this paper. In some of the popular auction sites, e.g., eBay, bidding takes place in multiple rounds. eBay implements a variant of an English auction to sell a single item; the bids are sealed, but the second highest bid (plus one small bid increment), which is the amount that the winner pays, is displayed throughout the auction. We employ a similar approach by accepting the bids one-by-one and by maintaining an efficient representation of the tentative outcome for the enlarged set of bids. We show that we can process each bid in $\tilde{O}(\sqrt{n})$ time where n denotes the number of items in the auction. More precisely, we present a data structure that is initialized by the entire set of n items; the bids are introduced one-by-one in any order; the data structure maintains a compact representation of a VCG outcome (allocation and prices) for the bids introduced so far and for the entire set of items; it takes $O(\sqrt{n}\log^2 n)$ time to introduce a bid; it takes $O(n)$ time to print the outcome at any time.

Organization. In Sect. 2, we give the formal definition of the problem and introduce some useful definitions. In Sect. 3, we present an incremental framework for solving the problem. In Sect. 4, we present a basic algorithm within the framework of Sect. 3. Built on the concepts introduced in Sect. 4, we introduce a data structure and present our fast algorithm in Sect. 5. The companion technical report [2] includes all of the material in the present version plus some details and the proofs of all lemmas and theorems, which are omitted due to space limitations. In [2, Section 6], we extend the incremental framework to compute the VCG prices, and we present the algorithm within that framework.

2 Preliminaries

A *bid* is a triple $u = (slope, intercept, id)$ where *slope* and *intercept* are real numbers, and *id* is an integer. We use the notation $u.slope$ and $u.intercept$ to refer to the first and second components of a bid u, respectively. The bids are ordered lexicographically. An *item* is a pair $v = (quality, id)$ where *quality* is a real number and *id* is an integer. We use the notation $v.quality$ to refer to the first component of an item v. The items are ordered lexicographically.

For any bid u and any item v, we define $w(u, v)$ as $u.\mathit{intercept} + u.\mathit{slope} \cdot v.\mathit{quality}$.

For any set of bids U and any set of items V, we define the pair (U, V) as a *unit-demand auction with linear edge weights* (*UDALEW*). Such an auction represents a unit-demand auction instance where the set of bids is U, the set of items is V, and each bid u in U offers an amount $w(u, v)$ on each item v in V.

A UDALEW $A = (U, V)$ corresponds to a complete weighted bipartite graph G where left vertices are U, right vertices are V, and the weight of the edge between a left vertex u and a right vertex v is equal to $w(u, v)$. Hence, for a UDALEW, we use the standard graph theoretic terminology, alluding to the corresponding graph. The family of all such graphs G corresponds to the general graph family introduced in [3].

A *matching* of a UDALEW (U, V) is a set M of bid-item pairs where each bid (resp., item) in M belongs to U (resp., V) and no bid (resp., item) appears more than once in M. The *weight* of a matching M, denoted $w(M)$, is defined as the sum, over all bid-item pairs (u, v) in M, of $w(u, v)$.

In this paper, we solve the problem of finding a VCG outcome (allocation and prices) for a given UDALEW A; a VCG allocation is any MWM of A, and we characterize the VCG prices in [2, Sect. 6.2]. We reduce the problem of finding an MWM to the problem of finding a maximum weight maximum cardinality matching (MWMCM) as follows: we enlarge the given UDALEW instance $A = (U, V)$ by adding $|V|$ dummy bids to U, each with intercept zero and slope zero; we compute an MWMCM M of the resulting UDALEW A'; we remove from M all bid-item pairs involving dummy bids.

We conclude this section with some definitions that prove to be useful in the remainder of the paper. For any totally ordered set S — such as a set of bids, a set of items, or an ordered matching which we introduce below — we make the following definitions: any integer i is an *index* in S if $1 \le i \le |S|$; for any element e in S, we define the *index of e in S*, denoted $\mathit{index}(e, S)$, as the position of e in the ascending order of elements in S, where the index of the first (resp., last) element, also called the *leftmost* (resp., *rightmost*) element, is 1 (resp., $|S|$); $S[i]$ denotes the element with index i in S; for any two indices i and j in S such that $i \le j$, $S[i : j]$ denotes the set $\{S[i], \ldots, S[j]\}$ of size $j - i + 1$; for any two integers i and j such that $i > j$, $S[i : j]$ denotes the empty set; for any integer i, $S[\,: i]$ (resp., $S[i :\,]$) denotes $S[1 : i]$ (resp., $S[i : |S|]$); a subset S' is a *contiguous* subset of S if $S' = S[i : j]$ for some $1 \le i \le j \le |S|$.

For any matching M, we define $\mathit{bids}(M)$ (resp., $\mathit{items}(M)$) as the set of bids (resp., items) that participate in M. A matching M is *ordered* if M is equal to $\bigcup_{1 \le i \le |M|} \{(U[i], V[i])\}$ where U denotes $\mathit{bids}(M)$ and V denotes $\mathit{items}(M)$. The order of the pairs in an ordered matching is determined by the order of the bids (equivalently, items) of those pairs.

3 Incremental Framework

In this section, we present an incremental framework for the problem of finding an MWMCM of a given UDALEW $A = (U, V)$. As discussed below, it is a straightforward problem if $|U| \le |V|$. Thus, the primary focus is on the case where $|U| > |V|$. We start with a useful definition and a simple lemma.

For any set of bids U and any set of items V such that $|U| = |V|$, we define $\mathit{matching}(U, V)$ as the ordered matching $\{(U[1], V[1]), \ldots, (U[|U|], V[|U|])\}$.

Lemma 1 below shows how to compute an MWMCM of a UDALEW where the number of bids is equal to the number of items. The proof follows from the rearrangement inequality [9, Section 10.2, Theorem 368].

▶ **Lemma 1.** *For any UDALEW $A = (U, V)$ such that $|U| = |V|$, matching(U, V) is an MWMCM of A.*

▶ **Corollary 2.** *For any UDALEW $A = (U, V)$ such that $|U| \geq |V|$, there exists an ordered MWMCM of A.*

If $|U| < |V|$ in a given UDALEW (U, V), then it is straightforward to reduce the problem to the case where $|U| = |V|$. Let U' (resp., U'') denote the set of the bids in U having negative (resp., nonnegative) slopes. Then we find an MWMCM M' of the UDALEW $(U', V[\,:|U'|])$ and an MWMCM M'' of the UDALEW $(U'', V[|V| - |U''| + 1 :\,])$, and we combine M' and M'' to obtain an MWMCM of (U, V).

It remains to consider the problem of finding an MWMCM of a UDALEW (U, V) where $|U| > |V|$. The following is a useful lemma. The proof is straightforward by an augmenting path argument; see [3, Lemma 7] for the proof of a similar claim.

▶ **Lemma 3.** *Let $A = (U, V)$ be a UDALEW such that $|U| \geq |V|$. Let u be a bid that does not belong to U. Let M be an MWMCM of A and let U' denote bids(M). Then, any MWMCM of the UDALEW $(U' + u, V)$ is an MWMCM of the UDALEW $(U + u, V)$.*

Lemma 3 shows that the problem of finding an MWMCM of a UDALEW (U, V) where $|U| = |V| + k$ reduces to k instances of the problem of finding an MWMCM of a UDALEW where the number of bids exceeds the number of items by one. Below we establish an efficient incremental framework for solving the MWMCM problem based on this reduction.

For any ordered matching M and any bid u that does not belong to *bids*(M), we define *insert*(M, u) as the ordered MWMCM M' of the UDALEW $A = (\textit{bids}(M) + u, \textit{items}(M))$ such that the bid that is left unassigned by M', i.e., $(\textit{bids}(M) + u) \setminus \textit{bids}(M')$, is maximum, where the existence of M' is implied by Corollary 2.

We want to devise a data structure that maintains a dynamic ordered matching M. When the data structure is initialized, it is given an ordered matching M', and M is set to M'; we say that the data structure has initialization cost $T(n)$ if initialization takes at most $T(|M'|)$ steps. Subsequently, the following two operations are supported: the *bid insertion* operation takes as input a bid u not in *bids*(M), and transforms the data structure so that M becomes *insert*(M, u); the *dump* operation returns a list representation of M. We say that the data structure has bid insertion (resp., dump) cost $T(n)$ if bid insertion (resp., dump) takes at most $T(|M|)$ steps.

▶ **Lemma 4.** *Let $\mathcal{D}$ be an ordered matching data structure with initialization cost $f(n)$, bid insertion cost $g(n)$, and dump cost $h(n)$. Let A be a UDALEW (U, V) such that $|U| \geq |V|$. Then an MWMCM of A can be computed in $O(f(|V|) + (|U| - |V|) \cdot g(|V|) + h(|V|))$ time.*

In Sect. 4, we give a simple linear-time bid insertion algorithm assuming an array representation of the ordered matching. Building on the concepts introduced in Sect. 4, Sect. 5 develops an ordered matching data structure with initialization cost $O(n \log^2 n)$, bid insertion cost $O(\sqrt{n} \log^2 n)$, and dump cost $O(n)$ (Theorem 8). The results of Sect. 5, together with Lemma 4, yield the $O(m\sqrt{n} \log^2 n)$ MWMCM time bound claimed in Sect. 1.

Looking from an auction perspective, as discussed in Sect. 2, our goal is to compute a VCG allocation and pricing given a UDALEW. In [2, Sect. 6], we show how to extend the data structure of Sect. 5 to maintain the VCG prices as each bid is inserted. The asymptotic time complexity of the operations remain the same; the additional computation for maintaining the VCG prices takes $O(\sqrt{n})$ time at each bid insertion, where n denotes the size of the matching maintained by the data structure.

4 A Basic Bid Insertion Algorithm

In this section, we describe a linear-time implementation of $\mathit{insert}(M, u)$ given an array representation of the ordered matching. The algorithm described here is not only useful because it introduces the concepts that the fast algorithm we introduce in Sect. 5 is built on, but also the same approach is used in certain "block scan" computations of that fast algorithm. We first introduce two functions that, in a sense evident by their definitions, restrict $\mathit{insert}(M, u)$ into two halves, left and right, of M split by u.

For any ordered matching M and any bid u that does not belong to $\mathit{bids}(M)$, we define $\mathit{insert}_L(M, u)$ (resp., $\mathit{insert}_R(M, u)$) as the ordered MCM M' of the UDALEW $A = (\mathit{bids}(M) + u, \mathit{items}(M))$ of maximum weight subject to the condition that the bid that is left unassigned by M', i.e., $(\mathit{bids}(M) + u) \setminus \mathit{bids}(M')$, is less (resp., greater) than u, where the ties are broken by choosing the MCM that leaves the maximum such bid unassigned; if no such MCM exists, i.e., u is less (resp., greater) than every bid in $\mathit{bids}(M)$, then $\mathit{insert}_L(M, u)$ (resp., $\mathit{insert}_R(M, u)$) is defined as M.

The following lemma characterizes $\mathit{insert}(M, u)$ in terms of $\mathit{insert}_L(M, u)$ and $\mathit{insert}_R(M, u)$; the proof directly follows from the definitions of $\mathit{insert}(M, u)$, $\mathit{insert}_L(M, u)$, and $\mathit{insert}_R(M, u)$.

▶ **Lemma 5.** *Let M be a nonempty ordered matching and let u be a bid that does not belong to $\mathit{bids}(M)$. Let M_L denote $\mathit{insert}_L(M, u)$ and let M_R denote $\mathit{insert}_R(M, u)$. Let W denote the maximum of $w(M_L)$, $w(M)$, and $w(M_R)$. Then,*

$$\mathit{insert}(M, u) = \begin{cases} M_R & \text{if } w(M_R) = W \\ M & \text{if } w(M) = W > w(M_R) \\ M_L & \text{otherwise.} \end{cases}$$

We now introduce some definitions that are used in Lemma 6 below to characterize $\mathit{insert}_L(M, u)$ and $\mathit{insert}_R(M, u)$.

For any ordered matching M and any two indices i and j in M, we define M_i^j as $\mathit{matching}(U - U[i], V - V[j])$, where U denotes $\mathit{bids}(M)$ and V denotes $\mathit{items}(M)$.

Let M be a nonempty ordered matching, let U denote $\mathit{bids}(M)$, and let V denote $\mathit{items}(M)$. Then we define $\Delta_L(M)$ as $w(M_1^{|M|}) - w(M)$, and we define $\Delta_R(M)$ as $w(M_{|M|}^1) - w(M)$. It is straightforward to see that $\Delta_L(M[i:j])$ and $\Delta_R(M[i:j])$ can be computed for any $1 \le i \le j \le |M|$ by the recurrences

$$\Delta_L(M[k-1:j]) = \Delta_L(M[k:j]) + w(U[k], V[k-1]) - w(U[k-1], V[k-1]) \tag{L1}$$
$$\Delta_R(M[i:k+1]) = \Delta_R(M[i:k]) + w(U[k], V[k+1]) - w(U[k+1], V[k+1]) \tag{R1}$$

with base cases $\Delta_L(M[j]) = -w(U[j], V[j])$ and $\Delta_R(M[i]) = -w(U[i], V[i])$.

Let M be a nonempty ordered matching. Letting W denote $\max_{1 \le i \le |M|} w(M_i^{|M|})$, we define $\Delta_L^*(M)$ as $W - w(M)$, and we define $\mathit{loser}_L(M)$ as $\max\left\{i \mid w(M_i^{|M|}) = W\right\}$. Symmetrically, letting W' denote $\max_{1 \le i \le |M|} w(M_i^1)$, we define $\Delta_R^*(M)$ as $W' - w(M)$, and we define $\mathit{loser}_R(M)$ as $\max\left\{i \mid w(M_i^1) = W'\right\}$. By Lemma 1 and by the definitions of $\Delta_L(M)$ and $\Delta_R(M)$, it is straightforward to see that $(\Delta_L^*(M), \mathit{loser}_L(M)) = \max_{1 \le i \le |M|}(\Delta_L(M[i:]), i)$ and $(\Delta_R^*(M), \mathit{loser}_R(M)) = \max_{1 \le i \le |M|}(\Delta_R(M[:i]), i)$ (the pairs compare lexicographically). Hence, $\Delta_L^*(M[i:j])$, $\mathit{loser}_L(M[i:j])$, $\Delta_R^*(M[i:j])$, and $\mathit{loser}_R(M[i:j])$ can be

Algorithm 1 A linear-time implementation of bid insertion. The difference of the weight of an MWMCM of the UDALEW $A = (\mathit{bids}(M) + u, \mathit{items}(M))$ and that of M is equal to δ, and the maximum bid in $\mathit{bids}(M) + u$ that is unmatched in some MWMCM of A is u^*.

```
Input: $M$ is an ordered matching and $u$ is a bid that does not belong to $\mathit{bids}(M)$.
Output: $\mathit{insert}(M, u)$.
 1: Let $U$ denote $\mathit{bids}(M)$ and let $V$ denote $\mathit{items}(M)$
 2: $C \leftarrow \{(0, u)\}$
 3: $k \leftarrow \mathit{index}(u, U + u)$
 4: if $k > 1$ then
 5:     for $i = k - 1$ down to 1 do
 6:         Compute $\Delta_L(M[i : k-1])$ via (L1)
 7:         Compute $\Delta_L^*(M[i : k-1])$ and $\mathit{loser}_L(M[i : k-1])$ via (L2)
 8:     end for
 9:     $C \leftarrow C + (w(u, V[k-1]) + \Delta_L^*(M[\,: k-1]), U[i])$ where $i = \mathit{loser}_L(M[\,: k-1])$
10: end if
11: if $k \le |M|$ then
12:     for $i = k$ to $|M|$ do
13:         Compute $\Delta_R(M[k : i])$ via (R1)
14:         Compute $\Delta_R^*(M[k : i])$ and $\mathit{loser}_R(M[k : i])$ via (R2)
15:     end for
16:     $C \leftarrow C + (w(u, V[k]) + \Delta_R^*(M[k :\,]), U[j])$ where $j = \mathit{loser}_R(M[k :\,]) + k - 1$
17: end if
18: $(\delta, u^*) \leftarrow$ the lexicographically maximum pair in $C$
19: return $\mathit{matching}(U + u - u^*, V)$
```

computed for any $1 \le i \le j \le |M|$ by the recurrences

$$(\Delta_L^*(M[k-1 : j]), \mathit{loser}_L(M[k-1 : j])) = \max\{(\Delta_L^*(M[k : j]), \mathit{loser}_L(M[k : j]) + 1), (\Delta_L(M[k-1 : j]), 1)\} \tag{L2}$$

$$(\Delta_R^*(M[i : k+1]), \mathit{loser}_R(M[i : k+1])) = \max\{(\Delta_R^*(M[i : k]), \mathit{loser}_R(M[i : k])), (\Delta_R(M[i : k+1]), k + 2 - i)\} \tag{R2}$$

with base cases $\Delta_L^*(M[j]) = -w(U[j], V[j])$, $\Delta_R^*(M[i]) = -w(U[i], V[i])$, and $\mathit{loser}_L(M[j]) = \mathit{loser}_R(M[i]) = 1$.

▶ **Lemma 6.** *Let M be a nonempty ordered matching, let U denote $\mathit{bids}(M)$, let V denote $\mathit{items}(M)$, let u be a bid that does not belong to U, let k denote $\mathit{index}(u, U + u)$, let M_L denote $\mathit{insert}_L(M, u)$, and let M_R denote $\mathit{insert}_R(M, u)$. If $k > 1$, then M_L is equal to $M_i^{k-1} + (u, V[k-1])$ and $w(M_L) = w(M) + \Delta_L^*(M[\,: k-1]) + w(u, V[k-1])$ where i denotes $\mathit{loser}_L(M[\,: k-1])$; otherwise, $M_L = M$. If $k \le |M|$, then M_R is equal to $M_j^k + (u, V[k])$ and $w(M_R) = w(M) + \Delta_R^*(M[k :\,]) + w(u, V[k])$ where j denotes $\mathit{loser}_R(M[k :\,]) + k - 1$; otherwise, $M_R = M$.*

Lemmas 5 and 6, together with (L1), (R1), (L2), and (R2), directly suggest a linear-time computation of $\mathit{insert}(M, u)$, as shown in Algorithm 1. If $\mathit{insert}_L(M, u)$ (resp., $\mathit{insert}_R(M, u)$) is not equal to M, then the algorithm computes the difference $w(\mathit{insert}_L(M, u)) - w(M)$ (resp., $w(\mathit{insert}_R(M, u)) - w(M)$) and adds a pair at line 9 (resp., line 16) to a set C where the first component is this difference, and the second component is the bid in $\mathit{bids}(M) + u$ that is left unassigned by $\mathit{insert}_L(M, u)$ (resp., $\mathit{insert}_R(M, u)$). Then by Lemma 5, the algorithm correctly returns $\mathit{insert}(M, u)$ by choosing the maximum pair of C at line 18.

5 A Superblock-Based Bid Insertion Algorithm

In this section, we describe an ordered matching data structure based on the concept of a "superblock", and we show how to use this data structure to obtain a significantly faster bid insertion algorithm than that presented in Sect. 4. Before beginning our formal presentation in Sect. 5.1, we provide a high-level overview of the main ideas.

Recall that an ordered matching data structure maintains a dynamic ordered matching M. Let n denote $|M|$. We maintain a partition of the bids of M into contiguous "groups" of size $\Theta(\ell)$, where ℓ is a parameter to be optimized later. The time complexity of Alg. 1 is linear because the **for** loops starting at lines 5 and 12 process bid-item pairs in M sequentially. Our rough plan is to accelerate the computations associated with this pair of loops by proceeding group-by-group. We can process a group in constant time if we are given six "auxiliary values" that depend on the "submatching" M' of M associated with the bids in the group, namely: $\Delta_L(M')$, $\Delta_R(M')$, $\Delta^*_L(M')$, $\Delta^*_R(M')$, $\mathit{loser}_L(M')$, and $\mathit{loser}_R(M')$. The auxiliary values associated with a group can be computed in $\Theta(\ell)$ time. A natural approach is to precompute these auxiliary values when a group is created or modified, or when the set of matched items associated with the group is modified. Unfortunately, a single bid insertion can cause each bid in a contiguous interval of $\Theta(n)$ bids to have a new matched item. For example, if a bid insertion introduces a "low" bid u and deletes a "high" bid u', then each bid between u and u' gets a new matched item one position to the right of its old matched item. Since a constant fraction of the groups might need to have their auxiliary values recomputed as a result of a bid insertion, the overall time complexity remains linear.

The preceding discussion suggests that it might be useful to have an efficient way to obtain the new auxiliary values of a group of bids when the corresponding interval of matched items is shifted left or right by one position. To this end, we enhance the precomputation associated with a group of bids as follows: Instead of precomputing only the auxiliary values corresponding to the group's current matched interval of items, we precompute the auxiliary values associated with shifts of $0, \pm 1, \pm 2, \ldots, \pm\Theta(\ell)$ positions around the current matched interval. That way, unless a group of bids is modified (e.g., due to a bid being deleted or inserted) we do not need to redo the precomputation with the group until it has been shifted $\Omega(\ell)$ times. Since the enhanced precomputation computes $\Theta(\ell)$ sets of auxiliary values instead of one set, a naive implementation of the enhanced precomputation has $\Theta(\ell^2)$ time complexity, leading once again to linear worst-case time complexity for bid insertion. We obtain a faster bid insertion algorithm by showing how to perform the enhanced precomputation in $O(\ell \log^2 \ell)$ time.

Our $O(\ell \log^2 \ell)$-time algorithm for performing the enhanced precomputation forms the core of our fast bid insertion algorithm. Here we briefly mention the main techniques used to perform the enhanced precomputation efficiently; the reader is referred to [2, Sect. 5.3.1] for further details. A divide-and-conquer approach is used to compute the auxiliary values associated with the functions loser_L and loser_R in $O(\ell \log \ell)$ time; the correctness of this approach is based on a monotonicity result (see [2, Lemmas 8 and 9]). A convolution-based approach is used to compute the auxiliary values based on Δ_L and Δ_R in $O(\ell \log \ell)$ time (see [2, Lemma 7]). The auxiliary values based on loser_L (resp., loser_R) are used within a divide-and-conquer framework to compute the auxiliary values based on Δ^*_L (resp., Δ^*_R); in the associated recurrence, the overhead term is dominated by the cost of evaluating the same kind of convolution as in the computation of the auxiliary values based on Δ_L and Δ_R. As a result, the overall time complexity for computing the auxiliary values based on Δ_L^* and Δ_R^* is $O(\ell \log^2 \ell)$.

Section 5.1 introduces the concept of a "block", which is used to represent a group of bids together with a contiguous interval of items that includes all of the items matched to the group. Section 5.3 presents a block data structure. When a block data structure is "initialized" with a group of bids and an interval of items, the enhanced precomputation discussed in the preceding paragraph is performed, and the associated auxiliary values are stored in tables. A handful of "fields" associated with the block are also initialized; these fields store basic information such as the number of bids or items in the block. After initialization, the block data structure is read-only: Whenever a block needs to be altered (e.g., because a bid needs to be inserted/deleted, because the block needs to be merged with an adjacent block), we destroy the block and create a new one. The operations supported by a block may be partitioned into three categories: "queries", "lookups" and "scans". Each query runs in constant time and returns the value of a specific field. Each lookup runs in constant time and uses a table lookup to retrieve one of the precomputed auxiliary values. Each of the two linear-time scan operations (one leftgoing, one rightgoing) performs a naive emulation of one of the **for** loops of Alg. 1; in the context of a given bid insertion, such operations are only invoked on the block containing the insertion position of the new bid.

Section 5.1 defines the concept of a superblock, which is used to represent an ordered matching as a sequence of blocks. A superblock-based ordered matching data structure is introduced in Sect. 5.3, where each of the constituent blocks is represented using the block data structure alluded to in the preceding paragraph. In Sect. 5.3, we simplify the presentation by setting the parameter ℓ to $\Theta(\sqrt{n})$. For this choice of ℓ, we show that bid insertion can be performed using $O(1)$ block initializations, $O(\sqrt{n})$ block queries, $O(\sqrt{n})$ block lookups, at most two block scans, and $O(\sqrt{n})$ additional overhead, resulting in an overall time complexity of $O(\sqrt{n} \log^2 n)$. In terms of the parameters ℓ and n, the approach of Sect. 5.3 can be generalized to perform bid insertion using $O(\lceil n/\ell^2 \rceil)$ block initializations, $O(n/\ell)$ block queries, $O(n/\ell)$ block lookups, at most two block scans, and $O(n/\ell)$ additional overhead; it is easy to verify that setting ℓ to $\Theta(\sqrt{n})$ minimizes the overall time complexity.

5.1 Blocks and Superblocks

We define a *block* B as a UDALEW (U, V) where $|U| \leq |V|$. For any block $B = (U, V)$, we define *shifts*(B) as $|V| - |U| + 1$. For any block $B = (U, V)$ and any integer t such that $1 \leq t \leq$ *shifts*(B), we define *matching*(B, t) as *matching*$(U, V[t : t + |U| - 1])$.

Let M be a nonempty ordered matching, let U denote *bids*(M), and let V denote *items*(M). Let m be a positive integer, and let $\langle a_0, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$, and $\langle c_1, \ldots, c_m \rangle$ be sequences of integers such that $a_0 = 0$, $a_m = |U|$, and $1 \leq b_i \leq a_{i-1} + 1 \leq a_i \leq c_i \leq |U|$ for $1 \leq i \leq m$. Let B_i denote the block $(U[a_{i-1} + 1 : a_i], V[b_i : c_i])$ for $1 \leq i \leq m$. Then the list of blocks $S = \langle B_1, \ldots, B_m \rangle$ is a *superblock*, and we make the following additional definitions: *matching*(S) denotes M; *size*(S) denotes $|M|$; *bids*(S) denotes U; *items*(S) denotes V; *shift*(S, i) and *shift*(S, B_i) both denote $b_i - a_{i-1}$ for $1 \leq i \leq m$; *sum*(S, i) denotes a_i for $0 \leq i \leq m$; the leftmost block B_1 and the rightmost block B_m are the *boundary blocks*, the remaining blocks $B_2, \ldots, B_{m-1}$ are the *interior blocks*. Remark: For any superblock S, *matching*$(S) = \bigcup_{1 \leq i \leq |S|}$ *matching*$(S[i],$ *shift*$(S, i))$.

5.2 Algorithm 2

We obtain a significantly faster bid insertion algorithm than Alg. 1 by accelerating the computations associated with the **for** loops starting at lines 5 and 12. Recall that the first loop computes $\Delta_L^*(M[\,:k-1])$ and $loser_L(M[\,:k-1])$, and the second one computes $\Delta_R^*(M[k:\,])$ and $loser_R(M[k:\,])$. These two loops process a trivial representation of M pair-by-pair using the recurrences (L1), (R1), (L2), and (R2). We start by generalizing these recurrences; these generalizations allow us to compute the aforementioned values more efficiently by looping over a superblock-based representation of the matching block-by-block, instead of pair-by-pair.

Let M denote $matching(U, V)$, and let i, j, and k be three indices in M such that $i \leq j < k$. Then the following equation generalizes (L1), and it is straightforward to prove by repeated application of (L1).

$$\Delta_L(M[i:k]) = \Delta_L(M[j+1:k]) + w(U[j+1], V[j]) + \Delta_L(M[i:j]). \tag{L1$'$}$$

We also give a generalization of (L2), where the proof follows from the definitions of Δ_L^* and $loser_L$.

$$(\Delta_L^*(M[i:k]), loser_L(M[i:k])) = \\ \max \begin{Bmatrix} (\Delta_L^*(M[j+1:k]), loser_L(M[j+1:k]) + j + 1 - i), \\ (\Delta_L^*(M[i:j]) + w(U[j+1], V[j]) + \Delta_L(M[j+1:k]), loser_L(M[i:j])) \end{Bmatrix} \tag{L2$'$}$$

Symmetric equations generalizing (R1) and (R2) are given in [2].

We use (L1′) and (L2′) within a loop that iterates over a superblock-based representation of the matching block-by-block. In each iteration of the loop, we are able to evaluate the right-hand side of (L1′) and (L2′) in constant time because the terms involving $M[j+1:k]$ are carried over from the previous iteration, and the terms involving $M[i:j]$ are already stored in precomputed tables associated with the blocks of the superblock.

The high-level algorithm is given in Alg. 2. The input is a superblock S that represents an ordered matching, denoted M (i.e., $matching(S) = M$), and a bid u that does not belong to $bids(S)$. The output is a superblock representing $insert(M, u)$. The unique bid u^* that is unmatched in $insert(M, u)$ is identified using the block-based framework alluded to above. After identifying u^*, if $u^* \neq u$, the algorithm invokes a subroutine $\textsc{Swap}(S, u^*, u)$ which, given a superblock S, a bid u^* that belongs to $bids(S)$, and a bid u that does not belong to $bids(S)$, returns a superblock that represents $matching(bids(S) + u - u^*, items(S))$. The correctness of Alg. 2 is established in [2, Lemma 6], where it is shown that Alg. 2 emulates the behavior of Alg. 1.

5.3 Fast Implementation of Algorithm 2

In this section, we first present a block data structure that precomputes the auxiliary tables mentioned in Sect. 5.2 in quasilinear time, thus allowing lines 13 and 14 of Alg. 2 to be performed in constant time. We then introduce a superblock-based ordered matching data structure that stores the blocks using the block data structure, where the sizes of the blocks are optimized to balance the cost of SWAP with that of the remaining operations in Alg. 2. We present our efficient implementation of SWAP, which constructs only a constant number of blocks, and analyze its time complexity in [2, Sections 5.3.3 and 5.3.4].

Let S be a superblock on which a bid insertion is performed, let B be a block in S, and let M_t denote $matching(B, t)$ for $1 \leq t \leq shifts(B)$. The algorithm may query $\Delta_L(M_t)$,

Algorithm 2 A high-level bid insertion algorithm using the superblock-based representation of an ordered matching.

```
Input: $S$ is a superblock and $u$ is a bid that does not belong to $bids(S)$.
Output: A superblock $S'$ such that $matching(S') = insert(matching(S), u)$.
1: Let $M$ denote $matching(S)$, let $U$ denote $bids(S)$, and let $V$ denote $items(S)$
2: Let $S[i]$ be $(U_i, V_i)$ for $1 \leq i \leq |S|$
3: $\sigma(i) \leftarrow sum(S, i)$ for $0 \leq i \leq |S|$
4: $C \leftarrow \{(0, u)\}$
5: $\ell \leftarrow |\{(U', V') \mid (U', V') \in S$ and $U'[1] < u\}|$
6: $k \leftarrow$ if $\ell < 1$ then $1$ else $index(u, U_\ell + u) + 1 + \sigma(\ell - 1)$
7: if $k > 1$ then
8:     for $i = k - 1$ down to $\sigma(\ell - 1) + 1$ do
9:         Compute $\Delta_L(M[i : k - 1])$ via (L1)
10:        Compute $\Delta^*_L(M[i : k - 1])$ and $loser_L(M[i : k - 1])$ via (L2)
11:    end for
12:    for $i = \ell - 1$ down to $1$ do
13:        Compute $\Delta_L(M[\sigma(i - 1) + 1 : k - 1])$ via (L1′)
14:        Compute $\Delta^*_L(M[\sigma(i - 1) + 1 : k - 1])$ and $loser_L(M[\sigma(i - 1) + 1 : k - 1])$ via (L2′)
15:    end for
16:    $C \leftarrow C + (w(u, V[k - 1]) + \Delta^*_L(M[\,: k - 1]), U[i])$ where $i = loser_L(M[\,: k - 1])$
17: end if
18: if $k \leq |M|$ then
19:    Compute $\Delta^*_R(M[k :\,])$ and $loser_R(M[k :\,])$. See [2] for the code, which is symmetric to lines 8 through 15.
20:    $C \leftarrow C + (w(u, V[k]) + \Delta^*_R(M[k :\,]), U[j])$ where $j = loser_R(M[k :\,]) + k - 1$
21: end if
22: $(\delta, u^*) \leftarrow$ the lexicographically maximum pair in $C$
23: return if $u^* \neq u$ then SWAP$(S, u^*, u)$ else $S$
```

$\Delta_R(M_t)$, $\Delta^*_L(M_t)$, $\Delta^*_R(M_t)$, $loser_L(M_t)$, and $loser_R(M_t)$ for $t = shift(S, B)$. If B is part of the superblocks for a series of bid insertions, then these queries may be performed for various t values. For a fast implementation of Alg. 2, instead of individually computing these quantities at query time, we efficiently precompute them during the construction of the block and store them in the following six lists. We define $\Delta_L(B)$ as the list of size $shifts(B)$ such that $\Delta_L(B)[t]$ is equal to $\Delta_L(M_t)$ for $1 \leq t \leq shifts(B)$. We define the lists $\Delta_R(B)$, $\Delta^*_L(B)$, $\Delta^*_R(B)$, $loser_L(B)$, and $loser_R(B)$ similarly. The representation of a block $B = (U, V)$ simply maintains each of the following explicitly as an array: U, V, $\Delta_L(B)$, $\Delta_R(B)$, $\Delta^*_L(B)$, $\Delta^*_R(B)$, $loser_L(B)$, and $loser_R(B)$. In what follows, we refer to that representation as the *block data structure for B*.

The main technical contribution of this paper is that we can compute the aforementioned lists efficiently as stated in the following theorem.

▶ **Theorem 7.** *The block data structure can be constructed in $O(|V| (\log shifts(B) + \log^2 |U|))$ time for any block $B = (U, V)$.*

We now introduce a data structure called a *superblock-based ordered matching* (*SOM*); the formal definition is deferred to [2, Sect. 5.3.2]. A SOM represents an ordered matching

M by maintaining a superblock S such that $\mathit{matching}(S) = M$, where S is stored as a list of block data structures.

▶ **Theorem 8.** *The SOM has initialization cost $O(n \log^2 n)$, bid insertion cost $O(\sqrt{n} \log^2 n)$, and dump cost $O(n)$.*

Theorem 8 states the main result of our paper, and is proved in [2, Sect. 5.3.4]. Here we briefly mention key performance-related properties of the SOM, deferring the details to [2, Sect. 5.3.2]. It is easy to see that Alg. 2 does not modify the superblock, except during Swap at line 23. When Swap modifies the superblock, existing blocks are not modified; rather, some existing blocks are deleted, and some newly constructed blocks are inserted. We define the blocks in a SOM so that each block has $\Theta(\sqrt{n})$ bids and $\Theta(\sqrt{n})$ items, and so that Swap can be implemented by constructing at most a constant number of blocks, where n denotes the size of the matching represented by the SOM; we give an $O(\sqrt{n} \log^2 n)$-time implementation of Swap in [2, Sections 5.3.3 and 5.3.4].

It is possible to support constant-time queries that return the bid matched to a given item with some additional bookkeeping. Queries to find whether a bid is matched or not, and if so, to return the matched item, can be implemented in logarithmic time by performing binary search. Finally, it is possible to initialize the SOM with a matching consisting of all dummy bids, each with intercept zero and slope zero, in linear time, since all of the weights involving those bids are zero, and thus it is trivial to construct the blocks.

References

1 G. Demange, D. Gale, and M. A. O. Sotomayor. Multi-item auctions. *The Journal of Political Economy*, 94:863–872, 1986.

2 N. O. Domaniç, C.-K. Lam, and C. G. Plaxton. Bipartite matching with linear edge weights. Technical Report TR–16–15, Department of Computer Science, University of Texas at Austin, October 2016.

3 N. O. Domaniç and C. G. Plaxton. Scheduling unit jobs with a common deadline to minimize the sum of weighted completion times and rejection penalties. In *Proceedings of the 25th International Symposium on Algorithms and Computation*, pages 646–657, 2014.

4 R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1413–1424, 2012.

5 M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, 1987.

6 H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30:209–221, 1985.

7 F. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistics Quarterly*, 14:313–316, 1967.

8 J. Green and J.-J. Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45:427–438, 1977.

9 G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities.* Cambridge University Press, 2nd edition, 1952.

10 I. Katriel. Matchings in node-weighted convex bipartite graphs. *INFORMS Journal on Computing*, 20:205–211, 2008.

11 H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

12 H. B. Leonard. Elicitation of honest preferences for the assignment of individuals to positions. *The Journal of Political Economy*, 91:461–479, 1983.

13 W. Lipski, Jr. and F. P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15:329–346, 1981.

14 C. G. Plaxton. Vertex-weighted matching in two-directional orthogonal ray graphs. In *Proceedings of the 24th International Symposium on Algorithms and Computation*, pages 524–534, 2013.

15 G. Steiner and J. S. Yeomans. A linear time algorithm for maximum matchings in convex, bipartite graphs. *Computers and Mathematics with Applications*, 31:91–96, 1996.

Raising Permutations to Powers in Place*

Hicham El-Zein[1], J. Ian Munro[2], and Matthew Robertson[3]

1 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
`helzein@uwaterloo.ca`

2 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
`imunro@uwaterloo.ca`

3 **Cheriton School of Computer Science, University of Waterloo, Ontario, Canada**
`m32rober@uwaterloo.ca`

Abstract

Given a permutation of n elements, stored as an array, we address the problem of replacing the permutation by its k^{th} power. We aim to perform this operation quickly using $o(n)$ bits of extra storage. To this end, we first present an algorithm for inverting permutations that uses $O(\lg^2 n)$ additional bits and runs in $O(n \lg n)$ worst case time. This result is then generalized to the situation in which the permutation is to be replaced by its k^{th} power. An algorithm whose worst case running time is $O(n \lg n)$ and uses $O(\lg^2 n + \min\{k \lg n, n^{3/4+\epsilon}\})$ additional bits is presented.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Algorithms, Combinatorics, Inplace, Permutations, Powers

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.29

1 Introduction

Permutations are fundamental in computer science and are the subject of extensive study. They are commonly used as a basic building block for space efficient encoding of strings [1, 8, 12, 14], binary relations [3, 2], integer functions [11] and many other combinatorial objects.

In this paper, we study the problem of transforming a permutation π to its k^{th} power π^k in place. By "in place," we mean that the algorithm executes while using "very little" extra space. Ideally, we want the algorithm to use only a polylogarithmic number of additional bits. The algorithm we present uses several new techniques that are of interest in their own right and could find broader applications.

One interesting application of inverting a permutation in place was encountered in the content of data ware-housing by a Waterloo company [4]. Under specific indexing schemes, the permutation corresponding to the rows of a relation sorted by any given key is explicitly stored. To perform certain joins, the inverse of a segment of the permutation is precisely what is needed. This permutation occupies a substantial portion of the space used by the indexing structure. Doubling this space requirement, to explicitly store the inverse of the permutation, for the sole purpose of improving the time to compute certain joins may not be practical, and indeed was not in the work leading to [4].

Since there are $n!$ permutations of length n, the number of bits required to represent a permutation is $\lceil \lg(n!) \rceil \sim n \lg n - n \lg e + O(\lg n)$ bits.[1] Munro et al. [11] studied the space efficient representation of general permutations where general powers of individual elements

[*] This work was sponsored by the NSERC of Canada and the Canada Research Chairs Program.

[1] We use $\lg n$ to denote $\log_2 n$

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 29; pp. 29:1–29:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can be computed quickly. They gave a representation taking the optimal $\lceil \lg(n!) \rceil + o(n)$ bits, that can compute the image of a single element of $\pi^k()$ in $O(\lg n / \lg\lg n)$ time; and a representation taking $(1+\epsilon)n \lg n$ bits where $\pi^k()$ can be computed in constant time. The preprocessing for these representations as presented in [11] requires an extra $O(n)$ words of space, so a solution that involves building them as an intermediate step will not be considered inplace and therefore does not apply to our current problem. For further details on permutation representations see [6, 10, 5].

Throughout this paper, we assume that the permutation is stored in an array $A[1,\dots,n]$ of n words. The array originally contains the values $\pi(1),\dots,\pi(n)$, then, afterwards, it contains the values $\pi^k(1),\dots,\pi^k(n)$. Storing A requires $n\lceil \lg n \rceil = n \lg n + n(\lceil \lg n \rceil - \lg n)$ bits. When $(\lceil \lg n \rceil - \lg n)$ is "big," we can reduce the space required by this representation by encoding a constant number c of consecutive elements into a single object. This object is essentially the c digits, base n number $\pi[i]\pi[i+1]\dots\pi[i+c-1]$. Encoding these n/c objects of size $\lceil c \lg n \rceil$ bits each, totals to $n \lg n + n/c$ bits. To decode a value, we need a constant number of arithmetic operations. This saving of memory at the cost of c accesses to interpret one element of A carries through all of our work.

This paper is organized as follows. In Section 2, we review previous work on permuting data in place [7], on which we base our work. In Section 3, we start by presenting an algorithm for inverting permutations that uses $O(b + \lg n)$ additional bits and runs in $O(n^2/b)$ worst case time. Using a different approach, we improve the worst case time complexity to $O(n \lg n)$, but using $O(\sqrt{n} \lg n)$ additional bits. This development then leads to our main algorithm for inverting permutations, we achieve an algorithm with a worst case time complexity of $O(n \lg n)$ using only $O(\lg^2 n)$ additional bits. Then we face the problem that while $\pi^{-1}()$ leaves the cycle structure as it was, higher powers may create more (smaller) cycles. This causes further difficulty which is addressed in Section 4 where we generalize the algorithm from Section 3 to the situation in which the permutation is to be replaced by its k^{th} power. An algorithm whose worst case running time is $O(n \lg n)$ and uses $O(\lg^2 n + \min\{k \lg n, n^{3/4+\epsilon}\})$ additional bits is presented. Our solution relies on Rubinstein's [13] work on finding factorizations into small terms modulo a parameter. The final result can be improved if better factorization is applied. However, we show that obtaining a better factorization is probably difficult since it would imply Vinogradov's conjecture [15]. We conclude our work in Section 5.

2 Background and Related Work

Fich et al. studied the problem of permuting external data according to a given permutation, in place [7]. That is, given an array B of length n and a permutation π given by an oracle or read only memory, rearrange the elements of B in place according to π.

It is not sufficient to simply assign $B[\pi(i)] \leftarrow B[i]$ for all $i \in \{1,\cdots,n\}$, because an element in B may have been modified before it has been accessed. A permutation can be thought of as a collection of disjoint cycles. The procedure ROTATE, rotates the values in B according to π by calling ROTATECYCLE on the leader of each cycle. A cycle leader is a uniquely identifiable position in each cycle. The smallest position in a cycle, or *min leader*, is a simple example of a cycle leader.

The problem is to identify a position as leader by starting at that position and traversing only forward along the cycle. Choosing the min leader would take $\Theta(n^2)$ value inspections in the worst case. A leader that we call the *local min leader* can be used to permute data in $O(n \lg n)$ worst case time complexity using only $O(\lg^2 n)$ additional bits [7]. As stated in [7], the local min leaders of a permutation π are characterized as follows. Let $E_1 = \{1,\dots,n\}$

```
procedure ROTATE($B$)
    for $i \leftarrow 0$ to $n-1$ do
        if ISLEADER($i$) then
            ROTATECYCLE($B$, $i$)
```

```
procedure ROTATECYCLE($B$, $leader$)
    $i \leftarrow \pi(leader)$
    while $i \neq leader$ do
        SWAP($B[i]$, $B[leader]$)
        $i \leftarrow \pi(i)$
```

Figure 1 Rotates the values in B according to a permutation π.

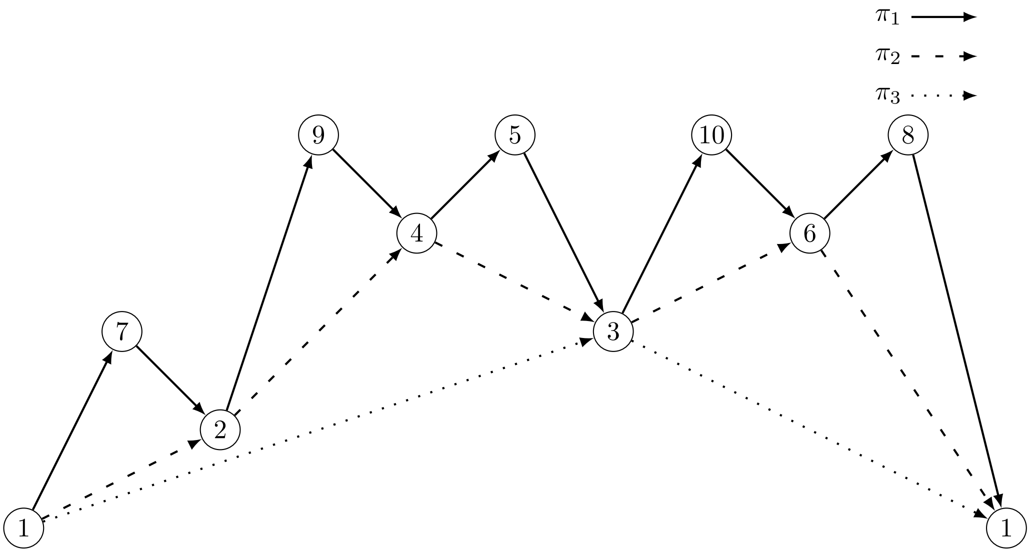


Figure 2 An illustration of π_i.

and $\pi_1 = \pi$. For positive integers $r > 1$, define E_r as the set of local minima in E_{r-1} encountered following the cycle representation of the permutation π_{r-1} and define π_r as the permutation that maps each element of E_r to the next element of E_r that is encountered following π_{r-1}. More formally, $E_r = \{i \in E_{r-1} | \pi_{r-1}^{-1}(i) > i < \pi_{r-1}(i)\}$ and $\pi_r : E_r \to E_r$ is defined such that $\pi_r(i) = \pi_{r-1}^m(i)$ where $m = \min\{m > 0 | \pi_{r-1}^m(i) \in E_r\}$. Since at most half the elements in each cycle are local minima, $|E_r| < |E_{r-1}|/2$ and $r \leq \lg n$. The leader of a cycle is the unique position i, such that $\pi_{r-1} \ldots \pi_1(i) \in E_r$. For example, if $\pi = (1\ 7\ 2\ 9\ 4\ 5\ 3\ 10\ 6\ 8)$ as illustrated in Figure 2 (similar to Figure 6 in [7]), then

$$\begin{aligned}
E_1 &= \{1,2,3,4,5,6,7,8,9,10\}, & \pi_1 &= (1\ 7\ 2\ 9\ 4\ 5\ 3\ 10\ 6\ 8)\\
E_2 &= \{1,2,4,3,6\}, & \pi_2 &= (1\ 2\ 4\ 3\ 6)\\
E_3 &= \{1,3\}, & \pi_3 &= (1\ 3)\\
E_4 &= \{1\}, & \pi_4 &= (1)
\end{aligned}$$

The local min leader of the only cycle in π is the position 9 since $\pi_3\pi_2\pi_1(9) = 1$.

The procedure ISLOCALMINLEADER (see Figure 3), checks if position i in the permutation is the local min leader of his cycle. It has the property of proceeding at most $4n$ steps on the permutation for a single element, and a total of $O(n \lg n)$ steps on the permutation for all elements. We treat the local min leader technique as a black box. There are a few occasions where we need details so we provide the procedure to make this paper more self contained. We refer the reader to [7] for further details on this procedure.

```
procedure IsLocalMinLeader($i$)
    $elbow[0] \leftarrow elbow[1] \leftarrow i$
    for $r \leftarrow 1, 2, \ldots$ do
        //loop invariant:
        $\{elbow[r] = \pi_{r-1} \ldots \pi_1(i)\}$
        Next($r$)
        if $elbow[r] > elbow[r-1]$ then
            $elbow[r] \leftarrow elbow[r-1]$
            Next($r$)
            if $elbow[r] > elbow[r-1]$ then
                return $false$
            $elbow[r+1] \leftarrow elbow[r]$
      else if $elbow[r] = elbow[r-1]$ then
            return $true$
```

```
procedure Next($r$)
    if $r = 1$ then
        $elbow[0] \leftarrow \pi(elbow[1])$
    else
        while $elbow[r-1] < elbow[r-2]$ do
            $elbow[r-1] \leftarrow elbow[r-2]$
    Next($r-1$)
        while $elbow[r-1] > elbow[r-2]$ do
            $elbow[r-1] \leftarrow elbow[r-2]$
    Next($r-1$)
```

Figure 3 Checks if index i is a local min leader.

```
procedure InvertCycle($A$, $leader$)
    $current \leftarrow A[leader]$
    $previous \leftarrow leader$
    while $current \neq leader$ do
        $next \leftarrow A[current]$
        $A[current] \leftarrow previous$
        $previous \leftarrow current$
        $current \leftarrow next$
    $A[leader] \leftarrow previous$
```

Figure 4 Inverts a permutation.

3 Inverting Permutations

To invert a permutation we can use the structure of the algorithm described in Figure 1, but invert the cycles instead of rotating the data. Figure 4 shows how to invert a cycle. The algorithm iterates over the permutation, and inverts each cycle only on its leader. A cycle leader must be used that will remain unchanged once the cycle is inverted. An example of such a cycle leader is the min leader.

Inverting a permutation using min leader will use $O(\lg n)$ additional bits and take $\Theta(n)$ time if the permutation consists of one large cycle in increasing order; or $\Theta(n^2)$ time if the permutation consists of one large cycle in decreasing order. We note that for a random cycle of length n this total cost would be about $n \lg n$. The analysis is similar to the bidirectional distributed algorithm for finding the smallest of a set of n uniquely numbered processors arranged in a circle [9]. However, our interest is in finding algorithms with good worst case performance.

A permutation can be inverted in linear time using a n-bit vector. The vector can be used to mark corresponding positions in π as their cycles are inverted. This is equivalent to using the min leader, but takes $n + O(\lg n)$ additional bits.

Using a technique presented in [7], the bit vector can be shrunk to b-bits by conceptually dividing the permutation into $\lceil n/b \rceil$ sections each of size b (except possibly the last section will be smaller). The b-bit vector is reset at the start of each section and is used to keep track of which positions are encountered in the section being processed. If the position

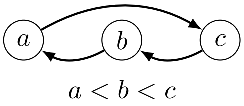


Figure 5 An example of a bad cycle.

under consideration for being a cycle leader has a corresponding bit with value 0, its cycle is traversed searching for a smaller position. If no smaller position is found, then the position is a cycle leader and the cycle is inverted. On the other hand, if the position under consideration has a corresponding bit with value 1, then the position was previously encountered as part of a cycle containing a smaller position in the section, and hence is not a cycle leader. Each cycle will be traversed at most n/b times, thus the total runtime is n^2/b and the space used is $b + O(\lg n)$.

▶ **Theorem 1.** *In the worst case, the array representation of a permutation of length n can be replaced with its own inverse in $O(n^2/b)$ time using $b + O(\lg n)$ additional bits of space.*

By setting $b = \sqrt{n}$ we get the following corollary.

▶ **Corollary 2.** *In the worst case, the array representation of a permutation of length n can be replaced with its own inverse in $O(n\sqrt{n})$ time using $O(\sqrt{n})$ additional bits of space.*

3.1 Inversion in $O(n \lg n)$ Time Using $O(\sqrt{n} \lg n)$ Bits

The local min leader of a cycle will, in general, change after the cycle has been inverted. Figure 5 shows a simple example of this: b is the leader of the cycle, but if it were inverted, c would become the leader. Since $c > b$, the algorithm in Figure 4 will invert the cycle once on b and then again on c because c will look like a leader when it is reached in the outer loop. Inverting the cycle the second time will undo the work of inverting it the first time. We will call a cycle with this problem a *bad cycle*.

▶ **Definition 3.** A *bad cycle* is a cycle with the property that if inverted, has a new cycle leader not yet processed, i.e., larger than the original leader.

It is not hard to build a permutation that will have $\Theta(n)$ bad cycles. Such a permutation could just repeat our bad cycle pattern and create exactly $\lfloor n/3 \rfloor$ bad cycles. So, there is not enough space to use even 1 bit to mark these cycles.

▶ **Theorem 4.** *A permutation π represented as an array can be replaced with π^{-1} in place using $O(\sqrt{n} \lg n)$ extra bits in $O(n \lg n)$ time.*

Proof. Although the permutation π can contain up to n cycles, the number of distinct cycle lengths in π, which we denote by k, is less than $\lfloor \sqrt{2n} \rfloor$ (since $\sum_{i=1}^{\lceil \sqrt{2n} \rceil} i > n$). We store these cycle lengths in an array L of size $O(\sqrt{n} \lg n)$ bits. This can be done in $O(n \lg n)$ time by iterating over the permutation and computing the length of every cycle as it is detected on its local min leader using the procedure IsLocalMinLeader (see Figure 3). After a length is detected, query a balanced binary search tree H to check if the length computed was already encountered; if it was not encountered, insert the new length to L and H. The cycle lengths are ranked according to their position in L.

If a position i is found to be the local min leader of a cycle α, then the minimum position in α is given by $x = \pi_{r-1} \dots \pi_1(i)$. Let $j = \pi_1 \dots \pi_{r-1}(x)$, then $x = \pi_{r-1}^{-1} \dots \pi_1^{-1}(j)$ and j is the local min leader of the inverse α^{-1} of α. When testing the position i for leadership, the procedure IsLocalMinLeader will store j in $elbow[0]$ upon termination because of its loop invariant (at the beginning of each iteration: $elbow[r] = \pi_{r-1} \dots \pi_1(i)$). Thus, we can identify the leader of α^{-1} while testing the leadership of position i without the need for testing each position in α^{-1}. A bad cycle can easily be identified by checking if $j > i$.

▶ **Definition 5.** A *tail* of a cycle is the position that points to its local min leader, i.e., if t is the tail of a cycle c with local min leader l, then $\pi(t) = l$.

The algorithm iterates over the permutation similar to the algorithm in Figure 4, and invert each cycle only on its local min leader. If a bad cycle α was detected, we modify the tail of the inverted cycle α^{-1} to point to the rank of the length of the cycle instead of back to the leader of the inverted cycle. Note that the tail ($\pi(elbow[0])$) can be found by probing $A[elbow[0]]$ before inverting the cycle.

When pointing to the ranks of the cycles length, we have to use values in the range of 1 to n, otherwise the size of each entry in A may increase to $\lceil \lg n \rceil + 1$ bits and we may end up using n additional bits. The problem now is that A does not distinguish between pointing to a cycle length rank, or pointing to a different position in the cycle. This can be solved with a table T of size $O(\sqrt{n} \lg n)$ bits that stores the positions of the permutation that point to its first k positions. T will initially store $\pi^{-1}(1), \dots, \pi^{-1}(k)$. It is set by initially traversing the permutation, then it is updated as cycles are inverted.

While testing for the leadership of a position i, if a position t is found such that $\pi(t) \leq k$, then t can be checked against T in $O(1)$ time to determine if $A[t]$ points to a cycle length rank or a position in the cycle. If it is the latter case, we simply continue. Else if it points to a cycle length rank, abort the procedure IsLocalMinLeader and do not invert the cycle. If the length traversed so far matches the cycle length stored in L at rank $A[t]$, then the position i is the local min leader of an already inverted cycle. Restore the cycle by setting $A[t] = i$.

The total time spent is $O(n \lg n)$, and the space used is $O(\sqrt{n} \lg n + \lg^2 n)$. ◀

3.2 Reducing Extra Space to $O(\lg^2 n)$ Bits

Next, we extend the approach presented in the previous subsection to achieve an algorithm for inverting permutations with $O(n \lg n)$ worst case time complexity while using only $O(\lg^2 n)$ bits. First we start with some definitions.

Given a permutation π, the *depth* of a position $e \in \pi$ is the maximum index d such that $\pi_{d-1} \dots \pi_2 \pi(e) \in E_d$.[2] For example, the depth of 10 in Figure 2 is 3 since $\pi_2\pi_1(10) = 1 \in E_3$ and $\pi_3\pi_2\pi_1(10) = 3 \notin E_4$. Let c be a cycle in π of size l with local min leader s_1. We define S_1 as the following sequence: $s_1, s_2, \dots, s_l$ where $s_i = \pi(s_{i-1})$ for $i > 1$; s_l is the tail of the cycle c. For $i > 1$, S_i is a subsequence of S_{i-1} formed by the local minima in S_{i-1} excluding S_{i-1}'s first and last elements. The *limited depth* of a position $e \in \pi$ is the maximum index d such that $\pi_{d-1} \dots \pi_2 \pi(e) \in S_d$. The values $s_1, \dots, s_{i-1}$ are not needed to evaluate the limited depth of s_i, but the values $s_i, \dots, s_l$ are required. The limited depth of a position is upper bounded by its depth. Notice that the first element in S_i is always $\pi_{i-1} \dots \pi(s_1)$, since s_1 is the local min leader of c. Moreover, the limited depth d of a cycle's local min leader is either unique or shared by at most one other element $\pi_1^{-1} \dots \pi_{d-1}^{-1}(\pi_d \dots \pi_2 \pi(s_1))$ in the cycle. The depth and limited depth of a position can be computed in a manner similar to the procedure IsLocalMinLeader with the same space and time complexity.

[2] For the definition of π_i where $i \in \{1, \dots, d\}$ check Section 2.

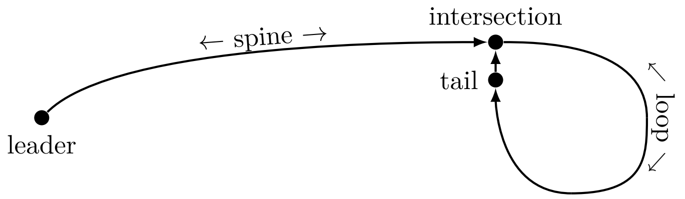


Figure 6 An example of a broken cycle.

We say that a cycle is *broken* if its tail points to a position other than its local min leader. We call this position the broken cycle's *intersection*. We define the *spine* to be the path from the leader to the intersection, and the *loop* to be the cycle containing the intersection and the tail. Figure 6 demonstrates these terms.

Following the algorithm described previously, when a cycle c is detected it is replaced by its inverse; if c is detected to be a bad cycle, the tail of c^{-1} is modified to store the limited depth of c^{-1}'s local min leader k. In that case, the tail of c^{-1} will be modified to point to the unique position whose limited depth is the same as k if that position was encountered before k, thus making c^{-1} a broken cycle. Finally, c^{-1} will be restored once k is encountered. As in the previous subsection, for A to distinguish between pointing to a limited depth, or pointing to a different position in the cycle we use a table T of size $O(\lg^2 n)$ bits that stores the positions of the permutation that point to its first $\lg n$ positions.

The algorithm iterates over the permutation. At each position i, it interleaves four scans $\mathscr{F}$, $\mathscr{L}$, $\mathscr{T}$ and $\mathscr{H}$. For every operation run on $\mathscr{F}$, a constant number of operations are run on $\mathscr{L}$; and for every operation run on $\mathscr{L}$ a constant number of operations are run on $\mathscr{T}$ and $\mathscr{H}$. $\mathscr{F}$ is used to determine whether i is the local min leader of its cycle (c or c^{-1}), $\mathscr{L}$ is used to determine the limited depth of i, and $\mathscr{T}$ and $\mathscr{H}$ are used to determine if i's cycle was broken, and to restore it. The $\mathscr{T}$ and $\mathscr{H}$ scans have two phases:

- The first phase is the classic tortoise and hare algorithm for cycle detection. It is used to check if i's cycle is broken. $\mathscr{T}$ (for tortoise) and $\mathscr{H}$ (for hare) both start at position i, $\mathscr{T}$ proceeds at one step per iteration and $\mathscr{H}$ proceeds at two steps until they meet at position j. Phase one will consist of no more than l iterations, where l is the length of i's cycle. This is because at each iteration, the forward distance (i.e. the distance from $\mathscr{H}$ to $\mathscr{T}$ traversing forward in the cycle) between the two pointers will decrease by one; or if the cycle was broken, the distance decreases once both pointers enter the broken cycle's loop. If one of the scans encounters a limited depth or if i is reachable from j, $\mathscr{T}$ and $\mathscr{H}$ are aborted while $\mathscr{F}$ and $\mathscr{L}$ continue. Otherwise, we know that the cycle is broken and we proceed to the second phase.
- The aim of the second phase is to find the tail of the broken cycle c^{-1}. Let λ be the length of c^{-1}'s loop, μ be the distance from i to c^{-1}'s intersection, and δ be the distance from the intersection to j. Denote by d_t and d_h the distance traveled by the pointers in $\mathscr{T}$ and $\mathscr{H}$ respectively. $d_t = \mu + \delta$ and $d_h = \mu + k\lambda + \delta$ where $k \in \mathbb{Z}^+$. We know

$$\begin{aligned} 2d_t &= d_h \\ 2(\mu + \delta) &= \mu + k\lambda + \delta \\ \mu &= k\lambda - \delta \ . \end{aligned}$$

Thus, if we reset $\mathscr{T}$'s pointer to position i, while $\mathscr{H}$ remains at j, and as in the first phase, $\mathscr{T}$ proceeds at one step per iteration and $\mathscr{H}$ proceeds at two steps: $\mathscr{T}$ and $\mathscr{H}$ will meet at c^{-1}'s intersection. Then, c^{-1}'s tail can be found by iterating through c^{-1}'s loop till a position that points to the intersection is reached. After finding the tail, the limited depth of the intersection (which will always be the same as the limited depth of c^{-1}'s leader) is computed.

The $\mathscr{L}$ scan aims to compute the limited depth of position i. To do so, $\mathscr{L}$ should identify the tail of c or c^{-1}. $\mathscr{L}$ identifies the tail correctly if it encounters a position storing a limited depth (then that position is the tail), or if the cycle is broken and the tail is computed by the $\mathscr{T}$ and $\mathscr{H}$ scans (as is the case when the cycle is broken and i is on its spine). In the other cases, the $\mathscr{L}$ scan assumes that the tail is the position pointing to i. It returns a correct value if i is a local min leader, and it may not return a correct value otherwise. However, returning an incorrect value in the other cases does not affect the correctness of the algorithm.

The $\mathscr{F}$ scan tests whether i is the local min leader of c or c^{-1}. If $\mathscr{F}$ encounters a limited depth or if the scans $\mathscr{T}$ and $\mathscr{H}$ detect that c^{-1} is broken, $\mathscr{F}$ will behave as if the tail of c^{-1} points to i. The $\mathscr{F}$ scan terminates on one of the following cases:

- The first case is $\mathscr{F}$ determines that i is not a local min leader. If so, the entire process of all four scans is aborted.
- The second case is $\mathscr{F}$ determines the position is a local min leader. Then, two cases can occur:
  - If c^{-1} was broken or a limited depth was encountered, then we know that the cycle is already inverted. Compare the limited depth of i that is computed by $\mathscr{L}$ to the limited depth stored or computed by $\mathscr{T}$ and $\mathscr{H}$. If the two values are equal make the tail point to i. Alternatively, abort all four scans.
  - Otherwise, the cycle c is not inverted. Invert c and if it was bad store in its tail the limited depth of c^{-1}'s local min leader.

Analysis: All four scans use $O(\lg^2 n)$ extra bits. The time complexity is bounded by the time complexity of $\mathscr{F}$, since the runtime of $\mathscr{L}$, $\mathscr{T}$ and $\mathscr{H}$ is at most a constant factor times the runtime of $\mathscr{F}$. For each cycle c, the time spent by F testing for leadership before inverting the cycle is $O(l \lg l)$ where l is the length of c. Inverting c and properly setting its tail if it was bad will take $O(l)$ time. After inverting c, if c^{-1} is bad at most one intermediate broken cycle can be formed, since the limited depth of the local min leader is unique or shared by at most one other position. This fact is crucial to our analysis, and it is the reason why the $\mathscr{L}$ scan is introduced. The time spent testing for leadership for indices in c^{-1} is divided into the following cases:

- c^{-1} is broken and the position i being tested is in c^{-1}'s loop.
- Otherwise either c^{-1} is broken and i is in the spine, or c^{-1} is not broken and the tail stores the limited depth of the leader.
  - If $\mathscr{T}$ does not inspect the tail, then the runtime will be the same as testing whether i is the local min leader of c^{-1}.
  - Otherwise, the procedure will test if i is the local min leader of the cycle formed by pointing the tail of c^{-1} to i. It will iterate at most 4 times from i to the tail [7]. So, the time complexity will be at most 4 times the time complexity of testing weather i is the local min leader of c^{-1}.

In all cases the runtime is bounded by $O(l \lg l)$. Thus, the total runtime per cycle is $O(l \lg l)$ and the total runtime for the whole algorithm is $O(n \lg n)$.

▶ **Theorem 6.** *In the worst case, the standard representation of a permutation of length n can be replaced with its own inverse in $O(n \lg n)$ time using $O(\lg^2 n)$ extra bits of space.*

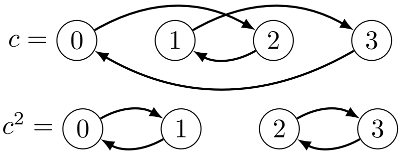


Figure 7 Raising c to the second power results in two separate cycles.

4 Arbitrary Powers

The k^{th} power of a permutation π is π^k defined as follows:

$$\pi^k(i) = \begin{cases} \pi^{k+1}(\pi^{-1}(i)) & k < 0 \\ i & k = 0 \\ \pi^{k-1}(\pi(i)) & k > 0 \end{cases}$$

where k is an arbitrary integer. In this section we extend the techniques presented in the previous section to cover the situation in which the permutation is to be replaced by its k^{th} power for an arbitrary integer k. We present an algorithm whose worst case running time is $O(n \lg n)$ and uses $O(\lg^2 n + \min\{k \lg n, n^{3/4+\epsilon}\})$ additional bits.

Without loss of generality, we assume that k is positive. If k is negative, we invert the permutation then raise it to the power of $-k$. Raising a cycle to an arbitrary power can result in several disjoint cycles as illustrated in Figure 7.

▶ **Lemma 7.** *Raising a cycle of length l to its k^{th} power, will produce $\gcd(k, l)$ cycles each of length $l/\gcd(k, l)$.*

Proof. Suppose μ cycles are produced. Since they are all symmetric, they will have the same length λ. λ is the smallest positive integer such that $(\pi^k)^\lambda(i) = \pi^{k\lambda}(i) = i$, so $k\lambda = cl$ for an integer c that is relatively prime with λ. Now

$$l = \lambda\mu$$
$$k = cl/\lambda = c\mu,$$

but c is relatively prime with λ, so $\mu = \gcd(k, l)$ and $\lambda = l/\gcd(k, l)$. ◀

Given a cycle, it is not hard to raise the cycle to its k^{th} power while using $O(k)$ words or $O(k \lg n)$ bits. Starting from position i, store $i, \pi(i), \pi^2(i), \dots, \pi^{k-1}(i)$ in an array B using $O(k \lg n)$ bits. Replace $A[i]$ with $A[\pi^{k-1}(i)]$, then replace $A[\pi(i)]$ with $A[\pi^k(i)]$, and so on until $A[\pi(i)^{l-k}]$ is reached where l is the length of the cycle. Replace $A[\pi(i)^{l-k}]$ till $A[\pi(i)^{l-1}]$ with the values stored in B. When the procedure terminates, $A[i], A[i+1], \dots, A[i+\gcd(k, l)-1]$ will contain a position from each resulting cycle. An algorithm to raise a permutation to its k^{th} power, will be the same as the algorithm presented in Subsection 3.2, however, the $\mathscr{T}$ scan will raise cycles to their k^{th} power instead of inverting them once they are detected. Then, it will iterate through every cycle of the resulting $\gcd(k, l)$ cycles and compute its leader to check if it was bad. If so, it computes the limited depth of the leader and store it in the cycle's tail.

▶ **Theorem 8.** *In the worst case, the standard representation of a permutation of length n can be replaced with its k^{th} power, when k is bounded by some polynomial function of n, in $O(n \lg n)$ time using $O(\lg^2 n + k \lg n)$ extra bits of space.*

Theorem 8 is useful if the value of k is small. In the next subsection, we show how to power permutations using $o(n)$ extra bits of space.

4.1 Powering Permutations in $O(n \lg n)$ Time Using $o(n)$ Extra Bits

To improve the space complexity, we only have to modify the way we are raising cycles to their k^{th} power. To raise a cycle to its k^{th} power we first find its length l, then we factorize $k \bmod l$. Since $k \bmod l < l$, it can be factored trivially in $o(l)$ time while using little extra storage.

Next, raise the cycle to the power of every prime factor p separately. Here we have to distinguish between two cases:

First Case: p and l are relatively prime. We will use the following theorem given by Rubinstein [13]:

▶ **Theorem 9** (Rubinstein [13], Theorem 4.3). *Let $\gcd(N, a) = 1$ and R be a rectangle. Then, $c_R(N, a)$, the number of solutions (x, y) to $xy = N \bmod a$ with (x, y) lying in the rectangle R is equal to*

$$\frac{\text{area}(R)}{a^2}\phi(a) + O(a^{1/2+\epsilon})$$

for any $\epsilon > 0$, where ϕ is Euler's totient function.

In particular, there exist a point (x, y) where $xy = N \bmod a$ in any square R with side length at least $a^{3/4+\epsilon}$ (R must be larger than $a^{3/2+\epsilon}$).

In this case $\gcd(p, l) = 1$, so there always exist two integers $x, y < l^{3/4+\epsilon}$ such that $xy = p \bmod l$. To find x and y, do a linear search that takes $O(l^{3/4+\epsilon})$ time. Then, raise the cycle to the x^{th} power followed by the y^{th} power using the method described in the previous subsection. The total runtime is $O(l)$ and the space used is $O(l^{3/4+\epsilon})$.

Second Case: p divides l. In this case $\gcd(p, l) = p$ (since p divides l). We will reduce this case to the previous one. Modify the permutation π to form the permutation π' that results from adding an additional position e to the cycle c in π to form the cycle c'. More formally, π' is defined as follows:

- Let a be a position in the cycle c; for all positions $i \in \pi$ except $\pi^{-1}(a)$, $\pi'(i) = \pi(i)$.
- $\pi'(\pi^{-1}(a)) = e$ (where e is a new position).
- $\pi'(e) = a$.

This modification can be done by storing a and two extra words, where the first word stores the inverse of a, and the second stores the image of e ($\pi'(e)$). Each time the array A is accessed at an index i, if $A[i]$ is equal to a, i is checked against the first word stored. If they match, then $A[i]$ points to a otherwise $A[i]$ points to e. Doing this eliminates the need for increasing the word size.

Let $\{c_{ij} | 0 \le i < l/p, 0 \le j < p\}$ be the positions of c, such that

- $\pi(c_{ij}) = c_{i(j+1)}$ if $j < p - 1$
- $\pi(c_{ij}) = c_{(i+1 \bmod l/p)0}$ if $j = p - 1$

Raising c to the power of p will result in p cycles such that the j^{th} cycle c_j will contain the positions $\{c_{ij} | 0 \le i < l/k\}$, where $\pi^p(c_{ij}) = c_{(i+1 \bmod l/p)j}$. The length of c' is $l + 1$ and $\gcd(l + 1, p) = 1$ (since p divides l), so raising c' to the p^{th} power will result in only one cycle.

Without loss of generality assume that $a = c_{00}$. The positions c_{ij} satisfying $a = \pi^m(c_{ij})$ for some $m \in [1, p - 1]$ are precisely $c_{((l/p)-1)j}$ where $j \in [0, p - 1]$. Notice that

- $\pi'^p(c_{ij}) = \pi^p(c_{ij})$ for all c_{ij} such that $a \neq \pi^m(c_{ij})$ for all $m \in [1, p - 1]$
- $\pi'^p(c_{((l/p)-1)0}) = e$
- $\pi'^p(c_{((l/p)-1)j}) = c_{0(j-1)}$ for $1 \le j < p$
- $\pi'^p(e) = c_{0(p-1)}$

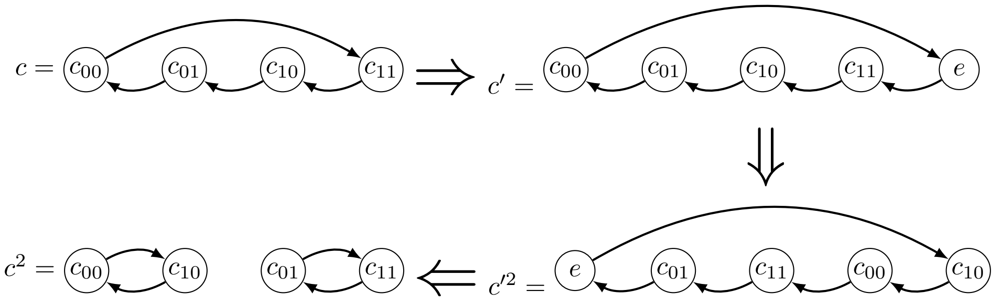


Figure 8 An illustration of case two.

Thus, if we traverse forward in c'^p starting from e, the first p positions are the positions in c_{p-1} ordered correctly, and the second p positions are the positions in c_{p-2}, and so on... After modifying π to π' raise c' to its p^{th} power. Iterate p positions starting from e, then set $A[c_{((l/p)-1)(p-1)}]$ to $c_{0(p-1)}$. Recursively raise c_{p-1} to the power of the rest of the prime factors. Repeat the same process for the rest of the cycles $c_{p-2}, \ldots, c_0$. Each time one of the resultant $\gcd(l,k)$ cycles is reached, find its local min leader and store the limited depth of the leader in the tail if it is a bad cycle. This process is illustrated in Figure 8.

▶ **Theorem 10.** *In the worst case, the standard representation of a permutation of length n can be replaced with its k^{th} power, when k is bounded by some polynomial function of n, in $O(n \lg n)$ time using $O(\lg^2 n + \min\{k \lg n, n^{3/4+\epsilon}\})$ extra bits of space.*

The space complexity in Theorem 10 can be improved if better factoring is applied. More precisely, if for any N and a where $\gcd(N,a) = 1$, we can find $g(a)$ factors $x_1, \ldots, x_{g(a)} \leq f(a)$ such that $x_1 x_2 \ldots x_{g(a)} = N \bmod a$ in $h(a)$ time, then we can achieve an algorithm with running time $O((n + h(n)) \lg n + g(n)n)$ that uses $O(\lg^2 n + \min\{k \lg n, f(n) \lg n\})$ extra bits of space.

Note that given any factoring algorithm as described above, any quadratic non-residue (mod p) can be factored to factors smaller than $f(p)$. Since at least one of the factors must also be a quadratic non-residue, this implies that the least quadratic non-residue (mod p) is smaller than $f(p)$. Thus, reducing $f(n)$ to $O(n^\epsilon)$ is probably difficult since this improvement would imply Vinogradov's conjecture [15] (that the least quadratic non-residue (mod p) lies below p^ϵ).

5 Conclusion

In this paper we presented an algorithm for inverting a permutation that runs in $O(n \lg n)$ worst case time and uses $O(\lg^2 n)$ additional bits. This algorithm is then extended to an algorithm for raising a permutation to its k^{th} power that runs in $O(n \lg n)$ time and uses $O(\lg^2 n + \min\{k \lg n, n^{3/4+\epsilon}\})$ extra bits of space. Both algorithms presented rely on the cycle's local min leader presented in [7]. Moreover, they can easily be adapted to utilize any different cycle leader. A better leader will yield a better algorithm without adding to the worst case time or space complexity for both problems as well as the problem of permuting in place [7].

References

1 Diego Arroyuelo, Gonzalo Navarro, and Kunihiko Sadakane. Reducing the space requirement of LZ-index. In Moshe Lewenstein and Gabriel Valiente, editors, *Proceedings of CPM*, volume 4009 of *Lecture Notes in Computer Science*, pages 318–329. Springer, 2006.

2 Jérémy Barbay, Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theor. Comput. Sci.*, 387(3):284–297, 2007.

3 Jérémy Barbay, Meng He, J. Ian Munro, and Srinivasa Rao Satti. Succinct indexes for strings, binary relations and multilabeled trees. *ACM Transactions on Algorithms*, 7(4):52, 2011.

4 Mariano Paulo Consens and Timothy Snider. Maintaining very large indexes supporting efficient relational querying, August 14 2001. US Patent 6,275,822.

5 Hicham El-Zein, J. Ian Munro, and Venkatesh Raman. Tradeoff between label space and auxiliary space for representation of equivalence classes. In *Proceedings of ISAAC, PartII*, volume 8889 of *Lecture Notes in Computer Science*, pages 543–552. Springer, 2014.

6 Hicham El-Zein, J. Ian Munro, and Siwei Yang. On the succinct representation of unlabeled permutations. In *Proceedings of ISAAC*, volume 9472 of *Lecture Notes in Computer Science*, pages 49–59. Springer, 2015.

7 Faith E. Fich, J. Ian Munro, and Patricio V. Poblete. Permuting in place. *SIAM Journal on Computing*, 24(2):266–278, 1995.

8 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of Seventeenth SODA*, pages 368–373. ACM Press, 2006.

9 Daniel S. Hirschberg and James Bartlett Sinclair. Decentralized extrema-finding in circular configurations of processors. *Communications of the ACM*, 23(11):627–628, 1980.

10 Moshe Lewenstein, J. Ian Munro, and Venkatesh Raman. Succinct data structures for representing equivalence classes. In *Proceedings of ISAAC, PartII*, volume 8283 of *Lecture Notes in Computer Science*, pages 502–512. Springer, 2013.

11 J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations and functions. *Theoretical Computer Science*, 438:74–88, 2012.

12 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), 2007.

13 Michael Rubinstein. The distribution of solutions to $xy = n \bmod a$ with an application to factoring integers, 2009.

14 Kunihiko Sadakane. New text indexing functionalities of the compressed suffix arrays. *J. Algorithms*, 48(2):294–313, 2003.

15 I. Vinogradov. *Selected works. With a biography by K. K. Mardzhanishvili. Translated from the Russian by Naidu Psv. Translation edited by Yu. A.* Springer-Verlag, Berlin, 1985.

Space-Efficient Plane-Sweep Algorithms

Amr Elmasry[1] **and Frank Kammer**[2]

1 **Department of Computer Engineering and Systems, Alexandria University, Alexandria, Egypt**
elmasry@alexu.edu.eg

2 **Institut für Informatik, Universität Augsburg, Augsburg, Germany**
kammer@informatik.uni-augsburg.de

Abstract

We introduce space-efficient plane-sweep algorithms for basic planar geometric problems. It is assumed that the input is in a read-only array of n items and that the available workspace is $\Theta(s)$ bits, where $\lg n \le s \le n \cdot \lg n$. Three techniques that can be used as general tools in different space-efficient algorithms are introduced and employed within our algorithms. In particular, we give an almost-optimal algorithm for finding the closest pair among a set of n points that runs in $O(n^2/s + n \cdot \lg s)$ time. We also give a simple algorithm to enumerate the intersections of n line segments that runs in $O((n^2/s^{2/3}) \cdot \lg s + k)$ time, where k is the number of intersections. The counting version can be solved in $O((n^2/s^{2/3}) \cdot \lg s)$ time. When the segments are axis-parallel, we give an $O((n^2/s) \cdot \lg^{4/3} s + n^{4/3} \cdot \lg^{1/3} n)$-time algorithm that counts the intersections and an $O((n^2/s) \cdot \lg s \cdot \lg\lg s + n \cdot \lg s + k)$-time algorithm that enumerates the intersections, where k is the number of intersections. We finally present an algorithm that runs in $O((n^2/s + n \cdot \lg s) \cdot \sqrt{(n/s) \cdot \lg n})$ time to calculate Klee's measure of axis-parallel rectangles.

1998 ACM Subject Classification E.1 Data Structures F.2.2 Nonnumerical Algorithms and Problems, I.1.2 Analysis of Algorithms, I.3.5 Geometric Algorithms

Keywords and phrases closest pair, line-segments intersection, Klee's measure

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.30

1 Introduction

Because of the rapid growth of the input data sizes in current applications, algorithms that are designed to efficiently utilize space are becoming even more important than before. One other reason to run a space-efficient algorithm is the limitation in the memory sizes that can be deployed to modern embedded systems. Therefore, many algorithms have been developed with the objective to optimize the time-space product.

Several models of computation have been considered for the case when writing in the input area is restricted. The objective of a space-efficient algorithm is to optimize the amount of extra space needed to perform its task. In the *multi-pass streaming model* [19] the input is assumed to be held in a read-only sequentially-accessible working space, and the goal would be to optimize the number of passes an algorithm makes over the input. In the *read-only word RAM* [16]—the model that we consider in this paper—the input is assumed to be stored on a read-only randomly-accessible working space and arithmetic operations on operands that fit in one word are assumed to take constant time each.

Throughout the paper, it is assumed that n is the number of items of the input, each stored in a constant number of words, and that the available workspace is $\Theta(s)$ bits, where $\lg n \le s \le n \lg n$. Since a single cursor, which is necessary to iterate over the input, already needs $\Theta(\lg n)$ bits, there is no hope to solve any of the problems with less workspace. In

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 30; pp. 30:1–30:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

addition, and as usual on a word RAM, it is assumed that operations on the input coordinates can be performed in constant time each. We emphasize that this assumption is not essential for our algorithms to work, but only scales with their running times.

Next, we survey some of the major results known for the read-only random-access model. Pagter and Rauhe [21] gave an asymptotically-optimal algorithm for sorting n elements that runs in $O(n^2/s + n \lg s)$ time. Beame [5] established a matching $\Omega(n^2)$ lower bound for the time-space product for sorting in the stronger branching-program model. Elmasry et al. [15] introduced space-efficient algorithms for basic graph problems. Concerning geometric problems, Chan [9] presented an algorithm for the closest-pair problem with integer coordinates in the word RAM model, and his algorithm can be made to work in the read-only model. Darwish and Elmasry [14] gave an optimal planar convex-hull construction algorithm that runs in $O(n^2/s + n \lg s)$ time. Konagaya and Asano [17] gave an algorithm for reporting line-segments intersections that runs in $O((n^2/\sqrt{s}) \cdot \sqrt{\lg n} + k)$ time, where k is the number of intersections. Recently, Korman et al. [18] gave space-efficient algorithms to construct triangulations and Voronoi diagrams whenever $s = \Omega(\lg n \cdot \lg \lg n)$ bits of working space are available. Asano et al. [3] considered space-efficient plane-sweep algorithms for Delaunay triangulation and Voronoi diagram. However, they only considered the case where $s = \Theta(\log n)$ bits, and both algorithms run in $O(n^2)$ time for this case.

As a building block for our algorithms we use the *adjustable navigation pile* [2]; an efficient priority-queue-like data structure that uses $O(s)$ bits, where $\lg n \leq s \leq n \lg n$, in the read-only random-access model of computation. Given a read-only input array of n elements and a specified value, an adjustable navigation pile can be initialized in $O(n)$ time. Subsequently, the elements that are larger than the given value can be streamed in sorted order in $O(n/s + \lg s)$ time per element. Thus, it is possible to stream the next k elements starting with a specified value in sorted order in $O((n/s + \lg s) \cdot k + n)$ time, and all the elements of the array can be streamed in sorted order in $O(n^2/s + n \lg s)$ time.

Another ingredient that we use in some of our algorithms is a *rank-select* data structure [12]. A rank-select data structure can be built on a bit vector of length n using $O(n)$ time and $o(n)$ extra bits, and supports in $O(1)$ time the queries *rank*(i), which returns the number of 1-bits in the first i positions of the bit vector, and *select*(j), which returns the index of the j-th 1-bit in the bit vector. In accordance, one can sequentially scan the entries of the bit vector that have 1-bits in $O(1)$ time per entry.

In this paper we give space-efficient plane-sweep algorithms that solve planar geometric problems where one moves a line across the plane and maintains the intersection of that line with the objects of interest. Many geometric problems have been solved using this kind of algorithm [7, 8, 13]. We assume that the sweep line moves over the plane from left to right. Typically, a plane-sweep algorithm uses a priority queue (*event queue*) to produce the upcoming events in order and a balanced binary search tree (*status structure*) to store and query the objects that cross the sweep line in order. The status structure is updated only at particular *event points*. Since $\Theta(n)$ objects might be part of the search tree, a typical plane-sweep algorithm needs $\Theta(n \lg n)$ bits, which is true for the standard algorithms of all problems considered here. In contrast to Asano et al. [3], all our algorithms allow a trade-off between time and space and work for all values of s where $\lg n \leq s \leq n \lg n$.

In Section 2 we introduce a general technique that we call the *divide-and-compress technique* relying on splitting the input array, and later employ it in our algorithms. In Section 3 we give a simple algorithm that enumerates intersections among n line segments and runs in $O((n^2/s^{2/3}) \cdot \lg s + k)$ time, where k is the number of intersections. Our algorithm is asymptotically faster than that of Konagaya and Asano for all values of s. We point out that the same approach can be used to count the number of intersections in $O((n^2/s^{2/3}) \cdot \lg s)$ time. In Section 4 we give an algorithm for finding the closest pair among n points whose running time is $O(n^2/s + n \lg s)$. To obtain this result, we combine new ideas with the classical plane-sweep and divide-and-conquer approaches for solving the closest-pair problem. A lower bound of $\Omega(n^{2-\epsilon})$ was shown by Yao [22] for the time-space product of the element-distinctness problem, where ϵ is an arbitrarily small positive constant. This lower bound applies for the closest-pair problem, indicating that our algorithm is close to optimal. In Section 5 we give an algorithm for counting the intersections among n axis-parallel line segments that runs in $O((n^2/s) \cdot \lg^{4/3} s + n^{4/3} \cdot \lg^{1/3} n)$ time. The idea is to partition the plane as a grid and to run local plane sweeps on parts of the plane with truncated segments. In Section 6 we sketch a so-called *batching technique* to represent the sweep line for special plane-sweep algorithms using fewer bits than usual, and then utilize this technique in Section 7 for enumerating the intersections among n axis-parallel line segments in $O((n^2/s) \cdot \lg s \cdot \lg \lg s + n \lg s + k)$ time, where k is the number of intersections. In Section 8 we show how to calculate Klee's measure (the area of the union) for n axis-parallel rectangles in $O((n^2/s) \cdot \lg n + n \lg s)$ time if the corners of the rectangles are stored in sorted order. In Section 9 we introduce another general technique that we call the *multi-scanning technique* where we partition the plane and run several plane sweeps interleaved. We use this technique to calculate Klee's measure in $O((n^2/s + n \lg s) \cdot \sqrt{(n/s) \cdot \lg n})$ time if the corners of the rectangles are unsorted. We conclude the paper in Section 10 with some comments.

2 A Divide-and-Compress Technique: Splitting the Input Array

We call a problem *decomposable* if any partitioning of the input into subsets allows us to compute a solution for the input by computing the partial solutions for these subsets as well as for the unions of all pairs of subsets and by combining these partial solutions. We also assume that the time needed to combine the results is bounded by the time to compute the partial solutions. Examples of such problems that we deal with in this paper are the axis-parallel line-segments intersections problem (Section 3) and closest-pair problem (Section 4). For the closest-pair distance, the overall solution is the minimum among the partial solutions for the subproblems. For the enumeration of the axis-parallel line-segments intersections, the overall solution is the union of the non-overlapping partial solutions. The general line-segments intersections problem is also decomposable, and can be handled using the same approach with slight modifications.

The following technique divides the instance in smaller parts and so compresses the necessary workspace used to solve a decomposable problem. Assume that the available workspace is enough to only handle a subset of the input that comprises $O(r)$ elements at a time, for some parameter r smaller than the number n of elements in the input array. Split the array into $\lceil n/r \rceil$ *batches* $B_1, \ldots, B_{\lceil n/r \rceil}$ of at most r consecutive elements each (the last batch may have less) and proceed as follows: For $i = 1, \ldots, \lceil n/r \rceil$ and $j = i + 1, \ldots, \lceil n/r \rceil$, apply the underlying algorithm within $B_i \cup B_j$. Compute the overall answer by combining the partial results. As we try all pairs of subproblems, the algorithm correctly explores all the possible subproblems $B_i \cup B_j$ for some i and j, and accordingly produces the output correctly for decomposable problems.

The number of the subproblems handled in sequence is $\Theta(n^2/r^2)$. Let the time needed to solve a subproblem of size $O(r)$ be $t(r) + k'$ where k' is the size of the output. Thus, the overall time spend by the algorithm is $O((n^2/r^2) \cdot t(r) + k)$ where k is the size of whole output.

▶ **Lemma 1.** *Suppose we know how to solve a decomposable problem $\mathcal{P}$ of size n using $s' = \Theta(f(n))$ bits in $O(n^2/g(s') + n \lg s')$ time, where $f, g : I\!N \to I\!R$ are functions with $\lg n \leq f(n) \leq n \lg n$. For any s where $\lg n \leq s \leq s'$, we can solve any instance I of $\mathcal{P}$ of size n in $O(n^2/g(s) + (n^2/f^{-1}(s)) \cdot \lg s)$ time with $O(s)$ bits. In particular, when $f(n) = O(n/\lg n)$ and $g(s) = O(s)$, we can solve I in $O(n^2/g(s))$ time and $O(s)$ bits.*

Proof. By definition of $\mathcal{P}$, we can solve instances of $\mathcal{P}$ that are of size $r = \lceil f^{-1}(s) \rceil$ using s bits in $t(r) = O(r^2/g(s) + r \lg s)$ time. By applying the above construction, we can solve I in $O((n^2/r^2) \cdot t(r)) = O(n^2/g(s) + (n^2/r) \cdot \lg s) = O(n^2/g(s) + (n^2/f^{-1}(s)) \cdot \lg s)$ time. If $f(n) = O(n/\lg n)$, then $f^{-1}(s) = \Omega(s \lg s)$, and we can solve I in $O(n^2/g(s) + n^2/s)$. If in addition $g(s) = O(s)$, the claimed time and space bounds follow. ◀

3 Line-Segments Intersections

Given a set of n line segments in the plane, the line-segments-intersections problem is to enumerate all the intersection points among these line segments. The counting version of the problem is to produce the number of these intersections. An optimal algorithm to enumerate all the intersections that runs in $O(n \lg n + k)$ time was given by Balaban [4], where n is the number of segments and k is the number of intersections returned. Chazelle [11], improving a result of Agarwal [1], showed how to count the intersections among n line segments in $O(n^{4/3} \lg^{1/3} n)$ time, and how to report k *bichromatic intersections* in $O(n^{4/3} \lg^{1/3} n + k)$ time, i.e., given sets of red and blue segments, to report all intersections between a red and a blue segment. All these algorithms require a linear number of words, i.e., $O(n \lg n)$ bits.

If the available workspace is $\Theta(s)$ bits with $\lg n \leq s \leq n \lg n$, we give next a straightforward application of the divide-and-compress technique. We can apply the reporting algorithms on batches of size $O(r)$ line segments, where we choose $r = \Theta(f^{-1}(s))$, i.e., $r = \Theta(s/\lg s)$. First we apply Balaban's algorithm for each batch separately, then we apply a bichromatic-intersections algorithm on every pair of batches (coloring one of them red and the other blue). Note that we cannot apply Balaban's algorithm on pairs of batches since the partial solutions will be overlapping (intersections among the segments of a batch will show up in several partial solutions), and hence combining the partial solutions would be problematic. Thus, the running time $t(r)$ on r segments is $O(r^{4/3} \cdot \lg^{1/3} r)$. The reported intersections are the union of the non-overlapping intersections found by solving the subproblems. Hence, the overall time for this algorithm is $O((n^2/r^2) \cdot t(r) + k) = O((n^2/r^{2/3}) \cdot \lg^{1/3} r + k) = O((n^2/s^{2/3}) \cdot \lg s + k)$ time, where k is the number of reported intersections.

▶ **Theorem 2.** *Given a read-only array of n elements and $\Theta(s)$ bits of workspace, where $\lg n \leq s \leq n \lg n$, the planar line-segments-intersections enumeration problem can be solved in $O((n^2/s^{2/3}) \cdot \lg s + k)$ time, where k is the number of intersections returned. The counting version can be solved in $O((n^2/s^{2/3}) \cdot \lg s)$ time.*

4 Closest Pair

Given a set of n points in the plane, the planar closest-pair problem is to identify a pair of points that are closest to each other.

Assume for the moment that the available workspace is $\Theta(s)$ bits, where $\sqrt{n} \cdot \lg n \leq s \leq n \lg n$. In a first step, we produce the points in sorted order according to their x-coordinate values using an adjustable navigation pile and partition them into groups having $\lceil s/\lg n \rceil$ successive points each (except possibly the last group). Partition the plane in vertical regions, called *vertical stripes*, where each region contains one group—if necessary, rotate the plane slightly. More exactly, choose the vertical regions such that the vertical lines separating the vertical stripes, called the *vertical separators*, pass through a point. We deal with the vertical separators in our workspace by storing, for each of them, $O(\lg n)$ bits of the index of the corresponding point in the input array. Since there are at most $m = \lceil (n/s) \cdot \lg n \rceil$ vertical stripes, references to the x-coordinate values of all the vertical separators can be stored in $O((n/s) \cdot \lg^2 n)$ bits, which is $O(s)$ as long as $s = \Omega(\sqrt{n} \cdot \lg n)$. The entities of all the separators can then be simultaneously stored within the available workspace. Additionally, all the points of a vertical stripe can fit in the available workspace. Thus, a standard closest-pair algorithm [13] can be applied to identify the closest pair among the points of each vertical stripe one after the other. We then find the pair with the minimum closest distance among all the vertical subproblems, and call this minimum distance δ.

To find a closest pair that is spread over two different stripes, we use a standard idea from the divide-and-conquer algorithm for the closest-pair problem. In a second step, we produce the points in sorted y-coordinate order using another adjustable navigation pile, but retain only the points that lie within a horizontal distance δ from any of the vertical separators. Call these points the *candidate points*. We consider the candidate points in the y-coordinate order in groups having $8m$ points each (except the last group that may have less points). Call the horizontal regions containing these groups the *horizontal stripes*. Note that references to the points of a horizontal stripe can be stored in $O((n/s) \lg^2 n) = O(s)$ bits, which can all fit in the available workspace. We can then apply a standard closest-pair algorithm within the working storage to identify the closest pair among the candidate points of every two consecutive horizontal stripes in order. Let δ' be the minimum closest distance among all the horizontal subproblems. Finally, we return $\min(\delta, \delta')$ as the closest-pair distance.

We prove next the correctness of the algorithm. We only have to show that the restriction to the points close to the vertical separators in the second step is correct. We slightly generalize the proof for the standard divide-and-conquer algorithm for the closest-pair problem. Since the distance between any pair of points within a vertical stripe is at least δ, any point that is at horizontal distance more than δ from all the vertical separators can not be closer than δ to any other point. We then only need to proceed with the candidate points that lie within a horizontal distance δ from any of the vertical separators. Fix a candidate point p. Given a specific vertical separator, for the candidate points above p to be closer than δ to p they must lie together with p within a $2\delta \times \delta$ rectangle centered at the vertical separator. Note that there could be at most 8 points above p within this rectangle whose distances to p are less than δ, since 2 rows of 3 circles of diameter δ can cover the whole rectangle and there can be at most one point in the two left and the two right circles as well as at most two points in the middle circles. Since there are m vertical separators, the number of candidate points P above p to be checked for possibly having a distance less than δ from p is at most $8m$; no other point above p can be at distance less than δ from p. (Actually, it suffices to check only 5 candidate points above p for each separator [13, Exercise 33.4-2].) Obviously, the points in P must be consecutive in the y-coordinate values. Since we store $8m$ candidate points per stripe, the points in P lie in only two horizontal stripes, the horizontal stripe that spans p and the horizontal stripe above it. We conclude that we need to only consider the mutual distances among the points of each two consecutive horizontal stripes.

We can produce the points in sorted order in both coordinates in $O(n^2/s + n \lg s)$ time using the adjustable navigation pile [2]. The time needed to execute the standard closest-pair algorithm for all the stripes is $O(n \lg s)$ [13]. The check whether each point is close to one of the separators or not runs in $O(n \lg n) = O(n \lg s)$ time using a binary search among

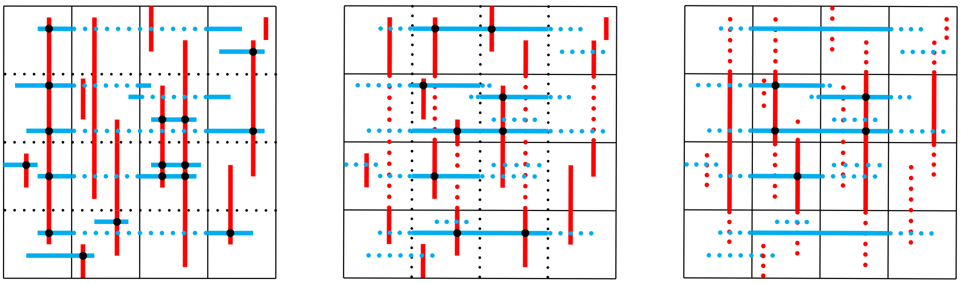


Figure 1 Counting axis-parallel line segments in three *phases*. Each stripe contains 12 points and, for clarity reasons, all stripes have the same size. The black dots on the crossings of two segments show the intersection points that are counted in each phase.

the x-coordinates of the separators for each point. Hence, the overall running time of the algorithm is $O(n^2/s + n \lg s)$.

Assume now that we have $\Theta(s)$ bits available, where $\lg n \le s < \sqrt{n} \cdot \lg n$. Let $r = s^2/\lg^2 s$. As $s = \Theta(\sqrt{r} \cdot \lg r)$, we can apply the above algorithm on instances of size $\Theta(r)$. In such a case, the running time for each of these instances would be $t(r) = O(r^2/s + r \lg s) = O(r^2/s)$. We then divide the input into $\lceil n/r \rceil$ batches of points and apply the divide-and-compress technique, compute the closest pair within every pair of batches and return the overall closest pair. The space needed is indeed $O(s)$, and the time consumed is $O((n/r)^2 t(r)) = O(n^2/s)$.

▶ **Theorem 3.** *Given a read-only array of n elements and $\Theta(s)$ bits of workspace, where $\lg n \le s \le n \lg n$, the planar closest-pair problem can be solved in $O(n^2/s + n \lg s)$ time.*

It is known that the closest-pair algorithm can be generalized from two to higher dimensions [13] to run in $O(n \lg^{d-1} n)$ time in d dimensions. Applying the divide-and-compress technique in a similar way as described above, we can solve the closest-pair problem in d dimensions with $\Theta(s)$ bits of workspace in $O(n^2/s + n \lg^{d-1} s)$ time, where $\lg n \le s \le n \lg n$.

5 Counting Axis-Parallel Line-Segments Intersections

Given a set of n axis-parallel (horizontal or vertical) line segments in the plane, we want to count the intersection points among these line segments.

Assume for the moment that the available workspace is $\Theta(s)$ bits, where $n^{2/3} \cdot \lg n \le s \le n \lg n$. First, we produce the endpoints of the line segments in sorted order according to their x-coordinate values using an adjustable navigation pile, and consider them in order in groups having $\lceil s/\lg n \rceil$ points each (except possibly the last group that may have fewer points). As in the previous section, these groups define the *vertical stripes* and *vertical separators*. Since there are $\lceil (n/s) \cdot \lg n \rceil = O(n^{1/3})$ vertical stripes, references to the x-coordinate values of all the separators can be stored within the workspace. We associate a line segment to a stripe if at least one of its two endpoints lies inside the stripe. If we consider the line segments of a vertical stripe, or more exactly, their positions in the input array, they can all fit in the workspace. Thus, we can apply a standard line-segments-intersections counting algorithm to each vertical stripe one after the other, and add these counts together. See the left side of Fig. 1.

Subsequently, we produce the points in sorted order according to their y-coordinate values using another adjustable navigation pile, and partition the plane in *horizontal stripes* (analogous to the definition of the vertical stripes) such that each has $\lceil s/\lg n \rceil$ points (except possibly the last group that may have less points). It follows that the $O(n^{1/3})$ references to the so-called *horizontal separators* can be simultaneously stored in the workspace. In a similar fashion as above, we apply a line-segments-intersections counting algorithm to each horizontal stripe one after the other, and add these counts to the accumulated count. To avoid counting intersections twice, we modify the access to the horizontal segments such that further computations consider the segments to be truncated. Each new endpoint lies on the closest vertical separator to the old endpoint intersecting the segment. Note that the intersections of the truncated parts of the horizontal segments with vertical segments have already been accounted for while dealing with the vertical stripes. See the middle of Fig. 1.

Let $\mathcal{R}_{i,j}$ be the *cell* formed by the intersection of the ith horizontal stripe with the jth vertical stripe. A line segment *spans* a cell if it crosses two of the cell's boundaries. It remains to account for the intersections among these spanning segments. The number of these intersections for each cell is the product of the numbers of its spanning horizontal and vertical segments. We show next how to count the spanning horizontal segments for each cell. The treatment for the vertical segments is similar. See the right side of Fig. 1.

A line segment is *interior* to a cell if both its endpoints lie inside the cell. For each cell $\mathcal{R}_{i,j}$, we store the count $b_{i,j}$ of horizontal segments beginning in the cell, the count $f_{i,j}$ of horizontal segments finishing in the cell, and the count $t_{i,j}$ of the horizontal segments interior to the cell. Since there are $O(n^{2/3})$ cells, all these values can be stored in $O(n^{2/3} \cdot \lg n)$ bits, which is $O(s)$ when $s \geq n^{2/3} \cdot \lg n$. For every horizontal segment, we locate the starting and ending cells using binary search among the separators, and increment the corresponding counters in accordance. We then scan the cells of every horizontal stripe sequentially while calculating $e_{i,j}$ the number of horizontal segments *entering* $\mathcal{R}_{i,j}$, i.e., the number of segments that have a non-empty intersection with $\mathcal{R}_{i,j}$ and $\mathcal{R}_{i,j-1}$; this is done using $e_{i,0} = 0$ and $e_{i,j} = e_{i,j-1} + b_{i,j-1} - f_{i,j-1}$. We finally obtain the number of horizontal segments spanning $\mathcal{R}_{i,j}$ as $e_{i,j} - f_{i,j} + t_{i,j}$. The time needed to produce the endpoints in sorted order in both coordinates using the adjustable navigation pile is $O(n^2/s + n \lg s)$ [2]. The time needed to execute the standard segments-intersection counting algorithm for all the stripes is $O(n^{4/3} \cdot \lg^{1/3} n)$. The time needed to perform binary search among the separators is $O(n \lg s)$. The time needed to count the intersections of the spanning segments of all the cells is constant per cell and sums up to $O(n^{2/3})$. It follows that the overall running time of the algorithm is $O(n^{4/3} \cdot \lg^{1/3} n)$.

Assume now that we have $\Theta(s)$ bits available, where $\lg n \leq s < n^{2/3} \cdot \lg n$. Let $r = s^{3/2}/\lg^{3/2} s$. Since $s = \Theta(r^{2/3} \lg r)$, we can apply the above algorithm on instances of $\Theta(r)$ elements. We divide the input array into $\lceil n/r \rceil$ batches of consecutive segments and apply the divide-and-compress technique. First apply the algorithm on instances for every batch individually, then on instances for every pair of batches. Using these computed counts, the overall count can be easily calculated. The running time for each instance would be $t(r) = O(r^{4/3} \cdot \lg^{1/3} s)$. The overall time consumed in this case is $O((n/r)^2 \cdot t(r)) = O((n^2/s) \cdot \lg^{4/3} s)$.

▶ **Theorem 4.** *Given a read-only array containing the endpoints of n line segments and $\Theta(s)$ bits of workspace, where $\lg n \leq s \leq n \lg n$, counting the planar axis-parallel line-segments intersections can be done in $O((n^2/s) \cdot \lg^{4/3} s + n^{4/3} \cdot \lg^{1/3} n)$ time.*

6 A Batching Technique: Processing Sweep-Line Events in Batches

We now show that, if the given objects are axis-parallel, one may reduce the working storage of the status structure to $\Theta(n)$ bits by processing the events in batches.

For a parameter m to be set later, suppose our plane is divided into m vertical and horizontal stripes such that each stripe contains $O(n/m)$ local objects, where an object is *local* for a stripe if it starts or ends within the stripe. As before, the boundaries of the stripes are called *separators*. The intersection of a horizontal stripe with a vertical stripe is called a *cell*. To apply the batching technique, we need a plane-sweep instance with the following two properties: (1) All the events of the event queue are on vertical separators, i.e., they result from so-called *horizontally spanning objects*. (2) All the objects of the status structure start and end on horizontal separators, i.e., they are so-called *vertically spanning objects*. Assume the available workspace is $\Theta(s)$ bits, where $n \leq s \leq n \lg n$. By setting $m = \lceil (n/s) \cdot \lg n \rceil$, we can store references to all local objects of a stripe and references to the coordinates of the separators in the working storage. Because of properties (1) and (2), it is enough to update the status structure only once per vertical stripe with a batch of objects. To 'represent' the status structure, we split the vertical stripe to m cells formed by the intersections with the horizontal stripes. Recall that there are $O(n/m)$ vertically spanning objects that are in a cells of the vertical stripe. We store their positions in an array using a total of $O((n/m) \cdot \lg n) = O(s)$ bits. In addition, we store for each of the m cells a bit vector of $O(n/m)$ bits indicating whether each of these objects spans the cell or not. Over and above, for each bit vector of a cell, we build a rank-select data structure that allows us to scan the vertical spanning objects of the cell in constant time per object. The bit vectors and the rank-select structures are enough to represent the status structure. Thus, the sweep line can be stored in a total of $O(s)$ bits. We use an adjustable navigation pile as our event queue to produce the events and the spanning objects in order. Since $s \geq n$, the time to produce all the events in order throughout the procedure is $O(n \lg s)$. When the sweep line moves to a new vertical stripe, we update the representation of the status structure as follows: The vertical spanning objects in the new stripe are produced by the navigation pile. For each such object, the cells that it spans are allocated in $O(m)$ time per object by simply comparing the object coordinates with the horizontal separators. The bit-vectors entries and the rank-select structures are updated accordingly. The time to update the status structure (build a new one) is $O(n)$. Throughout the algorithm, the total time to update the status structure is $O(n \cdot m) = O((n^2/s) \cdot \lg n) = O((n^2/s) \cdot \lg s)$.

It remains to show how to allocate an event point within the status structure representing the sweep line. We would be satisfied with only identifying the cell that contains this event point within the vertical stripe. We do that using binary search against the m horizontal separators, consuming $O(\lg m) = O(\lg \lg s)$ time per event point.

▶ **Lemma 5.** *Let $\mathcal{I}$ be a plane-sweep instance for which (1) and (2) holds. Using the batching technique, a sweep can be performed on a plane with n objects, using a data structure that can be stored in $\Theta(s)$ bits, where $n \leq s \leq n \lg n$. The sweep makes a total of $O((n/s) \cdot \lg s)$ stopovers, and the data structure can be rebuilt in $O(n)$ time per stopover plus a total of $O(n \lg s)$ time, and queried in $O(\lg \lg s)$ time per event. Handling all events at each stopover, we can run a plane-sweep algorithm on $\mathcal{I}$ in $O((n^2/s) \cdot \lg s \cdot \lg \lg s + n \cdot \lg s)$ total time.*

7 Enumerating Axis-Parallel Line-Segments Intersections

Assume for the moment that the available workspace is $\Theta(s)$ bits, where $n \leq s \leq n \lg n$.

We use the same ideas as in Section 5. As before, we split the plane into $m = \lceil (n/s) \cdot \lg n \rceil$ horizontal and vertical stripes where each except the last contains $\lceil n/m \rceil$ line segments. We enumerate the intersections among the local parts of the segments within the stripes by applying a standard line-segments-intersection enumeration algorithm.

By truncating the segments, we assume from now on that all the endpoints lie on the boundaries of the cells and the segments span the cells they cross. Note that each horizontal line segment that spans a cell must intersect all the vertical segments spanning the same cell. By applying the ideas of the batching technique, we store the vertical spanning line segments that lie in the current vertical stripe and build a status structure that consumes $\Theta(s)$ bits in $O(n)$ time. Using this data structure it is possible to enumerate the vertical segments that span a given cell in time proportional to the number of the reported segments. For each horizontal segment, we check if it spans any of the cells of the sweep line. We do that using binary search for each horizontal segment against the m horizontal separators. After every binary search for a horizontal segment, we query the status structure to find the vertical segments spanning the same cell, and so their intersections with the horizontal segment are computed and reported. After locating the crossing cells of all the horizontal segments with the sweep line, the sweep line is advanced to the next vertical stripe.

The total time needed to execute the standard algorithm locally within all the stripes is $O(n \lg s)$, which matches the time bound to build the status structure of all vertical stripes using the batching technique. The time to perform binary search for each of the horizontal segments against the m cells of the status structure is $O(n \lg m)$. Hence, we can compute all intersection points of a vertical stripe in $O(n \lg m + k')$ time, where k' is the number of these intersections. Since we repeat these actions for every vertical stripe as the sweep line advances, the total time is $O(n \cdot m \cdot \lg m + k) = O((n^2/s) \cdot \lg s \cdot \lg\lg s + k)$, where k is the number of intersections returned. Since we can partition the plane into stripes using a navigation pile in $O(n^2/s + n \lg s)$ time, the total time consumed by the whole algorithm is $O((n^2/s) \cdot \lg s \cdot \lg\lg s + n \lg s + k)$.

Assume next that we have $\Theta(s)$ bits of workspace, where $\lg n \leq s < n$. Let $r = s$. We can then apply the above algorithm on instances of size $\Theta(r)$. In such a case, the running time for each instance would be $t(r)+k' = O((r^2/s)\cdot\lg s\cdot\lg\lg s+r\lg s+k') = O((r^2/s)\cdot\lg s\cdot\lg\lg s+k')$, where k' is the number of intersections. We then divide the input into $\lceil n/r \rceil$ batches of segments and apply the divide-and-compress technique on pairs of batches, a batch of vertical segments with a batch of horizontal segments. The total time consumed is $O((n/r)^2 \cdot t(r) + k) = O((n^2/s) \cdot \lg s \cdot \lg\lg s + k)$, where k is the number of intersections returned.

▶ **Theorem 6.** *Given a read-only array containing the endpoints of n line segments and $\Theta(s)$ bits of workspace, where $\lg n \leq s \leq n \lg n$, enumerating the planar axis-parallel line-segments intersections is done in $O((n^2/s) \cdot \lg s \cdot \lg\lg s + n \lg s + k)$ time, where k is the number of intersections returned.*

8 Measure of Axis-Parallel Rectangles

We consider the problem of computing the *measure* of a set of n axis-parallel rectangles, i.e., the size of the area of the union. The problem was posed by V. Klee, and thus called *Klee's measure problem*. Bentley [6] described an $O(n \lg n)$-time algorithm that can be implemented with $\Theta(n \lg n)$ bits of working space. Bentley's algorithm sweeps a vertical line from left to right across the rectangles and maintains the intersection of the rectangles and the sweep line. Another algorithm to compute the measure was presented by Overmars and Yap [20]. A generalization of the algorithm to d dimensions was given by Chan [10].

Assume that the available workspace is $\Theta(s)$ bits, where $\lg n \leq s \leq n \lg n$. To compute the measure of a set of n axis-parallel rectangles, we use Bentley's algorithm as a subroutine. In this section, we restrict ourselves to the case where the corners of the rectangles are stored sorted by their x-coordinates. This restriction is dropped in the next section.

We split the plane into $m = \Theta((n/s) \cdot \lg n)$ horizontal stripes, where each stripe consists of $\Theta(s/\lg n)$ rectangle corners and accordingly fit in the available workspace. A rectangle is *spanning* a stripe if its vertical segments cross the two separators of the stripe. We process the stripes in sorted y-coordinate order, one after the other. By using an adjustable navigation pile, we produce and store the rectangles cornered within each stripe in sequence. Before processing a stripe and storing the rectangles, we truncate those rectangles such that they are shrunk to their intersection with the stripe. We would then run Bentley's algorithm on these rectangles. However, we need to also take into consideration the rectangles spanning the stripe. We show next how to do that efficiently.

We horizontally scan the stripe and keep track of the spanning segments and the corners. We accumulate as a global variable the width $\mathcal{W}$ of the union of the spanning rectangles so far. To do that, we maintain z as the difference between the number of scanned spanning segments that are left boundaries of a rectangle and the number of scanned spanning segments that are right boundaries. Whenever z becomes positive, we record this coordinate as x_1. Whenever z returns back to zero, we record this coordinate as x_2; we have just passed over a spanning area, and accordingly update $\mathcal{W}$ by adding to it the value $x_2 - x_1$. Whenever we meet a corner, we update its x-coordinate value as follows. If z is positive (the corner is in a spanning area), first set the x-coordinate of this corner to x_1. Either way, whether z is positive or zero, we subtract the current value of $\mathcal{W}$ from the x-coordinate of the corner. This process of relocating the corners is called *simplifying* the rectangles in [10]. After finishing the scan, we apply Bentley's algorithm to the relocated corners and calculate the measure within the current stripe. We also multiply $\mathcal{W}$ by the width of the stripe to get the area covered by the spanning rectangles, and add this area to the calculated measure. The total measure is the sum of the measures within all the stripes.

The time to sequentially scan the segments and simplify the rectangles within each stripe is $O(n)$ (as the segments are already sorted), and the time for applying Bentley's algorithm is $O((s/\lg n) \cdot \lg s)$. The total time to process all the m stripes is $O((n^2/s) \cdot \lg n + n \lg s)$.

▶ **Theorem 7.** *Given a read-only array storing the corners of n axis-parallel rectangles in sorted x-coordinate order, and the available workspace is $\Theta(s)$ bits, where $\lg n \leq s \leq n \lg n$, the measure (area of the union) can be computed in $O((n^2/s) \cdot \lg n + n \lg s)$ time.*

9 A Multi-Scanning Technique: Partitioning the Plane

In this section we introduce a technique to replace one global sweep with many local sweeps, and apply it to the measure problem if the input is not sorted. The idea is to perform alternating vertical and horizontal sweeps on parts of the plane to identify cells, each containing a set of objects that fit in the working storage. Once identified, we apply a local algorithm within each cell. By partitioning the plane into a grid of cells, we combine the local solutions for the cells together to obtain the final outcome. The details come next.

We partition the plane into $m = \lceil \sqrt{(n/s) \cdot \lg n} \rceil$ horizontal stripes, where each stripe consists of $O(n/m)$ corners. We process the horizontal stripes one after the other in sorted y-coordinate order using an adjustable navigation pile. Once the two separators of a horizontal stripe $\mathcal{H}$ are determined, we initialize an adjustable navigation pile $Y_{\mathcal{H}}$ for the stripe that allows us to stream the corners within $\mathcal{H}$ ordered by their y-coordinates. We start sweeping over the plane in sorted x-coordinate order using another adjustable navigation pile $X_{\mathcal{H}}$ that is initialized over the whole input. For this horizontal sweep, we are interested only in the corners in $\mathcal{H}$ as well as the vertical segments spanning $\mathcal{H}$—to find the spanning segments, we have to take all corners of the plane into consideration. Whenever the number of corners in

$\mathcal{H}$ produced by $X_{\mathcal{H}}$ is $\ell = \lceil s/\lg n \rceil$ (except for the last cell that may have less corners), we have reached a vertical separator that identifies, as a right boundary, a cell $\mathcal{V}$ within $\mathcal{H}$. The corners of a cell can be stored in $O(s)$ bits and hence fit in the working storage. During this horizontal sweep over $\mathcal{V}$, we calculate the horizontal width $\mathcal{W}_h$ of the area covered by the vertically spanning rectangles, and in the meantime simplify these corners of $\mathcal{V}$ (relocate the x-coordinates), as explained in the previous section, while storing them. We temporarily pause the horizontal sweep, and start a vertical sweep within $\mathcal{H}$ after initializing $Y_{\mathcal{H}}$ using the value of the horizontal separator between $\mathcal{H}$ and the stripe above it. During this vertical sweep, we calculate the vertical width $\mathcal{W}_v$ of the area covered by the horizontally spanning rectangles, and simplify the stored corners of $\mathcal{V}$ (this time, relocate the y-coordinates). Since the corners within $\mathcal{V}$ fit in the working storage, we compute Klee's measure of the parts of the simplified rectangles within the cell $\mathcal{V}$ using Bentley's algorithm. We add the areas covered by the spanning vertical and the spanning horizontal rectangles to adjust the measure, and subtract the intersection area $\mathcal{W}_h \times \mathcal{W}_v$ that has been added twice. We repeatedly proceed with the horizontal sweep using $X_{\mathcal{H}}$ to identify and partially process a cell, then alternately initialize $Y_{\mathcal{H}}$ and perform a vertical sweep within $\mathcal{H}$ to finish the processing of the cell. After all the cells of a horizontal stripe are processed, we repeat the same actions for the next horizontal stripes in sequence. Since we correctly calculate the measure within every cell, the overall sum of all the local measures is what we are looking for.

Concerning the running time, we consider the time to produce the segments by the navigation piles. Recall that $X_{\mathcal{H}}$ sweeps over all the n corners, whereas $Y_{\mathcal{H}}$ sweeps only over the $O(n/m)$ corners of $\mathcal{H}$. The navigation piles X for the horizontal sweeps repeatedly process all the input for every horizontal stripe. Since we have a total of m such sweeps, the total time consumed by the X navigation piles is $O((n^2/s + n \lg s) \cdot m)$. The navigation piles Y for the vertical sweeps process the $O(n/m)$ corners of a horizontal stripe in one sweep. Therefore, the total time for each of these vertical sweeps is $O((n/s + \lg s) \cdot n/m + n)$. It is straightforward to verify that $n/s + \lg s = \Omega(m)$ for all considered values of n and s (it is either true that $n/s > m$ or otherwise $\lg s = \Omega(m)$). The total number of vertical sweeps done within each horizontal stripe is $O((n/m)/\ell)$, which is $O(m)$ since $m = \lceil \sqrt{(n/s) \cdot \lg n} \rceil$. It follows that the total time of the vertical sweeps within one horizontal stripe is $O(n^2/s + n \lg s)$. Multiplying by the number of horizontal stripes m, the total time consumed by the Y navigation piles is $O((n^2/s + n \lg s) \cdot m)$, matching the bound for the X piles. The time needed by the extended local version of Bentley's algorithm within each cell is $O(\ell \cdot \lg \ell)$, resulting in a total of $O(n \cdot \lg s)$ time for all the calls to Bentley's algorithm. The time for the navigation piles is dominating.

▶ **Theorem 8.** *Given a read-only array containing the corners of n axis-parallel rectangles, and the available workspace is $\Theta(s)$ bits, where $\lg n \leq s \leq n \lg n$, the measure can be computed in $O((n^2/s + n \lg s) \cdot \sqrt{(n/s) \cdot \lg n})$ time.*

10 Concluding Comments

We have given space-efficient plane-sweep algorithms for some basic geometric problems. We believe that the techniques we introduce cover a range of ideas to handle many other plane-sweep algorithms in a space-efficient manner. Another question is if it is possible to get around with the extra logarithmic factors in the running times of the problem of enumerating the general and the axis-parallel line-segments intersections. It also remains open if it is possible to solve the measure problem more efficiently when the input is not sorted.

References

1 Pankaj K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete Comput. Geom.*, 5(6):533–573, 1990.

2 Tetsuo Asano, Amr Elmasry, and Jyrki Katajainen. Priority queues and sorting for read-only data. In *Proc. 10th International Conference on Theory and Applications of Models of Computation (TAMC 2013)*, volume 7876 of *LNCS*, pages 32–41, 2013. doi:10.1007/978-3-642-38236-9_4.

3 Tetsuo Asano, Wolfgang Mulzer, Günter Rote, and Yajun Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.

4 Ivan J. Balaban. An optimal algorithm for finding segments intersections. In *Proc. 11th Symposium on Computational Geometry*, pages 211–219, 1995. doi:10.1145/220279.220302.

5 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. doi:10.1137/0220017.

6 Jon Louis Bentley. Algorithms for Klee's rectangle problems, 1977. Unpublished manuscript.

7 Jon Louis Bentley and Thomas Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computers*, 28(9):643–647, 1979. doi:10.1109/TC.1979.1675432.

8 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 2008.

9 Timothy M. Chan. Closest-point problems simplified on the RAM. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 472–473, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545444.

10 Timothy M. Chan. Klee's measure problem made easy. In *Proc. 54th Anual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

11 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

12 David Clark. *Compact Pat Trees.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1996.

13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 3rd edition, 2009.

14 Omar Darwish and Amr Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *Proc. 22nd Annual European Symposium on Algorithms (ESA 2014)*, volume 8737 of *LNCS*, pages 284–295, 2014. doi:10.1007/978-3-662-44777-2_24.

15 Amr Elmasry, Frank Kammer, and Torben Hagerup. Space-efficient basic graph algorithms. In *Proc. 32nd Annual Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, LIPIcs, pages 288–301, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

16 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987. doi:10.1016/0022-0000(87)90002-X.

17 Matsuo Konagaya and Tetsuo Asano. Reporting all segment intersections using an arbitrary sized work space. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 96-A(6):1066–1071, 2013. URL: http://search.ieice.org/bin/summary.php?id=e96-a_6_1066.

18 Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein. Time-space trade-offs for triangulations and Voronoi diagrams. In *Proc. 14th Algorithms and Data Structures Symposium (WADS 2015)*, 2015.

19 J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12(3):315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

20 Mark H. Overmars and Chee-Keng Yap. New upper bounds in Klee's measure problem (extended abstract). In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS 1988)*, pages 550–556, 1988. doi:10.1109/SFCS.1988.21971.

21 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pages 264–268, 1998. doi:10.1109/SFCS.1998.743455.

22 Andrew Chi-Chih Yao. Near-optimal time-space tradeoff for element distinctness. *SIAM J. Comput.*, 23(5):966–975, 1994. doi:10.1137/S0097539788148959.

Linear Kernels and Linear-Time Algorithms for Finding Large Cuts

Michael Etscheid*[1] and Matthias Mnich†[2]

1 Universität Bonn, Bonn, Germany
etscheid@cs.uni-bonn.de

2 Universität Bonn, Bonn, Germany
mmnich@uni-bonn.de

Abstract

The maximum cut problem in graphs and its generalizations are fundamental combinatorial problems. Several of these cut problems were recently shown to be fixed-parameter tractable and admit polynomial kernels when parameterized above the tight lower bound measured by the size and order of the graph. In this paper we continue this line of research and considerably improve several of those results:

- We show that an algorithm by Crowston et al. [ICALP 2012] for (Signed) Max-Cut Above Edwards-Erdős Bound can be implemented in such a way that it runs in *linear time* $8^k \cdot O(m)$; this significantly improves the previous analysis with run time $8^k \cdot O(n^4)$.
- We give an *asymptotically optimal* kernel for (Signed) Max-Cut Above Edwards-Erdős Bound with $O(k)$ vertices, improving a kernel with $O(k^3)$ vertices by Crowston et al. [COCOON 2013].
- We improve *all* known kernels for strongly λ-extendable properties parameterized above tight lower bound by Crowston et al. [FSTTCS 2013] from $O(k^3)$ vertices to $O(k)$ vertices.
- As a consequence, Max Acyclic Subdigraph parameterized above Poljak-Turzík bound admits a kernel with $O(k)$ vertices and can be solved in time $2^{O(k)} \cdot n^{O(1)}$; this answers an open question by Crowston et al. [FSTTCS 2012].

All presented kernels can be computed in time $O(km)$.

1998 ACM Subject Classification F2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Max-Cut, fixed-parameter tractability, kernelization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.31

1 Introduction

A recent paradigm in parameterized complexity is to not only show a problem to be fixed-parameter tractable, but indeed to give algorithms with *optimal* run times in *both* the parameter and the input size. Ideally, we strive for algorithms that are *linear* in the input size, and optimal in the dependence on the parameter k assuming a standard hypothesis such as the Exponential Time Hypothesis [17]. New results in this direction include fixed-parameter algorithms for Graph Bipartization [18, 30], Planar Subgraph Isomorphism [9], DAG Partitioning [29] and Subset Feedback Vertex Set [20].

Here, we consider the fundamental Max-Cut problem from the view-point of linear-time fixed-parameter algorithms. In this classical NP-complete problem [19], the task is to find a

* Supported by ERC Starting Grant 306465 (BeyondWorstCase).
† Supported by ERC Starting Grant 306465 (BeyondWorstCase).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 31; pp. 31:1–31:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

bipartite subgraph of a given graph G with the maximum number $\mathsf{mc}(G)$ of edges. We refer to the survey [26] for an overview of the research area.

We focus on Max-Cut *parameterized above Edwards-Erdős bound.* This parameterization is motivated by the classical result of Edwards [10, 11] that any connected graph on n vertices and m edges admits a cut of size at least

$$m/2 + (n-1)/4 \ . \tag{1}$$

This lower bound is known as the *Edwards-Erdős bound*, and it is tight for cliques of every odd order n. Ngọc and Tuza [24] gave a *linear-time* algorithm that finds a cut of size at least (1).

Parameterizing Max-Cut above Edwards-Erdős bound means, for a given connected graph G and integer k, to determine if G admits a cut that exceeds (1) by an amount of k: formally, the problem Max-Cut Above Edwards-Erdős Bound (Max-Cut AEE) is to determine if $\mathsf{mc}(G) \geq |E(G)|/2 + (|V(G) - 1 + k)/4$. It was asked in a sequence of papers [5, 12, 21, 22] whether Max-Cut AEE is fixed-parameter tractable, before Crowston et al. [7] gave an algorithm that solves instances of this problem in time $8^k \cdot O(n^4)$, as well as a kernel of size $O(k^5)$. Their result inspired a lot of further research on this problem, leading to smaller kernels of size $O(k^3)$ [4] and fixed-parameter algorithms for generalizations [23] and variants [8].

In the Signed Max-Cut problem, we are given a graph G whose edges are labeled by $(+)$ or $(-)$, and we seek a maximum balanced subgraph H of G, where *balanced* means that each cycle has an even number of negative edges. Max-Cut is the special case where all edges are negative. Signed Max-Cut finds applications in, e.g., modeling social networks [14], statistical physics [1], portfolio risk analysis [15], and VLSI design [3]. The dual parameterization of Signed Max-Cut by the number of edge deletions was also shown to be fixed-parameter tractable [16].

Poljak and Turzík [25] showed that the property of having a large cut (i.e., a large bipartite subgraph) can be generalized to many other classical graph properties, including properties of oriented and edge-labeled graphs. They defined the notion of "λ-extendable" properties Π and generalized the lower bound (1) to tight lower bounds for all such properties; we refer to these lower bounds as the *Poljak-Turzík bound* for Π. Well-known examples of such properties include bipartite subgraphs, q-colorable subgraphs for fixed q, or acyclic subgraphs of oriented graphs. Mnich et al. [23] considered the problem Above Poljak-Turzík(Π) of finding subgraphs in Π with k edges above the *Poljak-Turzík bound*; they gave fixed-parameter algorithms for this problem on all "strongly" λ-extendable properties Π. A subclass of these, requiring certain technical conditions, was later shown to admit polynomial kernels [8].

1.1 Our Contributions

Linear-Time FPT. Our first result is that the fixed-parameter algorithm given by Crowston et al. [4] for the Signed Max-Cut AEE problem can be implemented in such a way that it runs in linear time.

▶ **Theorem 1** ($\star$). *The* (Signed) Max-Cut AEE *problem can be solved in time* $8^k \cdot O(m)$.

Theorem 1 considerably improves the earlier run time analysis [4, 7], which shows a run time of $8^k \cdot O(n^4)$. At the same time, our algorithm improves the very involved algorithm by Bollobás and Scott [2] that considers the weaker lower bound $m/2 + (\sqrt{8m+1} - 1)/8$ instead of (1). Third, Theorem 1 generalizes the linear-time algorithm by Ngọc and Tuza [24] for the special case of Max-Cut with $k = 0$. Note that Max-Cut AEE cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$ assuming the Exponential Time Hypothesis [7].

Linear Vertex Kernels. Our second contribution is a kernel with a linear number $O(k)$ of vertices for Max-Cut AEE and its generalization Signed Max-Cut AEE.

▶ **Theorem 2.** *The* (Signed) Max-Cut AEE *problem admits a kernel with $O(k)$ vertices, which can be computed in time $O(km)$.*

These results considerably improve the previous best kernel bound of $O(k^3)$ vertices by Crowston et al. [4]. Moreover, the presented kernel completely resolves the asymptotic kernelization complexity of (Signed) Max-Cut AEE, since a kernel with $o(k)$ vertices would again contradict the Exponential-Time Hypothesis, as the Max-Cut problem can be solved by checking all vertex bipartitions. On top of that, our kernelization is also *fast*. In fact, we only need to compute $O(k)$ DFS/BFS trees. The rest of the algorithm runs in time $O(m + n)$.

Extensions to Strongly λ-Extendable Properties. As mentioned, the property of graphs having large bipartite subgraphs can be generalized to λ-extendable properties as defined by Poljak and Turzík [25] (we defer the formal definitions to Section 2). For a given λ-extendable property Π, we consider the following problem.

Above Poljak-Turzík Bound(Π)

Input: A connected graph G and an integer k.

Question: Does G have a spanning subgraph $H \in \Pi$ s.t. $|E(H)| \geq \lambda \cdot |E(G)| + \frac{1-\lambda}{2} \cdot (|V(G)| - 1) + k$?

Note the slight change in the definition of k compared to (Signed) Max-Cut AEE, where k was divided by $4 = \frac{2}{1-\lambda}$ for $\lambda = \frac{1}{2}$.

Crowston et al. [4] gave polynomial kernels with $O(k^3)$ or $O(k^2)$ vertices for the problem Above Poljak-Turzík(Π), for all strongly λ-extendable properties Π on possibly oriented and/or labeled graphs satisfying at least one of the following properties.

(P1) $\lambda \neq \frac{1}{2}$; or
(P2) $G \in \Pi$ for all graphs G whose underlying simple graph is K_3; or
(P3) Π is a hereditary property of simple or oriented graphs.

Our third result improves *all* these kernels for strongly λ-extendable properties to asymptotically optimal $O(k)$ vertices:

▶ **Theorem 3.** *Let Π be any strongly λ-extendable property of (possibly oriented and/or labeled) graphs satisfying (P1), or (P2), or (P3). Then* Above Poljak-Turzík(Π) *admits a kernel with $O(k)$ vertices, which is computable in time $O(km)$.*

Consequences for Acyclic Subdigraphs. Theorem 3 has several applications. For instance, Raman and Saurabh [27] asked for the parameterized complexity of the Max Acyclic Subdigraph problem above the Poljak-Turzík bound: Given a weakly connected oriented graph G on n vertices and m arcs, does it have an acyclic sub-digraph of at least $m/2 + (n-1)/4 + k$ arcs? For this problem, Crowston et al. [6] gave an algorithm with run time $2^{O(k \log k)} \cdot n^{O(1)}$ and showed a kernel with $O(k^2)$ vertices. They explicitly asked whether the kernel size can be improved to $O(k)$ vertices, and whether the run time can be improved to $2^{O(k)} \cdot n^{O(1)}$. Here, we answer their questions in the affirmative by using Theorem 3 and then applying an $O^*(2^n)$-time algorithm by Raman and Saurabh [28, Thm. 2] to our kernel with $O(k)$ vertices.

▶ **Corollary 4.** *The* Max Acyclic Subdigraph *problem parameterized above Poljak-Turzík bound admits a kernel with $O(k)$ vertices and can be solved in time $2^{O(k)} \cdot n^{O(1)}$.*

Again, assuming the Exponential Time Hypothesis, the run time of this algorithm is asymptotically optimal.

Due to space constraints, proofs of statements marked by ($\star$) are deferred to the full version.

2 Preliminaries

We use $\uplus$ to denote the disjoint union of sets. The term "graph" refers to finite undirected graphs without self-loops, parallel edges, edge directions, or labels. For a graph G, let $V(G)$ denote its set of vertices and let $E(G)$ denote its set of edges. In an oriented graph, each edge $e = \{u, v\}$ has one of two directions, $\overrightarrow{e} = (u, v)$ and $\overleftarrow{e} = (v, u)$; thus, an oriented graph is a digraph without 2-cycles and loops. We sometimes write an edge $e = \{u, v\}$ as $e = uv$, if no confusion arises; this way, three distinct vertices a, b, c can *induce* a triangle $abca$. In a labeled graph, each edge in $E(G)$ receives one of a constant number of labels. For an oriented and/or labeled graph G, let $\langle G\rangle$ denote the underlying simple graph obtained from omitting orientations and/or labels. Throughout the paper, we assume graphs to be encoded as adjacency lists.

A graph is *connected* if there is a path between any two of its vertices. A *connected component* of G is a maximal connected subgraph of G. A *cut vertex* of a graph G is a vertex whose removal increases the number of connected components. A graph is *2-connected* if it does not contain any cut vertices. A maximal 2-connected subgraph of a graph G is called a *block* of G. A block that contains at most one cut vertex of G is called a *leaf block* of G. A *clique tree* is a connected graph whose blocks are *cliques*, where a clique is a complete subgraph of a graph. A *clique forest* is a graph whose connected components are clique trees.[1] For an oriented and/or labeled graph G we say that G has one of the above-defined properties if $\langle G\rangle$ does.

Let G be a graph. For a vertex $v \in V(G)$, let $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. For signed graphs G, we define $N_G(v) = N_{\langle G\rangle}(v)$. For a vertex set $V' \subseteq V(G)$, let $N_G(V') = (\bigcup_{v\in V'} N_G(v)) \setminus V'$. For disjoint vertex sets $V_1, V_2 \subseteq V(G)$, let $E(V_1, V_2)$ denote the set of edges with one endpoint in V_1 and the other endpoint in V_2. For signed graphs G, let $E^+(G) \subseteq E(G)$ be the edges with positive labels, and $E^-(G) = E(G) \setminus E^+(G)$ be the edges with negative labels. Define $N^+_G(v) = \{u \in V(G) \mid vu \in E^+(G)\}$ and $N^-_G(v) = \{u \in V(G) \mid vu \in E^-(G)\}$ for all $v \in V(G)$.

A *graph property* Π is simply a set of graphs. For a graph G, a Π-subgraph is a subgraph of G that belongs to Π. A graph property Π is *hereditary* if for any $G \in \Pi$ also all vertex-induced subgraphs of G belong to Π. Poljak and Turzík [25] defined the notion of "λ-extendability" for graph properties Π, and proved a lower bound on the size of any Π-subgraph in arbitrary graphs. A related notion of "strong λ-extendability" was introduced by Mnich et al. [23]; any strongly λ-extendable property is λ-extendable, but it is unclear whether the other direction holds.

[1] Clique forests are sometimes called *block graphs*; however, there are competing definitions for this term in the literature and so we refrain from using it.

▶ **Definition 5.** Let $\mathcal{G}$ be a class of (possibly labeled and/or oriented) graphs and let $\lambda \in (0,1)$. A (graph) property Π is *strongly λ-extendable* on $\mathcal{G}$ if it satisfies the following properties:

(i) *inclusiveness*: $\{G \in \mathcal{G} \mid \langle G \rangle \in K_1, K_2\} \subseteq \Pi$.

(ii) *block additivity*: $G \in \mathcal{G}$ belongs to Π if and only if each block of G belongs to Π.

(iii) *extendability*: For any $G \in \mathcal{G}$ and any partition $U \uplus W$ of $V(G)$ for which $G[U], G[W] \in \Pi$ there is a set $F \subseteq E(U,W)$ of size $|F| \geq \lambda |E(U,W)|$ for which $G - (E(U,W) \setminus F) \in \Pi$.

The set of all bipartite graphs $\Pi_{\text{bipartite}}$ is a strongly $\frac{1}{2}$-extendable property. Thus, Max-Cut AEE is equivalent to Above Poljak-Turzík Bound($\Pi_{\text{bipartite}}$).

Poljak and Turzík[25] showed that, given a (strongly) λ-extendable property Π, any connected graph G contains a subgraph H with at least $\lambda|E(G)| + \frac{1-\lambda}{2}(|V(G)|-1)$ edges such that $H \in \Pi$. We denote this lower bound by $\mathsf{pt}(G)$. Further, we define the *excess* of G over this lower bound with respect to Π as $\mathsf{ex}(G) = \max\{|E(H)| - \mathsf{pt}(G) \mid H \subseteq G, H \in \Pi\}$. When considering properties of labeled and/or oriented graphs, we denote by $\mathsf{ex}(K_t)$ the minimum value of $\mathsf{ex}(G)$ over all labeled and/or oriented graphs G with $\langle G \rangle = K_t$; here, K_t denotes the complete graph of order t. (Our definition slightly differs from the one by Crowston et al. [8].)

A strongly λ-extendable property Π *diverges on cliques* if $\mathsf{ex}(K_j) > \frac{1-\lambda}{2}$ for some $j \in \mathbb{N}$. For example, every strongly λ-extendable property with $\lambda \neq \frac{1}{2}$ diverges on cliques [8]. We recall the following fact about diverging properties:

▶ **Proposition 6** ([8, Lemma 8]). *Let Π be a strongly λ-extendable property diverging on cliques, and let $j \in \mathbb{N}, a > 0$ be such that $\mathsf{ex}(K_j) = \frac{1-\lambda}{2} + a$. Then $\mathsf{ex}(K_i) \geq ra$ for all $i \geq rj$.*

We need the following proposition in all sections. For Signed Max Cut, we will apply it with $\lambda = \frac{1}{2}$.

▶ **Proposition 7** ([8, Lemma 6]). *Let Π be a strongly λ-extendable property, let G be a connected graph and let $U_1 \uplus U_2$ be a partition of $V(G)$ into non-empty sets U_1, U_2. For $i = 1,2$ let c_i be the number of connected components of $G[U_i]$. If $\mathsf{ex}(G[U_i]) \geq k_i$ for some $k_i \in \mathbb{R}$ and $i = 1,2$, then $\mathsf{ex}(G) \geq k_1 + k_2 - \frac{1-\lambda}{2}(c_1 + c_2 - 1)$.*

3 Linear-Time Fixed-Parameter Algorithms and Linear Vertex Kernels for Signed Max Cut

In this section we consider the Signed Max-Cut AEE problem. We show that the fixed-parameter algorithm given by Crowston et al. [4] can be implemented in such a way that it runs in time $8^k \cdot O(|E(G)|)$. That is, given a connected graph G whose edges are labeled either positive $(+)$ or negative $(-)$, and an integer k, we can decide in time $8^k \cdot O(|E(G)|)$ whether G has a balanced subgraph of size $|E(G)|/2 + (|V(G)| - 1 + k)/4$. This will prove Theorem 1. In the second part of the section we will show how to obtain a kernel with $O(k)$ vertices and thus prove Theorem 2.

Let us first reformulate the Signed Max-Cut AEE problem.

▶ **Proposition 8** (Harary [13]). *A signed graph G is balanced if and only if there exists a partition $V_1 \uplus V_2 = V(G)$ such that all edges in $G[V_1]$ and $G[V_2]$ are positive and all edges $E(V_1, V_2)$ between V_1 and V_2 are negative.*

3.1 Linear-Time Fixed-Parameter Algorithm

The algorithm by Crowston et al. [4] starts by applying the following seven reduction rules. We restate them here, as they are crucial for our results. A reduction rule is *1-safe* if, on input

(G, k) it returns a pair (G', k') such that (G, k) is a "yes"-instance for Signed Max-Cut AEE if (G', k') is. (Note that the converse direction does not have to hold.) In a signed graph G we call a triangle *positive* if its number of negative edges is even. In the description of the rules, G is always a connected signed graph and C is always a clique that does not contain a positive triangle.

▶ **Reduction Rule 9.** *If abca is a positive triangle such that $G - \{a, b, c\}$ is connected, then mark a, b, c, delete them, and set $k' = k - 3$.*

▶ **Reduction Rule 10.** *If abca is a positive triangle such that $G - \{a, b, c\}$ has exactly two connected components C and Y, then mark a, b, c, delete them, delete C, and set $k' = k - 2$.*

▶ **Reduction Rule 11.** *Let C be a connected component of $G - v$ for some vertex $v \in V(G)$. If there exist $a, b \in V(C)$ such that $G - \{a, b\}$ is connected and there is an edge av but no edge bv, then mark a, b, delete them, and set $k' = k - 2$.*

▶ **Reduction Rule 12.** *Let C be a connected component of $G - v$ for some vertex $v \in V(G)$. If there exist $a, b \in C$ such that $G - \{a, b\}$ is connected and vabv is a positive triangle, then mark a, b, delete them, and set $k' = k - 4$.*

▶ **Reduction Rule 13.** *If there is a vertex $v \in V(G)$ such that $G - v$ has a connected component C such that $G[V(C) \cup \{v\}]$ is a clique that does not contain a positive triangle, then delete C. If $|V(C)|$ is odd, then set $k' = k - 1$. Otherwise, set $k' = k$.*

▶ **Reduction Rule 14.** *If abc is a vertex-induced path in G for some vertices $a, b, c \in V(G)$ such that $G - \{a, b, c\}$ is connected, then mark a, b, c, delete them, and set $k' = k - 1$.*

▶ **Reduction Rule 15.** *Let C, Y be the connected components of $G - \{v, b\}$ for some vertices $v, b \in V(G)$ such that $vb \notin E(G)$. If $G[V(C) \cup \{v\}]$ and $G[V(C) \cup \{b\}]$ are cliques that do not contain a positive triangle, then mark v, b, delete them, delete C, and set $k' = k - 1$.*

We slightly changed Rule 13. Crowston et al. [4] always set $k' = k$, whereas we set $k' = k-1$ when $|V(C)|$ is odd. In this case, $\mathsf{pt}(G[V(C) \cup \{v\}])$ cannot be integral because $|V(C) \cup \{v\}|$ is even, and thus $\mathsf{ex}(G[V(C) \cup \{v\}]) \geq \frac{1}{4}$. Therefore our change for k is 1-safe due to the following result.

▶ **Proposition 16** ([4, Lemma 2])**.** *Let G be a connected signed graph and Z be a connected component of $G - v$ for some $v \in V(G)$. Then $\mathsf{ex}(G) = \mathsf{ex}(G - Z) + \mathsf{ex}(G[V(Z) \cup \{v\}])$.*

We subsume the results by Crowston et al. [4] in the following proposition.

▶ **Proposition 17** ([4])**.** *Rules 9–15 are 1-safe. To any connected signed graph with at least one edge, one of these rules applies and the resulting graph is connected. If S is the set of vertices marked during the exhaustive application of Rules 9–15 on a connected signed graph G, then $G - S$ is a clique forest. If $|S| > 3k$, then (G, k) is a "yes"-instance.*

Following Crowston et al. [4, Corollary 3], we assume – without loss of generality – from now on that the resulting clique forest $G - S$ does not contain a positive edge.

▶ **Lemma 18** ($\star$)**.** *Let G be a connected signed graph, let X be a leaf block of G, and let $r \in V(G)$ such that $V(X) \setminus \{r\}$ does not contain a cut vertex of G. Then we can apply one of the Rules 9–15 to G deleting and marking only vertices from X in time $O(|E(X)|)$.*

Given an instance (G, k), we can thus compute in time $O(k \cdot |E(G)|)$ a vertex set S that either proves that (G, k) is a "yes"-instance or $G - S$ is a clique forest. We now show that, if a partition for the vertices in S is already given, we can in time $O(|E(G)|)$ compute an optimal extension to G. We use the following problem, which goes back to Crowston et al. [7].

MAX-CUT EXTENSION

Input: A clique forest G_S with weight functions $w_i : V(G_S) \to \mathbb{N}_0$ for $i = 0, 1$.

Task: Find an assignment $\varphi : V(G_S) \to \{0, 1\}$ maximizing $\sum_{xy \in E(G_S)} |\varphi(x) - \varphi(y)| + \sum_{i=0}^{1} \sum_{x:\, \varphi(x)=i} w_i(x)$.

▶ **Lemma 19** (⋆). MAX-CUT EXTENSION *can be solved in time $O(|V(G_S)| + |E(G_S)|)$ on a clique forest G_S.*

We now give a proof sketch for Theorem 1. Lemma 18 allows us to find the set S from Proposition 17 in time $O(km)$ (the case that k is not decreased can only take $O(m)$ total time). Guess one of the at most 2^{3k} partitions on S and solve the corresponding MAX-CUT EXTENSION problem with Lemma 19.

3.2 A Linear Vertex Kernal for Signed Max-Cut AEE

For the whole section, let G^0 be the original graph, let S be the set of marked vertices during the exhaustive application of Rules 9–15 on G^0, and let G^r be the resulting graph after the exhaustive application of our kernelization Rules 20–21 (to be defined later) on G^0.

If there is a (unique by Proposition 17) remaining vertex v left after the exhaustive application of Rules 9–15, then add a path vwx to G, i.e., define $G' = (V(G) \cup \{w, x\}, E(G) \cup \{vw, wx\})$. Then $(G', k + 2)$ is an instance of MAX-CUT AEE that is due to Proposition 16 equivalent to (G, k) because the excess of a path of length 2 is 2/4. This implies that we can w.l.o.g. assume that every vertex gets removed during the exhaustive application of the reduction rules because we can assume we finish with deleting the new path with Rule 14. Furthermore, as Rule 13 can then not be applied last, we can assume that at least one of the vertices that are removed last is contained in S.

We will use two-way reduction rules which are similar to the two-way reduction rules by Crowston et al. [4]. However, our two-way reduction rules have the property that connected components of $G - S$ cannot fall apart, i.e., two blocks in $G^r - S$ are reachable from each other if and only if the corresponding blocks in $G^0 - S$ are reachable from each other. We can thus show that Rules 9–15 can behave "equivalently" on G^r as on G^0 (Lemma 24), i.e., that the same set S of vertices can also be marked in G^r. This is the crucial idea which allows us to obtain better kernelization results than previous papers, as it allows the following analysis.

To show size bounds for our kernel G^r, we (hypothetically) change the set of rules in such a way that whenever a vertex $s \in S$ is about to be removed, we additionally remove internal vertices from different blocks of $G^r - S$ that are all adjacent to s. This means that for every $s \in S$, we find a star-like structure Y_s such that Y_s is removed together with s, and the excess on Y_s grows linearly in $|Y_s|$. We can distribute the internal vertices from $G - S$ in such a way to the different Y_s that all generated graphs are still connected. Then the large excess of the different Y_s translates to a large excess of G^r through Proposition 7.

We use this approach twice to first bound the number of special blocks (Lemma 25) and then the number of internal vertices in special blocks (Lemma 27) to $O(k)$. On the other hand, due to Rules 20–21 a constant fraction of vertices in $G^r - S$ must be adjacent to S. This completes the proof.

Let C be a block in the clique forest $G - S$. Define $C_{\text{int}} = \{v \in V(C) \mid N_{G-S}(v) \subseteq V(C)\}$ as the *interior* of C, and $C_{\text{ext}} = V(C) \setminus C_{\text{int}}$ as the *exterior* of C. The block C is called *special* if $C_{\text{int}} \cap N_G(S)$ is non-empty. Let $\mathcal{B}$ be the set of blocks and $\mathcal{B}_s$ be the set of special blocks in $G^r - S$. A block C is a Δ*-block* if it is not special, contains exactly three vertices, and $|C_{\text{ext}}| \leq 2$.

We now give our two-way reduction rules, which on input (G, k) produce an instance (G', k) of Signed Max-Cut AEE. Note that the parameter k does not change. We call a rule *2-safe* if (G, k) is a "yes"-instance if and only if (G', k) is. The first rule is again due to Crowston et al. [4], who showed it to be 2-safe. The run time analysis is our work. Recall our assumption that (without loss of generality) $G - S$ does not contain any positive edges.

▶ **Reduction Rule 20.** *Let C be a block in $G - S$. If there exists $X \subseteq C_{\text{int}}$ such that $|X| > \frac{|V(C)|+|N_G(X)\cap S|}{2} \geq 1$, $N^+_G(x) \cap S = N^+_G(X) \cap S$ and $N^-_G(x) \cap S = N^-_G(X) \cap S$ for all $x \in X$, then delete two arbitrary vertices $x_1, x_2 \in X$.*

▶ **Reduction Rule 21.** *Let C_1, C_2 be two Δ-blocks in $G - S$ which share a common vertex v. Make a block out of $V(C_1) \cup V(C_2)$, i.e., add negative edges $\{\{u, w\} \mid u \in V(C_1) \setminus \{v\}, w \in V(C_2) \setminus \{v\}\}$ to G.*

▶ **Lemma 22** ($\star$). *Rules 20–21 are 2-safe. If they are applied to a connected graph G, then the resulting graph G' is also connected.*

▶ **Lemma 23** ($\star$). *Given S, Rules 20–21 can be applied exhaustively to G^0 in total time $O(m+n)$.*

▶ **Lemma 24** ($\star$). *Rules 9–15 can be applied exhaustively to the graph G^r in such a way that the set S' of marked vertices is equal to S. Moreover, if only the Rules 11/13/14/15 are applied to G^0, the same set of rules is applied to G^r.*

The last part of the lemma will be needed later in Section 4.2.

▶ **Lemma 25** ($\star$). *If $G^r - S$ has more than $11k$ special blocks, then (G^r, k) is a "yes"-instance of* Signed Max-Cut AEE*.*

▶ **Lemma 26** ($\star$). *If $G^r - S$ has more than $48k$ blocks, then (G^r, k) is a "yes"-instance of* Signed Max-Cut AEE*. Otherwise, $G^r - S$ has at most $48k$ external vertices, and $\sum_{B\in\mathcal{B}} |B_{\text{ext}}| \leq 96k$.*

▶ **Lemma 27** ($\star$). *If there are more than $117k$ internal vertices in special blocks in $G^r - S$, then (G^r, k) is a "yes"-instance of* Signed Max-Cut AEE*.*

We are now ready to prove Theorem 2.

Proof of Theorem 2. Let (G^0, k) be an instance of Signed Max-Cut AEE. Like in Section 3.1, apply Rules 9–15 exhaustively to (G^0, k) in time $O(k \cdot |E(G^0|)$, producing an instance (G', k') and a vertex set S of marked vertices. If $k' \leq 0$, then (G', k') and thus also (G, k) is a "yes"-instance.

Now apply Rules 20–21 exhaustively to (G^0, k) in time $O(|E(G)|)$ (Lemma 23) to obtain an equivalent instance (G^r, k). Check whether (G^r, k) is a "yes"-instance due to Lemma 26 or Lemma 27. If this is not the case, then there are at most $3k$ vertices in S, at most $48k$ external vertices in $G^r - S$ and at most $117k$ internal vertices in special blocks. If there were more internal than external vertices in a non-special block, we could apply Rule 20 to this block. Thus, the number of internal vertices in non-special blocks is bounded by $96k$ according to Lemma 26. Hence, the total number of vertices in G^r is bounded by $3k + 48k + 117k + 96k = 264k$. ◀

4 Linear Vertex Kernels for λ-Extendable Properties

In this section we extend our linear kernels for Signed Max-Cut to all strongly λ-extendable properties satisfying (P1), or (P2), or (P3). Henceforth, fix a strongly λ-extendable property Π, and let (G^0, k) be an instance of Above Poljak-Turzík Bound(Π). For notational brevity, we assume the empty graph to be in Π.

As in the previous section, we use a set of 1-safe reduction rules devised by Mnich et al. [23] to find a set S such that $G^0 - S$ is a clique forest; the difference compared to Signed Max-Cut is the different change of k. Since we change the reduction rules slightly in the next section, we refrain from stating the rules by Mnich et al. here.

▶ **Lemma 28** ([23]). *There is an algorithm that, given a connected graph G and $k \in \mathbb{N}$, either decides that $\mathsf{ex}(G) \geq k$, or finds a set S of at most $\frac{6k}{1-\lambda}$ vertices such that $G - S$ is a clique forest. This also holds for all strongly λ-extendable properties of oriented and/or labeled graphs.*

The detection which of the reduction rules can be applied to a graph G is completely analogous to the Signed Max-Cut reduction rules. Hence, it follows immediately from Lemma 18 that the rules can be applied exhaustively in time $O(km)$.

4.1 Linear Kernel for Properties Diverging on Cliques

We show that Above Poljak-Turzík Bound(Π) admits kernels with $O(k)$ vertices for all strongly λ-extendable properties Π that are diverging on cliques and for which $\mathsf{ex}(K_i) > 0$ for all $i \geq 2$.

▶ **Lemma 29** ($\star$). *Let Π be a strongly λ-extendable property diverging on cliques, and suppose that $\mathsf{ex}(K_i) > 0$ for all $i \geq 2$. Then* Above Poljak-Turzík Bound(Π) *admits a kernel with $O(k)$ vertices.*

▶ **Theorem 30.** *Let Π be a strongly λ-extendable property. If $\lambda \neq \frac{1}{2}$ or $G \in \Pi$ for every G with $\langle G \rangle = K_3$, then* Above Poljak-Turzík Bound(Π) *has a kernel with $O(k)$ vertices.*

Proof. Lemmas 24-26 from Crowston et al. [8] show that if $\lambda \neq \frac{1}{2}$ or $K_3 \in \Pi$, then Π diverges on cliques and $\mathsf{ex}(K_i) > 0$ for all $i \geq 2$. Therefore, we can apply Lemma 29. ◀

4.2 Strongly $\frac{1}{2}$-Extendable Properties on Oriented Graphs

We now turn to strongly $\frac{1}{2}$-extendable properties Π on oriented graphs. First of all we modify the reduction rules by Mnich et al. [23] in such a way that they are compliant with Rules 9–15. Let G always be a connected graph.

▶ **Reduction Rule 31.** *Let C be a connected component of $G - v$ for some vertex $v \in V(G)$ such that $G[V \cup \{v\}]$ is a clique. Delete C and set $k' = k$.*

▶ **Reduction Rule 32.** *Let C be a connected component of $G - v$ for some vertex $v \in V(G)$ such that C is a clique. If there exist $a, b \in V(C)$ such that $G - \{a, b\}$ is connected and $av \in E(G)$, but $bv \notin E(G)$, then mark a, b, delete them, and set $k' = k - \frac{1}{2}$.*

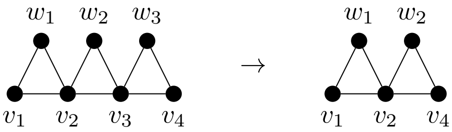


Figure 1 Illustration of Rule 38.

▶ **Reduction Rule 33.** *Let abc be a vertex-induced path for some vertices $a, b, c \in V(G)$ such that $G - \{a, b, c\}$ is connected. Mark a, b, c, delete them, and set $k' = k - \frac{1}{4}$.*

▶ **Reduction Rule 34.** *Let $v, b \in V(G)$ such that $vb \notin E(G)$ and $G - \{v, b\}$ has exactly two connected components C, Y. If $G[V(C) \cup \{v\}]$ and $G[V(C) \cup \{b\}]$ are cliques, then mark v, b, delete them, delete C, and set $k' = k - \frac{1}{4}$.*

Rules 31–34 are exactly Rules 13/11/14/15 for Signed Max-Cut AEE with all edges negative.

▶ **Lemma 35** ($\star$). *Rules 31–34 are 1-safe. To any connected graph with at least one edge, one of the rules applies and the resulting graph is connected. If S is the set of marked vertices, then $G - S$ is a clique forest. If $|S| > 12k$, then (G, k) is a "yes"-instance.*

Like Crowston et al. [8], we restrict ourselves to hereditary properties. Let $\overrightarrow{K}_3$ be the orientation of K_3 which is an oriented cycle, and let $\overset{\rightharpoonup\!\!\!\!\nrightarrow}{K}_3$ be the only (up to isomorphisms) other orientation of K_3. Crowston et al. [8] showed that if $\overrightarrow{K}_3 \in \Pi$, then also $\overset{\rightharpoonup\!\!\!\!\nrightarrow}{K}_3 \in \Pi$, and thus Theorem 30 applies. We now consider the case that $\overrightarrow{K}_3 \notin \Pi$ together with $\overset{\rightharpoonup\!\!\!\!\nrightarrow}{K}_3 \in \Pi$.

▶ **Proposition 36** ([8]). *Let Π be a hereditary strongly $\frac{1}{2}$-extendable property on oriented graphs with $\overset{\rightharpoonup\!\!\!\!\nrightarrow}{K}_3 \in \Pi$. Then $\mathsf{ex}(K_i) > 0$ for all $i \geq 4$ and Π diverges on cliques.*

Following this lemma, the conditions of Lemma 29 are almost satisfied. The only oriented cliques without positive excess are K_1 and $\overrightarrow{K}_3$, because $\mathsf{ex}(K_2) = \frac{1}{4}$ for $\frac{1}{2}$-extendable properties. Blocks isomorphic to K_1 can only occur as isolated vertices in $G - S$. We can bound these like in the previous section. Hence, we only need reduction rules to bound the number of blocks B in a clique forest with $B \cong \overrightarrow{K}_3$.

Let Π be a hereditary strongly $\frac{1}{2}$-extendable property on oriented graphs with $\overset{\rightharpoonup\!\!\!\!\nrightarrow}{K}_3 \in \Pi$. Let (G^0, k) be an instance of Above Poljak-Turzík(Π). Lemma 35 either proves that (G^0, k) is a "yes"-instance, or it finds a set S of at most $12k$ vertices such that $G^0 - S$ is a clique forest. Starting with (G^0, k), we apply the following reduction rules, which on input (G, k) produce an equivalent instance (G', k).

▶ **Reduction Rule 37.** *Delete B_{int} of leaf blocks B in $G - S$ with $B \cong \overrightarrow{K}_3$ and $N_G(S) \cap B_{\text{int}} = \emptyset$.*

▶ **Reduction Rule 38.** *Let B_1, B_2, B_3 be non-leaf-blocks in $G - S$ and $v_1, \ldots, v_4 \in V(G)$ be such that* (i) *$v_i, v_{i+1} \in (B_i)_{\text{ext}}$ for all $i \in \{1, 2, 3\}$;* (ii) *$B_i \cong \overrightarrow{K}_3$ for all $i \in \{1, 2, 3\}$; and* (iii) *$N_G(\{v_2, v_3, w_1, w_2, w_3\}) = \{v_1, v_4\}$, where w_i is the internal vertex of B_i. Delete v_3 and w_3. Add edges $v_2 v_4$ and $w_2 v_4$.*

Intuitively speaking, Rule 38 takes three blocks in $G - S$ that form a "path" and are all isomorphic to $\overrightarrow{K}_3$. If all vertices except the "endpoints" v_1 and v_4 are not adjacent to S, then it is safe to delete one block. For an illustration, see Fig. 1.

▶ **Lemma 39** ($\star$). *Let Π be a hereditary strongly $\frac{1}{2}$-extendable property on oriented graphs with $\overset{\nrightarrow}{K}_3 \in \Pi$. Then Rules 37–38 are 2-safe. The resulting graphs are connected.*

From now on, let G^r be the resulting graph after the exhaustive application of Rules 37–38 on G^0. Rules 37–38 are special cases of Rules 20–21. Because Rules 31–34 are Rules 13/11/14/15 for Signed Max-Cut AEE with all edges negative, the next lemma follows from Lemma 24.

▶ **Lemma 40.** *Rules 31–34 can be applied exhaustively on the graph G^r in such a way that the set S' of vertices removed by their application is equal to S.*

Let $\mathcal{B}^+$ be the set of blocks of $G^r - S$ with positive excess, and let $\mathcal{B}^-$ be the other blocks, i.e., the blocks B with $B \cong \vec{K}_3$ or $B \cong K_1$. Let $R \subseteq V(G) \setminus S$ be the set of vertices that are only contained in exactly two blocks $B_1, B_2 \in \mathcal{B}^-$ such that $(B_1)_{\text{int}} = (B_2)_{\text{int}} = \emptyset$. Further, let $V^+ \subseteq V(G) \setminus S$ be the set of vertices in blocks with positive excess, V^- be the set of vertices in blocks from $\mathcal{B}^-$, and let $V^-_{\text{int}} \uplus V^-_{\text{ext}} = V^-$ be the set of internal and external vertices of blocks $B \in \mathcal{B}^-$, respectively. Note that V^+ and V^- may intersect.

▶ **Lemma 41** ($\star$). *It holds $|V^-| = O(|(R \cup V^-_{\text{int}}) \cap N_{G^r}(S)|)$. Furthermore, if $|(R \cup V^-_{\text{int}}) \cap N_{G^r}(S)| > 48k$, then (G^r, k) is a "yes"-instance.*

Using the same approach as in Section 4.1, one can show that $|V^+| = O(k)$ or (G^r, k) is a "yes"-instance. As Lemma 41 bounds $|V^-| = O(k)$ for every "no"-instance, and $V^+ \cup V^- \cup S = V(G^r)$, this suffices to prove the following result.

▶ **Theorem 42** ($\star$). *Let Π be a hereditary strongly $\frac{1}{2}$-extendable property on oriented graphs with $\overset{\nrightarrow}{K}_3 \in \Pi$. Then* Above Poljak-Turzík Bound(Π) *admits a kernel with $O(k)$ vertices.*

Proof of Theorem 3. Let $\lambda \in (0,1)$ and let Π be a strongly λ-extendable property of (possibly oriented and/or labeled) graphs. If $\lambda \neq \frac{1}{2}$ or $G \in \Pi$ for every G with $\langle G \rangle = K_3$, we can use Theorem 30. Otherwise, we only have to consider the case that Π is a hereditary property of simple or oriented graphs.

Consider the case that $\vec{K}_3 \in \Pi$ or $\overset{\nrightarrow}{K}_3 \in \Pi$. If $\vec{K}_3 \in \Pi$, then Crowston et al. [8] show that $\overset{\nrightarrow}{K}_3 \in \Pi$, i.e., we can use Theorem 30. And if $\overset{\nrightarrow}{K}_3 \in \Pi$, we use Theorem 42.

Now we may suppose that $G \notin \Pi$ for every G with $\langle G \rangle = K_3$. Then Crowston et al. [8] show that Π is the set of all bipartite graphs. Hence, in the case of simple graphs as well as if $\vec{K}_3, \overset{\nrightarrow}{K}_3 \notin \Pi$ for oriented graphs, we can use Theorem 2 to obtain a linear vertex kernel.

It is easy to see that Rules 37–38 can be applied exhaustively in time $O(m)$. As λ is constant and we can apply every other reduction rule in linear time, it follows a total run time of $O(\lambda \cdot km) = O(km)$. ◀

5 Discussion

For the classical (Signed) Max-Cut problem, and its wide generalization to strongly λ-extendable properties, parameterized above the classical Poljak-Turzík bound, we improved the run time analysis for a known fixed-parameter algorithm to $8^k \cdot O(m)$. We further improved all known kernels with $O(k^3)$ vertices for these problems to asymptotically optimal $O(k)$ vertices. We did not try to optimize the hidden constants, as the analysis is already quite cumbersome.

It remains an interesting question whether all positive results presented here extend to edge-weighted graphs, where each edge receives a positive integer weight and the number m of edges in the Edwards-Erdős bound (1) is replaced by the total sum of the edge weights.

Further, Mnich et al. [23] showed fixed-parameter tractability of Above Poljak-Turzík Bound(Π) for *all* strongly λ-extendable properties Π. However, the polynomial kernelization results by Crowston et al. [8] as well as in this paper do not seem to apply to the special case of non-hereditary $\frac{1}{2}$-extendable properties. Such properties Π exist; e.g., $\Pi = \{G \in \mathcal{G} \mid C \not\cong K_3$ for all 2-connected components C of $G\}$. Also, for $\frac{1}{2}$-extendable properties on labeled graphs we only showed a polynomial kernel for the special case of Signed Max-Cut. It would be desirable to avoid these restrictions.

References

1 Francisco Barahona. On the computational complexity of Ising spin glass models. *J. Phys. A*, 15(10):3241–3253, 1982. URL: http://stacks.iop.org/0305-4470/15/3241.

2 Béla Bollobás and Alexander Scott. Better bounds for Max Cut. In Béla Bollobás, editor, *Contemporary Combinatorics*, volume 10 of *Bolyai Soc. Math. Studies*, pages 185–246, 2002.

3 C. Chiang, A. B. Kahng, S. Sinha, X. Xu, and A. Z. Zelikovsky. Fast and efficient bright-field aapsm conflict detection and correction. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 26(1):115–126, Jan 2007. doi:10.1109/TCAD.2006.882642.

4 R. Crowston, G. Gutin, M. Jones, and G. Muciaccia. Maximum balanced subgraph problem parameterized above lower bound. *Theoret. Comput. Sci.*, 513:53–64, 2013. doi:10.1016/j.tcs.2013.10.026.

5 Robert Crowston, Michael Fellows, Gregory Gutin, Mark Jones, Frances Rosamond, Stéphan Thomassé, and Anders Yeo. Simultaneously Satisfying Linear Equations Over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 Parameterized Above Average. In *Proc. FSTTCS 2011*, volume 13 of *Leibniz Int'l Proc. Informatics*, pages 229–240. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.229.

6 Robert Crowston, Gregory Gutin, and Mark Jones. Directed Acyclic Subgraph Problem Parameterized above the Poljak-Turzík Bound. In *Proc. FSTTCS 2012*, volume 18 of *Leibniz Int'l Proc. Informatics*, pages 400–411. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.400.

7 Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the Edwards-Erdős bound. *Algorithmica*, pages 1–24, 2014. doi:10.1007/s00453-014-9870-z.

8 Robert Crowston, Mark Jones, Gabriele Muciaccia, Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Polynomial kernels for λ-extendible properties parameterized above the Poljak-Turzík bound. In *Proc. FSTTCS 2013*, volume 24 of *Leibniz Int'l Proc. Informatics*, pages 43–54. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.43.

9 Frederic Dorn. Planar Subgraph Isomorphism Revisited. In *Proc. STACS 2010*, volume 5 of *Leibniz Int'l Proc. Informatics*, pages 263–274. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2010.

10 C. S. Edwards. Some extremal properties of bipartite subgraphs. *Canad. J. Math.*, 25:475–485, 1973.

11 C. S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Recent Advances in Graph Theory*, pages 167–181, 1975.

12 Gregory Gutin and Anders Yeo. Note on maximal bisection above tight lower bound. *Inform. Process. Lett.*, 110(21):966–969, 2010. doi:10.1016/j.ipl.2010.08.001.

13 Frank Harary. On the notion of balance of a signed graph. *Michigan Math. J.*, 2:143–146 (1955), 1953–54.

14 Frank Harary. On the measurement of structural balance. *Behavioral Sci.*, 4:316–323, 1959.

15 Frank Harary, Meng-Hiot Lim, and Donald C. Wunsch. Signed graphs for portfolio analysis in risk management. *IMA J. Mgmt. Math.*, 13(3):201–210, 2002. doi:10.1093/imaman/13.3.201.

16 Falk Hüffner, Nadja Betzler, and Rolf Niedermeier. Separator-based data reduction for signed graph balancing. *J. Comb. Optim.*, 20(4):335–360, 2010. doi:10.1007/s10878-009-9212-2.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

18 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *Proc. SODA 2014*, pages 1749–1761, 2014. doi:10.1137/1.9781611973402.127.

19 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

20 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. In *Proc. ICALP 2015*, volume 9134 of *Lecture Notes Comput. Sci.*, pages 935–946. Springer, 2015. doi:10.1007/978-3-662-47672-7_76.

21 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. Technical Report TR97-033, Electronic Colloquium on Computational Complexity, 1997. URL: http://eccc.hpi-web.de/report/1997/033/.

22 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *J. Comput. System Sci.*, 75(2):137–153, 2009. doi:10.1016/j.jcss.2008.08.004.

23 Matthias Mnich, Geevarghese Philip, Saket Saurabh, and Ondřej Suchý. Beyond Max-Cut: λ-extendible properties parameterized above the Poljak-Turzík bound. *J. Comput. System Sci.*, 80(7):1384–1403, 2014. doi:10.1016/j.jcss.2014.04.011.

24 N. V. Ngọc and Zsolt Tuza. Linear-time approximation algorithms for the max cut problem. *Combin. Probab. Comput.*, 2(2):201–210, 1993. doi:10.1017/S0963548300000596.

25 Svatopluk Poljak and Daniel Turzík. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Math.*, 58(1):99–104, 1986. doi:10.1016/0012-365X(86)90192-5.

26 Svatopluk Poljak and Zsolt Tuza. Maximum cuts and large bipartite subgraphs. In *Combinatorial optimization (New Brunswick, NJ, 1992–1993)*, volume 20 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 181–244. Amer. Math. Soc., 1995.

27 Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoret. Comput. Sci.*, 351(3):446–458, 2006. doi:10.1016/j.tcs.2005.10.010.

28 Venkatesh Raman and Saket Saurabh. Improved fixed parameter tractable algorithms for two “edge” problems: MAXCUT and MAXDAG. *Inform. Process. Lett.*, 104(2):65–72, 2007. doi:10.1016/j.ipl.2007.05.014.

29 René van Bevern. *Fixed-Parameter Linear-Time Algorithms for* NP*-hard Graph and Hypergraph Problems Arising in Industrial Applications.* PhD thesis, TU Berlin, 2014.

30 Magnus Wahlström. Half-integrality, LP-branching and FPT algorithms. In *Proc. SODA 2014*, pages 1762–1781, 2014. doi:10.1137/1.9781611973402.128.

Universal Guard Problems*

Sándor P. Fekete[1], **Qian Li**[2], **Joseph S. B. Mitchell**[3], **and Christian Scheffer**[4]

1 **Department of Computer Science, TU Braunschweig, Braunschweig, Germany**
s.fekete@tu-bs.de

2 **Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY, USA**
qian.li.1@stonybrook.edu

3 **Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY, USA**
joseph.mitchell@stonybrook.edu

4 **Department of Computer Science, TU Braunschweig, Braunschweig, Germany**
c.scheffer@tu-bs.de

Abstract

We provide a spectrum of results for the *Universal Guard Problem*, in which one is to obtain a small set of points ("guards") that are "universal" in their ability to guard any of a set of possible polygonal domains in the plane. We give upper and lower bounds on the number of universal guards that are always sufficient to guard all polygons having a given set of n vertices, or to guard all polygons in a given set of k polygons on an n-point vertex set. Our upper bound proofs include algorithms to construct universal guard sets of the respective cardinalities.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Art Gallery Problem, universal guarding, polygonization, worst-case bounds, robust covering

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.32

1 Introduction

Problems of finding optimal covers are among the most fundamental algorithmic challenges that play an important role in many contexts. One of the best-studied prototypes in a geometric setting is the classic Art Gallery Problem (AGP), which asks for a small number of points ("guards") required for covering ("seeing") all of the points within a geometric domain. An enormous body of work on algorithmic aspects of visibility coverage and related problems (see, e.g., O'Rourke [24], Keil [19], and [25]) was spawned by Klee's question for worst-case bounds more than 40 years ago: How many guards are always sufficient to guard all of the points in a simple polygon having n vertices? The answer, as shown originally by Chvátal [4], and with a very simple and elegant proof by Fisk [11], is that $\lfloor n/3 \rfloor$ guards are always sufficient, and sometimes necessary, to guard a simple n-gon.

While Klee's question was posed about guarding an n-vertex *simple polygon*, a related question about *point sets* was posed at the 2014 NYU Goodman-Pollack Fest: Given a set S of n points in the plane, how many *universal* guards are sometimes necessary and always sufficient to guard any simple polygon with vertex set S? This problem, and several related

* Q. Li and J. S. B. Mitchell are partially supported by the National Science Foundation (CCF-1526406).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 32; pp. 32:1–32:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

questions, are studied in this paper. We give the first set of results on universal guarding, including combinatorial bounds and efficient algorithms to compute universal guard sets that achieve the upper bounds we prove. We focus on the case in which guards must be placed at a subset of the input set S and thus will be vertex guards for any polygonalization of S.

A strong motivation for our study is the problem of computing guard sets in the face of uncertainty. In our model, we require that the guards are *robust* with respect to different possible polygonalizations consistent with a given set of points (e.g., obtained by scanning an environment). Our Universal Guard Problem is, in a sense, an extreme version of the problem of guarding a set of possible polygonalizations that are consistent with a given set of sample points that are the polygon vertices: In the universal setting, we require that the guards are a rich enough set to achieve visibility coverage for *all* possible polygonalizations. Another variant studied here is the *k-universal* guarding problem in which the guards must perform visibility coverage for a set of k different polygonalizations of the input points. Further, in the full version of the paper [10], we study the case in which guards are required to be placed at non-convex hull points of S, or at points of a regular rectangular grid.

Related Work

In addition to the worst-case results for the AGP, related work includes algorithmic results for computing a minimum-cardinality guard set. The problem of computing an optimal guard set is known to be NP-hard [24], even in very basic settings such as guarding a 1.5D terrain [21]. Ghosh [13, 14] observed that greedy set cover yields an $O(\log n)$-approximation for guarding with the fewest vertices. Using techniques of Clarkson [5] and Brönnimann-Goodrich [3], $O(\log OPT)$-approximation algorithms were given, if guards are restricted to vertices or points of a discrete grid [7, 8, 15]. For the special case of *rectangle visibility* in rectilinear polygons, an exact optimization algorithm is known [27]. Recently, for vertex guards (or discrete guards on the boundary) in a simple polygon P, King and Kirkpatrick [20] obtained an $O(\log\log OPT)$-approximation, by building ϵ-nets of size $O((1/\epsilon) loglog(1/\epsilon))$ for the associated hitting set instances, and applying [3]. For the special case of guarding 1.5D terrains, local search yields a PTAS [22, 12]. Experiments based on heuristics for computing upper and lower bounds on guard numbers have been shown to perform very well in practice [1]. Methods of combinatorial optimization with insights and algorithms from computational geometry have been successfully combined for the Art Gallery Problem, leading to provably optimal guard sets for instances of significant size [2, 6, 23, 26, 9].

The notion of "universality" has been studied in other contexts in combinatorial optimization [18, 16], including the traveling salesman problem (TSP), Steiner trees, and set cover. For example, in the universal TSP, one desires a single "master" tour on all input points so that, for *any* subset S of the input points, the tour obtained by visiting S in the order specified by the master tour yields a tour that approximates an optimal tour on the subset.

Our Results

We introduce a family of universal coverage problems for the classic Art Gallery Problems. We provide a spectrum of lower and upper bounds for the required numbers of guards. See Table 2 and 3 for a detailed overview, and the following Section 2 for involved notation.

2 Preliminaries

For $n \in \mathbb{N}$, let $\mathcal{S}(n)$ be the set of all discrete point sets in the plane that have cardinality n. A single *shell* of a point set S is the subset of points of S on the boundary of the convex

Table 1 The universal guard numbers considered in this paper.

universal guard numbers	$\boldsymbol{u}(n)$	$\max_{S\in\mathcal{S}(n)} w(\mathcal{P}(S))$
m-shelled universal guard numbers	$\boldsymbol{s}(n,m)$	$\max_{S\in\mathcal{S}(n,m)} w(\mathcal{P}(S))$
interior universal guard numbers	$\boldsymbol{i}(n)$	$\max_{S\in\mathcal{S}(n)} \boldsymbol{i}(\mathcal{P}(S))$
k-universal guard numbers of simple polygons	$\boldsymbol{u}_k(n)$	$\max_{S\in\mathcal{S}(n)} \max_{\substack{A\subseteq\mathcal{P}(S)\\ s.t.\ \lvert A\rvert=k}} w(A)$
k-universal guard numbers of polygons w. holes	$\boldsymbol{h}_k(n)$	$\max_{S\in\mathcal{S}(n)} \max_{\substack{A\subseteq\mathcal{H}(S)\\ s.t.\ \lvert A\rvert=k}} w(A)$
grid universal guard numbers	$\boldsymbol{g}(n)$	$\max_{S\in\mathcal{S}_g(n)} w(\mathcal{P}(S))$

Table 2 Results for simple polygons. The approaches for the upper bounds for $\boldsymbol{u}(n)$ and $\boldsymbol{s}(n,m)$ also apply to polygons with holes, yielding the same upper bounds.

$m,n\in\mathbb{N}$	$\boldsymbol{u}(n)$	$\boldsymbol{s}(n,m)$	$\boldsymbol{g}(n)$	$\boldsymbol{i}(n)$
lower bounds	$\left(1-\Theta\left(\frac{1}{\sqrt{n}}\right)\right)n$	$\left(1-\frac{1}{2(m-1)}-\frac{8m}{n(m-1)}\right)n$	$\lfloor\frac{n}{2}\rfloor$	$n-\mathcal{O}(1)$
upper bounds	$\left(1-\Theta\left(\frac{1}{n}\right)\right)n$	$\left(1-\frac{1}{16n^{\left(1-\frac{1}{2m}\right)}}\right)n$	$\lfloor\frac{n}{2}\rfloor$	$n-\Omega(1)$

Table 3 Overview of our results for k-universal guard numbers of simple polygons and of polygons with holes. We give a new corresponding approach for the upper bounds of $\boldsymbol{h}_1(n), \boldsymbol{h}_2(n), \ldots$. We also consider the lower bounds for $\boldsymbol{u}_1(n), \boldsymbol{u}_2(n), \ldots$ as lower bounds for $\boldsymbol{h}_1(n), \boldsymbol{h}_2(n), \ldots$.

$n\in\mathbb{N}$	$\boldsymbol{u}_2(n)$	$\boldsymbol{u}_3(n)$	$\boldsymbol{u}_4(n)$	$\boldsymbol{u}_5(n)$	$\boldsymbol{u}_k(n)$ for $k\geq 6$	$\boldsymbol{h}_k(n)$ for $k\in\mathbb{N}$
lower bounds	$\lfloor\frac{3n}{8}\rfloor$	$\frac{4n}{9}$	$\frac{n}{2}-\mathcal{O}(\sqrt{n})$	$\frac{n}{2}-\mathcal{O}(\sqrt{n})$	$\frac{5n}{9}$	$\frac{5n}{9}$
upper bounds	$\frac{5n}{9}$	$\frac{19n}{27}$	$\frac{65n}{81}$	$\frac{211n}{243}$	$(1-(\frac{2}{3})^k)n$	$(1-(\frac{5}{8})^k)n$

hull of S. Recursively, for $k\geq 2$, a point set lies on k shells, if removing the points on its convex hull, leaves a set that lies on $k-1$ shells. We denote by $\mathcal{S}_g(n)\subset\mathcal{S}(n)$ and $\mathcal{S}(n,m)\subset\mathcal{S}(n)$ the set of all discrete point sets that form a rectangular $a\times b$-grid of n points for $a,b,a\cdot b=n\in\mathbb{N}$, and the set of all discrete point sets that lie on m shells for $m\in\mathbb{N}$, respectively.

For $S\in\mathcal{S}(n)$, let $\mathcal{P}(S)$ (resp., $\mathcal{H}(S)$) be the set of all simple polygons (resp., polygons with holes) whose vertex set equals S.

Let P be a polygon. We say a point $p\in P$ *sees* (w.r.t. P) another point $q\in P$ if $pq\subset P$; we then write $p\leftrightarrow_P q$. The *visible region* (w.r.t. P) of a point $g\in P$ is $V_P(g)=\{a\in P : g\leftrightarrow_P a\}$. A point set $G\subseteq S$ is a *guard set* for P if $\bigcup_{g\in G}V_P(g)=P$. Furthermore, we say that G is an *interior guard set for* P if G is a guard set for P and no $g\in G$ is a vertex of the convex hull of P.

For a set A of polygons we say that $G\subseteq S$ is a(n) (interior) guard set of A if G is a(n) *(interior) guard set* for each $P\in A$. We denote by $w(A)$ the minimum cardinality guard set for A and by $\boldsymbol{i}(A)$ the minimum cardinality interior guard set for A. Furthermore, for any given point set S we say that $G\subseteq S$ is a *guard set for* S if G is a guard set for $\mathcal{P}(S)$. For $k,m,n\in\mathbb{N}$, the guard numbers are listed in Table 1.

3 Bounds for Universal Guard Numbers

In the following, we provide different lower and upper bounds for the universal guard numbers. In particular, the provided bounds can be classified by the number of shells on which the points of the considered point set are located.

3.1 Lower Bounds for Universal Guard Numbers

In this section we give lower bounds for the universal guard numbers $\boldsymbol{u}(n)$ and $\boldsymbol{s}(n,m)$ for $n \in \mathbb{N}$ and $m \geq 2$. In particular, we provide lower bound constructions that can be described by the following approach: For any given $n \in \mathbb{N}$ and $m \geq 2$, we construct a point set $S_m \in \mathcal{S}(n)$ as follows. S_m is partitioned into pairwise disjoint subsets $B_1, \ldots, B_m$, such that $\bigcup_{i=1}^{m} B_i = S$. For $i \in \{1, ..., m\}$, each B_i lies on a circle C_i such that C_i is enclosed by C_{i+1} for $i \in \{1, ..., m-1\}$. Furthermore, $C_1, \ldots, C_m$ are concentric and have "sufficiently large" radii; see Sections 3.1.1, 3.1.2, and 3.1.3 for details. In particular, the radii depend on the approaches that are applied for the different cases $m = 2$, $m = 3$, and $m \geq 4$. We place four equidistant points on C_m. The remaining points are placed on $C_{m-1}, \ldots, C_1$.

Note that $\boldsymbol{s}(n,1) = 1$ holds, because for every convex point set $S \in \mathcal{S}(n)$, $\mathcal{P}(S)$ consists of only the boundary of the convex hull of S. Thus we start with the case of $m = 2$.

3.1.1 Lower Bounds for $\boldsymbol{s}(n, 2)$

We give an approach that provides a lower bound for $\boldsymbol{s}(n,2)$. In particular, for any $n \in \mathbb{N}$, we construct a point set $S_2 \in \mathcal{S}(n)$ having $n-4$ equally spaced points lie on circle C_1 and 4 equally spaced points on a larger concentric circle C_2, such that these 4 points form a square containing C_1; see Figure 5. In order to assure that the constructed subsets of S_2 and $S_3, S_4, \ldots$ (which are described later) are nonempty, we require $n \geq 32$ for the rest of Section 3.1.

Let v be a point from the square and let p, q be two consecutive points from the circle C_1, such that the segments vp and vq do not intersect the interior of the circle C_1; see Figure 1(a). We choose the side lengths of the square such that the cone c that is induced by p and q with apex at v contains at most $\frac{n}{8}$ points from C_1 for all choices of v, p, and q.

▶ **Lemma 1.** *Let G be a guard set of S_2. Then we have $|G| > \frac{n}{2} - 4$.*

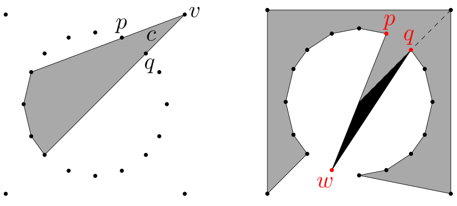
Proof. Suppose $|G| \leq \lfloor \frac{n-4}{2} \rfloor - 1$. This implies that there are two points $p, q \in S_m \setminus G$ such that p and q lie adjacent on C_1; see Figure 1(b). Let w_1, w_2, w_3, and w_4 be the four points from the square. At most two points $v_1, v_2 \in \{w_1, w_2, w_3, w_4\}$ span a cone, such that v_1p, v_1q, v_2p, v_2q do not intersect the interior of C_1. W.l.o.g., we assume that these two different cones c_1 and c_2 exist. c_1 and c_2 contain at most $\frac{n}{4}$ points from C. Thus, there is another point $w \in S_2 \setminus G$ such that $v \notin c_1 \cup c_2$. This implies that there is a polygon in which w is not seen by a guard from G; see Figure 1(b). This is a contradiction to the assumption that G is a guard set.

Thus we have $|G| > \lfloor \frac{n-4}{2} \rfloor - 1 \geq \frac{n-4}{2} - 2 = \frac{n}{2} - 4$. This concludes the proof. ◀

▶ **Corollary 2.** $\boldsymbol{s}(n,2) \geq \lfloor \frac{n}{2} \rfloor - 4$

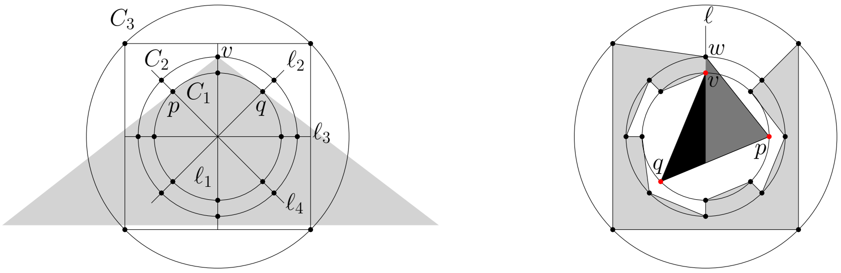
3.1.2 A First Lower Bound for $\boldsymbol{s}(n, 3)$

The high-level idea is to guarantee in the construction of S_3 that at most two points on C_1 are unguarded; see Figure 2 for the idea of the proof of contradiction. By constructing



(a) (b)

Figure 1 Lower-bound construction for $\boldsymbol{s}(n,2)$.



(a) Lower-bound construction for $\boldsymbol{s}(n,3)$. (b) An empty chamber $\triangle(w,p,q,v)$.

Figure 2 The lower-bound construction for $\boldsymbol{s}(n,3)$.

$S_3 = B_1 \cup B_2 \cup B_3$ such that $|B_1| = \lfloor \frac{n-4}{2} \rfloor$, $|B_2| = \lceil \frac{n-4}{2} \rceil$, and $|B_3| = 4$, we obtain $|G| \geq \frac{n}{2} - 5$ for any guard set G of S_3.

We consider the lower-bound construction S_m for $m-1 = 2$ and $n = (m-1)2^l + 4 = 3 \cdot 2^l + 4$ for any $l \geq 4$, i.e., for all $S_3 \in \mathcal{S}\left(2 \cdot 2^l + 4\right)$ for any $l \geq 2$. The argument can easily be extended to $n \in \mathbb{N}$.

The points of B_2 and B_3 are placed on C_2 and C_3, such that they lie on 2^{l-1} lines; see Figure 2(a). Let $v \in B_2$ be chosen arbitrarily and $p, q \in B_1$ such that p and q are the neighbors of the point from B_1 that corresponds to $v \in B_2$. We choose the radius of C_2 such that the cone that is induced by p and q and with apex at v contains all points from B_1; see the gray cone in Figure 2(a). Furthermore, we choose the radius of C_1 such that the square that is induced by the four points from B_1 contains all points from $B_1 \cup B_2$.

The key construction that we apply in the proofs of our lower bounds are *chambers*.

▶ **Definition 3.** Let S be an arbitrary discrete point set in the plane. Four points $p_1, p_2, p_3, p_4 \in S$ form a *chamber, denoted* $\triangle(p_1, p_2, p_3, p_4)$, if (1) p_1 and p_2 lie on different sides of the line p_3p_4 and (2) p_3 and p_4 lie on the same side of the line p_1p_2, and (3) there is no point from S that lies inside the polygon that is bounded by the polygonal chain $\langle p_1, p_2, p_3, p_4 \rangle$.

Let $G \subseteq S$. We say that $\triangle(p_1, p_2, p_3, p_4)$ is *empty* (w.r.t. G) if $p_2, p_3, p_4 \notin G$. Let $P \in \mathcal{P}(S)$. We say that $\triangle(p_1, p_2, p_3, p_4)$ *is part of* P if $p_1p_2, p_2p_3, p_3p_4 \subset \partial P$.

Our proofs are based on the following simple observation.

▶ **Observation 4.** *Let G be a guard set for a polygon P. There is no empty chamber that is part of P.*

Based on Observation 4 we prove the following lemma, which we then apply to the construction above to obtain our lower bound for $\boldsymbol{s}(n,m)$.

▶ **Lemma 5.** *Let G be a guard set for $\mathcal{P}(S_3)$. Then we have $|B_1 \setminus G| \leq 2$.*

Proof. Suppose there are three points $v, q, p \in B_1 \setminus G$. W.l.o.g., we assume that q and p lie on different sides w.r.t. the line ℓ that corresponds to the placement of v; see Figure 2(b). Furthermore, we denote the point from B_2 that lies above v by w. By construction it follows that w, p, q, and v form an empty chamber $\triangle(w, p, q, v)$. Furthermore, we construct a polygon $P \in \mathcal{P}(S_3)$ such that $\triangle(w, p, q, v)$ is part of P; see Figure 2(b). By Observation 4 it follows that G is not a guard set for P, a contradiction. This concludes the proof. ◀

There is a corresponding construction for all other values $n \in \mathbb{N}$. In particular, we place four points equidistant on C_3, $\lceil \frac{n-4}{2} \rceil$ equidistant points on C_2, and $\lfloor \frac{n-4}{2} \rfloor$ points on C_1, such that each point from C_1 lies below a point from C_2. The same argument as above applies to the resulting construction of a point set. The constructions of S_m can be modified so that no three points lie on the same line, by a slight perturbation. Thus, S_3 can be assumed to be in general position. We obtain the following corollary.

▶ **Corollary 6.** $\boldsymbol{s}(n, 3) \geq \frac{n}{2} - 5$.

Proof. Lemma 5 implies that in the construction S_3 at least $\lfloor \frac{n-4}{2} \rfloor - 2$ points from B_1 are guarded. Let G be an arbitrarily chosen guard set for $\mathcal{P}(S_3)$. Thus we obtain $|G| \geq \lfloor \frac{n-4}{2} \rfloor - 2 \geq \frac{n-4}{2} - 3 = \frac{n}{2} - 5$. ◀

In the following section we generalize the above approach from the case of three shells to the case of m shells and combine that argument with the approach that we applied for the case of $m = 2$. This also leads to the improved lower bound $\boldsymbol{u}_3(n) \geq (\frac{3}{4} - \mathcal{O}(\frac{1}{n}))n$.

3.1.3 (Improved) Lower Bounds for $\boldsymbol{u}(n)$ and $\boldsymbol{s}(n, m)$ for $m \geq 3$

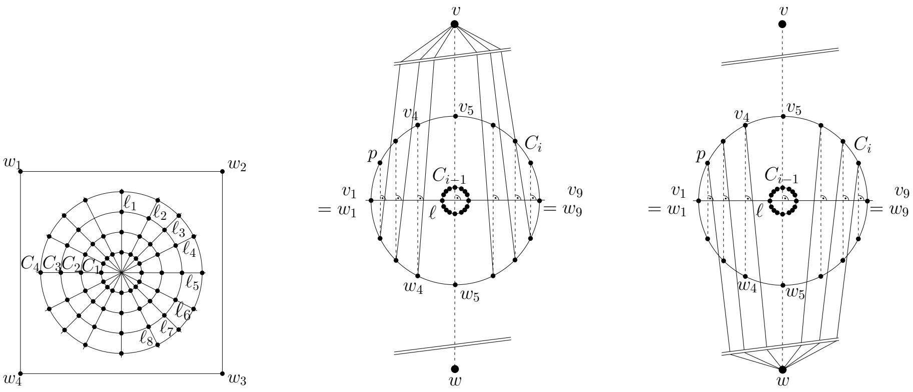
In this section we give general constructions $S_3, S_4, \ldots$ of the point sets that yield our lower bounds for $\boldsymbol{s}(n, m)$ for $m \geq 3$. The main difference in the construction of S_m for $m \geq 3$, compared to the previous section, is the choice of the radii of $C_1, ..., C_m$. Similar as in the previous section, we guarantee that on each circle $C_3, C_4, \ldots$ at most constant many points are unguarded. Roughly speaking, the general idea is to choose five arbitrary points q_1, q_2, q_3, q_4, q_5 on C_i for $i \in \{3, 4, \ldots\}$. There are three points $u_1, u_2, u_3 \in \{q_1, q_2, q_3, q_4, q_5\}$, such that the triangle induced by u_1, u_2, u_3 does not contain the common mid point of $C_1, C_2, \ldots$. By choosing the radius of C_{i+1} sufficiently large, we obtain that there is a chamber $\triangle(u_1, u_2, u_3, p)$, where p is a point on C_{i+1}. This implies that $\triangle(u_1, u_2, u_3, p)$ is empty if q_1, q_2, q_3, q_4, q_5 are unguarded. Thus, at most four points on C_i are allowed to be unguarded; see Corollary 9.

Finally, we show how the arguments for S_m yield lower bounds for $\boldsymbol{s}(n, m)$ and $\boldsymbol{u}(n)$.

Similar to the approach of the previous section, the constructed point sets $S_3, S_4, \ldots$ can be modified to be in general position.

The Construction of S_m for $m \geq 3$: We construct S_m such that $|B_1| = \cdots = |B_{m-1}| = 2^l$, $|B_m| = 4$, and hence $n = (m-1)2^l + 4$ for $l \geq 4$. In particular, similar as for the construction of S_3 from the previous section, we place the points of $B_1, \ldots, B_{m-1}$ equidistant on the circles $C_1, \ldots, C_{m-1}$, such that the points lie on 2^{l-1} lines $\ell_1, \ldots, \ell_{2^{l-1}}$; see Figure 3(a).

In order to apply an argument that makes use of chambers, we need the following notation of points on a circle C_i. Let $n' := 2^l$. Let $v_1, ..., v_{1+n'/2}$ be the points on C_i to one side or on $\ell \in \{\ell_1, ..., \ell_{n'/2}\}$. Let $w_1, ..., w_{1+n'/2}$ be their reflection across ℓ; see Figure 3(b)+(c). Let $v_1, ..., v_{1+n'/2}$ and $w_1, ..., w_{1+n'/2}$ be the points that lie not below and not above ℓ; see Figure 3(b)+(c). Let $v, w \in C_{i+1}$ be the points that correspond to $v_{1+n'/4}$ and $w_{1+n'/4}$.



(a) Construction of the circles $C_1, ..., C_m$.

(b) Segments between v and vertices from the opposite side of C_i.

(c) Segments between v and vertices from the opposite side of C_i.

Figure 3 Construction of S_m for $n = 68$. For a simplified illustration we changed the ratios of the circles' radii and we shortened the lines adjacent to v.

For $i \in \{1, \ldots, m-1\}$, we choose the radius of C_{i+1} compared to the radius of C_i sufficiently large, such that the following conditions are fulfilled; see Figure 3(b)+(c):

- vw_j intersects $v_j v_{j+1}$ in its interior for all $j \in \{1, ..., n/4+1\}$,
- vw_j intersects $v_{j-1} v_j$ in its interior for all $j \in \{n/4+2, ..., n/2+1\}$,
- wv_j intersects the segment $w_j w_{j+1}$ in its interior for all $j \in \{1, ..., n/4+1\}$, and
- wv_j intersects the segment $w_{j-1} w_j$ in its interior for all $j \in \{n/4+2, ..., n/2+1\}$.

Finally, we place the four points $w_1, w_2, w_3, w_4 \in B_m$ such that all circles lie in the convex hull of w_1, w_2, w_3, and w_4; see Figure 3(a).

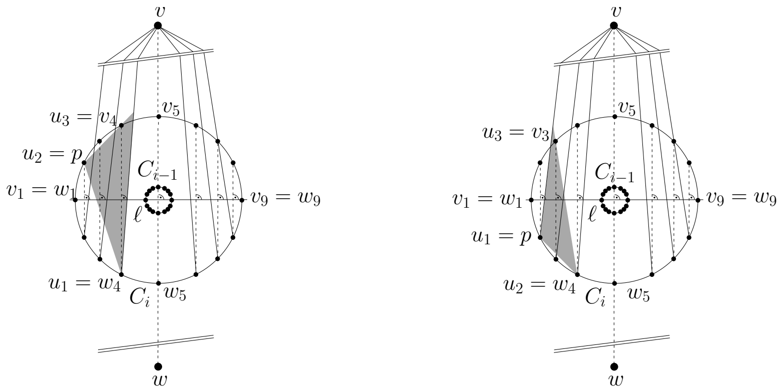
The Analysis of S_m for $m \geq 3$: First we show that we can choose three points u_1, u_2, u_3 from five arbitrarily chosen points from C_i, such that there is another point $u \in C_{i+1}$ with $\triangle(u, u_1, u_2, u_3)$ being a chamber; see Lemma 7. Next, we construct a polygon $P \in \mathcal{P}(S_m)$, such that $\triangle(u, u_1, u_2, u_3)$ is a part of P; see Lemma 8. Finally, by combining Lemma 7 and Lemma 8 we establish that on each C_i, at most four points are allowed to be unguarded; see Corollary 9. This leads to several lower bounds for $\boldsymbol{s}(n, m)$ and $\boldsymbol{u}(n)$.

▶ **Lemma 7.** *Let $q_1, q_2, q_3, q_4, q_5 \in A_i$ be chosen arbitrarily. There are three points $u_1, u_2, u_3 \in \{q_1, q_2, q_3, q_4, q_5\}$ and a point $u \in A_{i+1}$, such that $\triangle(u, u_1, u_2, u_3)$ is a chamber.*

Proof. We choose u_1, u_2, u_3 from $\{q_1, q_2, q_3, q_4, q_5\}$, such that u_1, u_2, u_3 lie in the same half of C_i, i.e., such that the midpoint of C_i does not lie inside the triangle t that is induced by u_1, u_2, u_3; see Figure 4. W.l.o.g., we assume that u_2 lies between u_1 and u_3. Otherwise, we rename the points appropriately.

We distinguish two cases. (C1) The number of points between u_1 and u_3 is odd and (C2) the number of points between u_1 and u_3 is even. For (C1) and (C2) we use different chambers for achieving the required contradiction; see Figure 4. A detailed analysis can be found in the full paper [10]. ◀

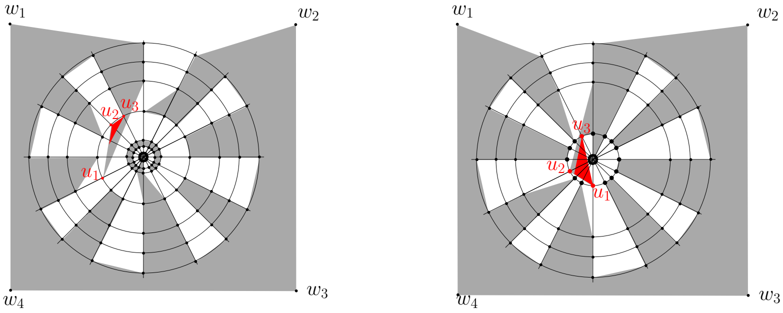
▶ **Lemma 8.** *There is a polygon $P \in \mathcal{P}(S_m)$ such that $\triangle(u, u_1, u_2, u_3)$ is part of P.*



(a) Chambers for (C1).
of points between u_1 and u_2 is odd.

(b) Chambers for (C2).
of points between u_1 and u_2 is even.

Figure 4 Configuration of Lemma 7: three points from C_i in the same half of C_i imply a chamber.



(a) The case in which the number of points between u_1 and u_2 is odd.

(b) The case in which the number of points between u_1 and u_2 is even.

Figure 5 Construction of $\mathcal{P}$ for $k = 6$ and $n = 16$. For a simplified illustration we changed the ratios of the circles' radii (otherwise the figure would become too large).

Proof. We construct P for the cases (C1) and (C2) separately; see Figure 5. In both cases we walk upwards on the line $\ell \in \{\ell_1, \dots, \ell_{n'/2}\}$ until we reach C_1. Next we orbit C_i in a zig-zag approach and finally connect all points from $C_{i-1}, \dots, C_1$ in a similar manner; see Figure 5. ◀

The combination of Lemma 7 and Lemma 8 implies the following corollary.

▶ **Corollary 9.** *Let $G \subset S_m$ be a guard set of $\mathcal{P}(S_m)$. Then $|B_i \backslash G| \le 4$, for $i \in \{1, \dots, m-2\}$.*

Lower bounds for $\boldsymbol{s}(n, m)$ and $\boldsymbol{u}(n)$ which are implied by Corollary 9: We combine the approach for $\boldsymbol{s}(n, 2)$ with Corollary 9, which yields the following lower bound for $\boldsymbol{s}(n, m)$ for $m \ge 3$.

▶ **Corollary 10.** *Let $m \ge 3$ and $n' = 2^l$ with $l \ge 4$. Furthermore, let $G \subset S_m$ be a guard set of S_m. Then we have $|G| \ge \left(1 - \frac{1}{2(m-1)} + \frac{8m}{n(m-1)}\right)|S_m|$.*

Proof. By Corollary 9 it follows that $(m-2)(n'-4)$ points from $B_1 \cup \cdots \cup B_{m-2}$ are guarded. Furthermore, by applying the approach of Lemma 1 to B_{m-1} and B_m yields that at least

$\frac{n'}{2} - 4$ points from $B_{m-1} \cup B_m$ are guarded. Thus we obtain $|G| \geq (m-2)(n'-4) + \frac{n'}{2} - 4$ which is upper-bounded by $|S_m|\left(1 - \frac{1}{2(m-1)} - \frac{8m}{|S_m|(m-1)}\right)$ because $n' = \frac{|S_m|-4}{m-1}$. ◀

▶ **Theorem 11.** *$\boldsymbol{s}(n,m) \geq n\left(1 - \frac{1}{2(m-1)} + \frac{8m}{n(m-1)}\right)$ for $m \geq 3$.*

By choosing m appropriately, we obtain the following lower bound:

▶ **Lemma 12.** *For any $c < 1$ and any guard set G for S_m there is an $m \in \mathbb{N}$ with $|G| > c|S_m|$.*

Proof. The approach is to choose $m := \lceil \frac{2n'}{n'-4-cn'} \rceil$, which will imply $|G| > c|S_m|$.

Suppose $|G| \leq c|S_m|$. Corollary 9 implies that at most four points on each circle $C_i \in \{C_1, ..., C_k\}$ are unguarded. This leads to a contradiction as follows. We have $|S_m| = 4+(m-1)n'$. On $C_1, ..., C_{m-2}$ there are at most four vertices that are unguarded. W.l.o.g., we assume that w_1, w_2, w_3, w_4, and all points on C_m are unguarded. Thus, $|G| \geq (m-2)(n'-4)$. By assumption we know $|G| \leq c(4+(m-1)n')$. By applying $m = \lceil \frac{2n'}{n'-4-cn'} \rceil$, we obtain a contradiction as follows: $(m-2)(n'-4) \leq c(4+(m-1)n')$ implies that $8 \leq 4$, since $m = \lceil \frac{2n'}{n'-4-cn'} \rceil$. ◀

By choosing c appropriately, Lemma 12 leads to our general upper bound for $\boldsymbol{u}(n)$.

▶ **Theorem 13.** *There is an $m \in \mathbb{N}$ such that $|G| > (1 - \frac{10}{\sqrt{|S_m|}})|S_m|$ holds for any guard set G for $\mathcal{P}(S_m)$.*

Proof. Choose $c := (1 - \frac{5}{n'})$ in the approach of Lemma 12. This implies that at least $(1 - \frac{5}{n'})|S_m|$ points have to be guarded. Furthermore, we have $|S_m| = 4 + (m-1)n'$ and $m = \lceil \frac{2n'}{n'-4-cn'} \rceil$. This implies $m \leq \frac{2n'}{n'-4-\left(1-\frac{5}{n'}\right)n'} + 1 = 2n' + 1$. Furthermore, $|S_m| \leq 4 + 2(n')^2$ implies $\sqrt{|S_m|/2}/ - 1 \leq n$. Finally, applying Lemma 12 yields $|G| > \left(1 - \frac{5\sqrt{2}}{\sqrt{|S_m|-1}}\right)|S_m| > \left(1 - \frac{10}{\sqrt{|S_m|}}\right)|S_m|$. ◀

▶ **Theorem 14.** *$\boldsymbol{u}(n) \geq \left(1 - \frac{10}{\sqrt{n}}\right) n$.*

3.2 Upper Bounds for Universal Guard Numbers

In the following we give an approach to computing a non-trivial guard set of a given point set. The number of the computed guards depends on the number m of shells of the considered point set S. This approach yields upper bounds for $\boldsymbol{s}(n,m)$ for $m \geq 2$.

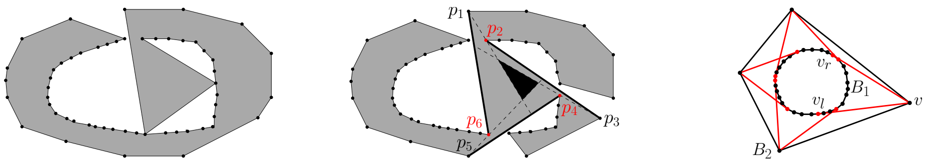
For the case of $m = 1$, a naïve approach is simply to select one arbitrarily chosen guard from S. In that case, $\mathcal{P}(S)$ just consists of the polygon that corresponds to the boundary of the convex hull of S and an arbitrarily chosen point from S sees all points from all polygons of $\mathcal{P}(S)$.

In the following, we first give an approach for the case of $m = 2$. Then, we generalize that approach to the case of $m \geq 3$.

3.2.1 Upper Bounds for $\boldsymbol{s}(n,2)$

First we describe the approach, followed by showing that the computed point set G is a guard set for the considered point set. This leads to upper bounds for $|G|$ which imply the required upper bounds for $\boldsymbol{s}(n,m)$.

The high-level idea is to avoid areas that are unguarded by structures similar to chambers. In particular, in the case of $m = 2$, a chamber cannot be part of a simple polygon; otherwise,



(a) chambers are not part of a simple polygon if $m = 2$.

(b) For $m = 2$ a similar structure may be part of a simple polygon.

(c) Avoiding chambers and similar structures by tangent points.

Figure 6 Possible chambers in case of two shells and how we avoid them.

the boundary of P meets points at least twice (Figure 6(a)). However, there is another structure that has an effect similar to that of chambers and that also may cause unguarded areas; see Figure 6(b). In the example of Figure 6(b), our approach guarantees that p_2 or p_6, p_2 or p_4, and p_4 or p_6 is guarded. More generally, for p_1, p_3, and p_5 we guarantee that the unguarded points lie on one side w.r.t. the tangent points; see Figure 6(c).

In particular, let B_1 be the points on the inner shell and B_2 be the points on the outer shell of the input point set S. If $|B_2| \geq \sqrt{|B_1|}/2$ we set $G = B_1$. Otherwise, we choose all points from B_2 and every second point from B_1. Furthermore, we compute for each $v \in B_2$ the two tangent points v_l and v_r to B_1 (see Figure 6(c)) and insert v_l and v_r into G. Let $\langle v_1, \ldots, v_k \rangle \subset B_1$ be a sequence of maximal length that does not contain any tangent point as previously computed. We insert all remaining points from $B_1 \setminus \{v_1, \ldots, v_k\}$ that were not already inserted in G.

▶ **Theorem 15.** *For each point set S that lies on two convex hulls, we can compute in $\mathcal{O}(|S| \log |S|)$ time a guard set G with $|G| \leq (1 - \frac{1}{\sqrt{8|S|}})|S|$.*

▶ **Corollary 16.** $s\,(n, 2) \leq (1 - \frac{1}{\sqrt{8n}})n$

3.2.2 Upper Bounds for $s\,(n, m)$ for $m \geq 3$

In this section we generalize the above approach to the case of $m \geq 3$.

Let $B_1, \ldots, B_m$ be the pairwise disjoint subsets of S that lie on the m shells of S. The high-level idea of the approach is the following. If B_m is "large enough" (larger than a value λ), we set $G = \bigcup_{\ell=1}^{m-1} B_\ell$. Otherwise, we carefully choose one subset B_j for $j \in \{1, \ldots, m\}$ and select partially its points as unguarded. All the remaining points are selected for G. In particular, we set $\bigcup_{\ell \in \{1,\ldots,m\} \setminus \{j\}} B_\ell \subset G$. Then, we compute the tangent points on B_j for all points from $\bigcup_{\ell=j+1}^{m} B_\ell$. Finally, we apply the same subroutine as in the case $m = 2$.

We choose $j := \arg\max_{\ell \in \{1,\ldots,m-1\}} \left(\frac{n_\ell}{2\sum_{i=\ell+1}^{m} n_i} - 1 \right)$ and $\lambda := \frac{n_j}{2\sum_{i=j+1}^{m} n_i} - 1$. We refer to the full paper for the detailed steps of the approach.

By applying a similar argument as for the case of $m = 2$, we can show that the computed point set $G \subseteq S$ is a guard set for $\mathcal{P}\,(S)$. For details, see the full paper.

▶ **Theorem 17.** *For any point set S that lies on m convex hulls we can compute in $\mathcal{O}(n \log n)$ time a guard set G with $|G| \leq \left(1 - \frac{1}{16|S|^{\left(1 - \frac{1}{2m}\right)}}\right)|S|$*

This leads to our generalized upper bound for $s\,(n, m)$ for $m \geq 3$.

▶ **Corollary 18.** $s\,(n, m) \leq \left(1 - \frac{1}{16n^{\left(1 - \frac{1}{2m}\right)}}\right)n.$

4 Bounds for the k-Universal Guard Numbers

In the following we state several lower and upper bounds for various k-universal guard numbers; proof details are in the full paper.

4.1 Lower bounds for $u_k(n)$

▶ **Theorem 19.** $\boldsymbol{u}_2(n) \geq \lfloor \frac{3n}{8} \rfloor$

▶ **Theorem 20.** $\boldsymbol{u}_3(n) \geq \lfloor \frac{4n}{9} \rfloor$.

▶ **Theorem 21.** $\boldsymbol{u}_5(n) \geq \boldsymbol{u}_4(n) \geq \frac{n}{2} - 8\sqrt{n} - 23$.

▶ **Theorem 22.** $\boldsymbol{u}_k(n) \geq \lfloor \frac{5n}{9} \rfloor$ *for* $k \geq 6$.

4.2 Upper Bounds for k-Universal Guard Numbers

We give non-trivial upper bounds for $\boldsymbol{u}_k(n)$ and $\boldsymbol{h}_k(n)$, for all values $n, k \in \mathbb{N}$. In particular, we provide algorithms that efficiently compute guard sets for $\mathcal{P}(S)$ and $\mathcal{H}(S)$ for any given $S \in \mathcal{S}(n)$ and analyze the computed guard sets.

▶ **Theorem 23.** $\boldsymbol{u}_k(n) \leq \left(1 - \left(\frac{2}{3}\right)^k\right)$.

Hoffmann et al. [17] showed $\boldsymbol{h}_1(n) \leq \lfloor \frac{3n}{8} \rfloor$. Our approach implies for the traditional guard number $\boldsymbol{h}_1(n) \leq \lfloor \frac{n}{2} \rfloor$.

The following theorem shows that we can combine our approach with the method from [17].

▶ **Theorem 24.** $\boldsymbol{h}_k(n) \leq \left(1 - \left(\frac{5}{8}\right)^k\right) n$.

5 Other Variants

Due to limited space, we state two variants of the Universal Art Gallery Problem but defer the technical details to the full paper.

5.1 Interior Guards

In the Interior Universal Guards Problem (UGPI) we allow guards to be placed only at points of S that are not convex hull vertices of S. For this case, we obtain an asymptotically tight bound on the number of universal guards:

▶ **Theorem 25.** $\boldsymbol{i}(n) = n - \Theta(1)$

5.2 Full Grid Sets

A natural special case arises when considering universal guards for a full set of $n = a \times b$ grid points on an integer lattice. We are also able in this case to achieve a tight worst-case bound:

▶ **Theorem 26.** $\boldsymbol{g}(n) = \lfloor \frac{n}{2} \rfloor$.

6 Conclusion

There are many open problems that are interesting challenges for future work. In particular, can the upper bound approaches for $\boldsymbol{u}_k(n)$ and $\boldsymbol{h}_k(n)$ be improved by making use of the number of shells? Can the general approach of Theorem 23 be improved? What about lower bounds for k-UGP for $k \geq 7$?

The quest for better bounds is also closely related to other combinatorial challenges. Is an instance of the 2-UGP 5-colorable? If so, our results give a first trivial upper bound of $\frac{3}{5}n$ for the 2-UGP, which would be of independent interest. Is the bound of $\frac{1}{2}n$ for the intersection-free k-UGP tight? Further questions consider the setting in which each vertex v has a bounded candidate set of vertices that may be adjacent to v. Other variants arise when the ratio of the lengths of the edges of the considered polygons is upper- and lower-bounded by given constants. It may also be interesting to explore possible relations between universal guard problems and universal graphs.

References

1 Yoav Amit, Joseph S. B. Mitchell, and Eli Packer. Locating guards for visibility coverage of polygons. *Int. J. Comp. Geom. Appl.*, 20(5):601–630, October 2010.

2 Dorit Borrmann, Pedro J. de Rezende, Cid C. de Souza, Sándor P. Fekete, Stephan Friedrichs, Alexander Kröller, Andreas Nüchter, Christiane Schmidt, and Davi C. Tozoni. Point guards and point clouds: Solving general art gallery problems. In *Symp. Comp. Geom. (SoCG'13)*, pages 347–348, 2013.

3 H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14:263–279, 1995.

4 V. Chvátal. A combinatorial theorem in plane geometry. *J. Comb. Th. B*, 18:39–41, 1975.

5 Kenneth L Clarkson. Algorithms for polytope covering and approximation. In *Algorithms and Data Structures*, pages 246–252. Springer, 1993.

6 Marcelo C. Couto, Pedro J. de Rezende, and Cid C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *Int. Transact. Op. Res.*, 18:425–448, 2011.

7 Alon Efrat and Sariel Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

8 Alon Efrat, Sariel Har-Peled, and Joseph S. B. Mitchell. Approximation algorithms for location problems in sensor networks. In *2nd Int. Conf. Broadband Networks*, pages 767–776, 2005.

9 Sándor P. Fekete, Stephan Friedrichs, Alexander Kröller, and Christiane Schmidt. Facets for art gallery problems. *Algorithmica*, 73:411–440, 2015.

10 Sándor P. Fekete, Qian Li, Joseph S. B. Mitchell, and Christian Scheffer. Universal guard problems. *CoRR*, abs/1611.08315, 2016. URL: `http://arxiv.org/abs/1611.08315`.

11 S. Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Th. B*, 24:374, 1978.

12 Stephan Friedrichs, Michael Hemmer, and Christiane Schmidt. A PTAS for the continuous 1.5d terrain guarding problem. *CoRR*, abs/1405.6564, 2014.

13 S. K. Ghosh. Approximation algorithms for art gallery problems. In *Proc. Canadian Inform. Process. Soc. Congress*, 1987.

14 Subir Kumar Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, 158(6):718–722, 2010.

15 Hector Gonzalez-Banos and Jean-Claude Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, 2001.

16 Mohammad T Hajiaghayi, Robert Kleinberg, and Tom Leighton. Improved lower and upper bounds for universal TSP in planar metrics. In *Proc. Symp. Disc. Alg. (SODA)*, pages 649–658, 2006.

17 F. Hoffmann, M. Kaufmann, and K. Kriegel. The art gallery theorem for polygons with holes. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 39–48, 1991.

18 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *Symp. Theory Comp. (STOC)*, pages 386–395, 2005.

19 J. Mark Keil. Polygon decomposition. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier, Amsterdam, 2000.

20 James King and David Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete & Computational Geometry*, 46(2):252–269, 2011.

21 James King and Erik Krohn. Terrain guarding is NP-hard. *SIAM J. Comp.*, 40(5):1316–1339, 2011.

22 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi Varadarajan. Guarding terrains via local search. *Journal of Computational Geometry*, 5(1):168–178, 2014.

23 Alexander Kröller, Tobias Baumgartner, Sándor P. Fekete, and Christiane Schmidt. Exact solutions and bounds for general art gallery problems. *ACM Journal of Experimental Algorithmics*, 17(1), 2012. doi:10.1145/2133803.2184449.

24 J. O'Rourke. *Art Gallery Theorems and Algorithms.* The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

25 T. C. Shermer. Recent results in art galleries. *Proc. IEEE*, 80(9):1384–1399, September 1992.

26 Davi C. Tozoni, Pedro J. de Rezende, and Cid C. de Souza. The quest for optimal solutions for the art gallery problem: a practical iterative algorithm. *Optimization Online*, January 2013.

27 C. Worman and M. Keil. Polygon decomposition and the orthogonal Art Gallery Problem. *Int. J. Comp. Geom. Appl.*, 17(2):105–138, 2007.

Fast Approximation Algorithms for the Generalized Survivable Network Design Problem

Andreas Emil Feldmann*[1], Jochen Könemann[2], Kanstantsin Pashkovich[3], and Laura Sanità[4]

1 **Dept. of Applied Mathematics (KAM), Charles University, Prague, Czech Republic**
`feldmann.a.e@gmail.com`

2 **Dept. of Combinatorics and Optimization, University of Waterloo, Canada**
`jochen@uwaterloo.ca`

3 **Dept. of Combinatorics and Optimization, University of Waterloo, Canada**
`kpashkovich@uwaterloo.ca`

4 **Dept. of Combinatorics and Optimization, University of Waterloo, Canada**
`laura.sanita@uwaterloo.ca`

Abstract

In a standard f*-connectivity* network design problem, we are given an undirected graph $G = (V, E)$, a *cut-requirement function* $f : 2^V \to \mathbb{N}$, and non-negative costs $c(e)$ for all $e \in E$. We are then asked to find a minimum-cost vector $x \in \mathbb{N}^E$ such that $x(\delta(S)) \geq f(S)$ for all $S \subseteq V$. We focus on the class of such problems where f is a *proper* function. This encodes many well-studied NP-hard problems such as the *generalized survivable network design* problem.

In this paper we present the first *strongly polynomial time* FPTAS for solving the LP relaxation of the standard IP formulation of the f-connectivity problem with general proper functions f. Implementing Jain's algorithm, this yields a strongly polynomial time $(2 + \epsilon)$-approximation for the generalized survivable network design problem (where we consider *rounding up* of rationals an arithmetic operation).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization

Keywords and phrases strongly polynomial runtime, generalized survivable network design, primal-dual method

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.33

1 Introduction

The input to a typical *network design* problem consists of a directed or undirected graph $G = (V, E)$, non-negative unit-capacity *installation costs* $c(e)$ for all $e \in E$, and a collection of *connectivity requirements* among the vertices in V. The goal is then to find a minimum-cost capacity installation in G that satisfies the connectivity requirements. The above abstract problem class captures many practically relevant optimization problems, many of which are NP-hard. Therefore, maybe not surprisingly, there has been a tremendous amount of research in the area of approximation algorithms for network design problems throughout

[*] Supported by project CE-ITI (GAČR no. P202/12/G061) of the Czech Science Foundation.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 33; pp. 33:1–33:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the last four decades (e.g., see [17, 18]).

$$\begin{aligned} \min \ & \sum_{e \in E} c(e)x(e) && \text{(IP)} \\ \text{s.t.} \ & x(\delta(S)) \geq f(S) \quad \forall S \subset V \\ & x \geq \nvdash, x \text{ integer.} \end{aligned}$$

Connectivity requirements can be modelled in many ways, but we will adopt the *f-connectivity* viewpoint in this paper. Here, one is given a *cut-requirement function* $f : 2^V \to \mathbb{N}$, and one wants to find a minimum-cost non-negative integer vector x such that for each $S \subseteq V$, the sum of variables $x(e)$ for edges e crossing the cut S is at least $f(S)$. In other words, we are interested in problems that can be encoded by integer program (IP); here $\delta(S)$ denotes the set of edges incident to a vertex in S and a vertex outside S, and $x(\delta(S)) := \sum_{e \in \delta(S)} x(e)$.

Restricting even further, we will henceforth only be concerned with instances of (IP) where f is *proper*, that is, f satisfies the three properties of *maximality* (i.e., $f(A \cup B) \leq \max\{f(A), f(B)\}$ for all disjoint sets $A, B \subseteq V$), *symmetry* (i.e., $f(S) = f(V \setminus S)$ for all $S \subseteq V$), and $f(V) = 0$. Program (IP) with proper cut-requirement function f captures (among others) the special case where the goal is to find a minimum-cost network that has $r(u, v)$ edge-disjoint paths connecting any pair u, v of vertices (for given non-negative integer parameters r). The implicit cut-requirement function in this case is then given by $f(S) := \max_{u \in S, v \in V \setminus S} r(u, v)$ for all $S \subseteq V$.

Based on the *primal-dual* method, Goemans and Williamson [12] first gave a $2\mathcal{H}(f_{\max})$-approximation algorithm for (IP) with proper cut-requirement functions where one is allowed to pick edges multiple times in the solution. Goemans et al. [11] later obtained the same performance ratio for the setting where multiple copies of edges are not allowed. More recently, in a breakthrough result, Jain [19] obtained a 2-approximation for the more general class of *skew-supermodular* cut-requirement functions based on *iterative rounding.*

$$\begin{aligned} \min \ & \sum_{e \in E} c(e)x(e) && (\text{LP}_1) \\ \text{s.t.} \ & x(\delta(S)) \geq f(S) - z(\delta(S)) \quad \forall S \subseteq V \\ & x(e) = 0 \quad \forall e \in I \\ & x \geq \nvdash \end{aligned}$$

Jain's algorithm iteratively fixes the value of a subset of variables in (IP). To aid this, he first defines a slightly more general version of the IP, where the value of certain variables is fixed. Specifically, for a set $I \subseteq E$ of edges, assume that the value of variable $x(e)$ is fixed to $z(e) \in \mathbb{N}$. The LP relaxation of the IP for the corresponding residual problem is given in (LP_1). Jain's key observation was that the extreme points of the feasible region of (LP_1) are *sparse*, and have at least one variable with value at least 1/2.

Capitalizing on this insight, his algorithm then iteratively solves $O(|V|)$ instantiations of (LP_1) while intermittently rounding up the values of large variables in the computed solutions. In order to solve (LP_1) one needs to employ the Ellipsoid method [16] together with a polynomial-time seperation oracle for the LP's constraints (see [7]).

Our work is motivated by *Open Problem 4* in Williamson and Shmoys' recent book [23] where the authors point out that solving (LP_1) for general (proper) functions f may be computationally quite demanding despite the fact that it can be done efficiently in a theoretical sense. The authors leave as an open problem the design of a *primal-dual* 2-approximation for the survivable network design problem. Our main result is a replacement of the Ellipsoid-based exact LP-solver calls in Jain's algorithm by approximate ones that are based on the (in a sense) primal-dual *multiplicative-weights* method of [10]. We realize that the likely intended meaning of *primal-dual* in Williamson and Shmoys' open problem statement is the *primal-dual method for approximation algorithms* (as in [12]). However we believe that the contribution made in this paper is in line with the motivation given for Open Problem 4 in [23]: we substantially speed up LP computations in Jain's algorithm at the expense of an inconsequential loss in performance guarantee of the algorithm.

▶ **Theorem 1.** *For any $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm for* (LP$_1$) *that runs in* strongly polynomial time[1] *independently of the values of c and f.*

In contrast to our result, Jain [19] observes that the relaxations of (IP) and (LP$_1$) are of a *combinatorial* nature, and hence can be solved in strongly polynomial-time via Tardos' algorithm [22] *whenever* their number of variables and constraints are polynomially bounded (in the problem dimension). For example, (IP) and (LP$_1$) have an equivalent compact representation when $f(S) = \max_{u \in S, v \notin S} r(u,v)$ for all $S \subseteq V$. We also note that one can argue that the Ellipsoid method applied to (LP$_1$) and the linear relaxation of (IP) terminates in a strongly polynomial number of steps *whenever* function $f(S)$ is polynomially bounded (in the problem dimension), for all $S \subseteq V$ as this implies small encoding-length of vertices of the feasible region of (IP) and (LP$_1$).

To achieve the result in Theorem 1, we rely on the *multiplicative weights method* of Garg and Könemann [10] (henceforth referred to as GK). This is a natural idea as (LP$_1$) belongs to the class of *positive covering LPs.* As such, [10] applies to the LP dual of (LP$_1$). The algorithm can therefore be used to compute an approximate pair of primal and dual solutions in strongly-polynomial time as long as we are able to provide it with a strongly-polynomial time (approximation) algorithm for the so called *shortest row problem.* For (LP$_1$) this boils down to computing

$$\min_{\substack{f(S) - z(\delta(S)) \geq 1 \\ S \subseteq V}} \frac{x(\delta(S))}{f(S) - z(\delta(S))},$$

for given $x \in \mathbb{R}^E_+$ and $z \in \mathbb{Z}^E_+$, i.e., finding a corresponding set S. The above shortest row problem is solved directly by Gabow et al.'s strongly-polynomial time separation oracle for the constraints of (LP$_1$) (see [7]) in the case where $I = \emptyset$, and hence $z = \nvdash$. Once $I \neq \emptyset$, Gabow et al.'s algorithm can not be used directly to give a strongly polynomial-time algorithm, and a more subtle approach is needed. In fact, in this case, we provide only a $(1+\zeta)$-approximate solution to the shortest row problem (for appropriate $\zeta > 0$). As is well-known (e.g., see [6, 10]), the exact shortest-row subroutine used in GK may be replaced by an α-approximate one, sacrificing a factor of α in the overall performance ratio of the algorithm in [10]. We obtain the following direct corollary of Theorem 1.

▶ **Corollary 2.** *Combining Theorem 1 with Jain's algorithm, we obtain a strongly polynomial-time*[2] $(2+\varepsilon)$*-approximation algorithm for* (IP)*, that does not use linear programming solvers.*

We once again stress that the above results hold for *any not necessarily bounded* proper cut-requirement function f.

[1] An algorithm is *strongly polynomial* if its number of arithmetic operations, i.e. the number of additions, subtractions, multiplications, divisions and comparisons, is bounded by a polynomial in the dimension of the problem (i.e., the number of data items in the input), and the length of the numbers occurring during the algorithm is bounded by a polynomial in the length of the input.

[2] If rounding up numbers is considered an arithmetic operation.

Further related work

The past 30 years have seen significant research on solving linear programs efficiently; e.g., see the work by Shahrokhi & Matula [21], Luby & Nisan [20], Grigoriadis & Khachian [14, 15], Young [24, 25], Garg & Könemann [10], Fleischer [6], and Iyengar & Bienstock [3]. We refer the reader to two recent surveys by Bienstock [2] and Arora, Hazan & Kale [1].

Particularly relevant to this paper is the work by Fleischer [5] who previously proposed a Lagrangian-based approximation scheme for positive covering LPs with added *variable upper bounds*. Their algorithm builds on [10] and [6], and achieves a performance ratio of $(1+\epsilon)$ for any positive ϵ using $O(\epsilon^{-2} m \log(Cm))$ calls to a separation oracle for the given covering problem; here m denotes the number of variables, and C is bounded by the maximum objective function coefficient. Garg and Khandekar [9] later addressed the same problem, and presented an improved algorithm with $O(m\epsilon^{-2}\log m + \min\{n, \log\log C\})$ calls to an oracle for the most violated constraint.

The algorithms in [5, 9] naturally apply to solving LP relaxations of various network design IPs where the multiplicity $x(e)$ of each edge e is limited to some given upper bound. As the approaches in [5, 9] need to approximate the same type of the shortest row problem, as an immediate corollary of our result, we obtain a strongly polynomial-time $(2+\varepsilon)$-approximation algorithm for (IP) with constant upper bounds on the variables. This captures in particular the interesting case in which we have *binary* constraints for x.

Finally, we also mention the work of Garg & Khandekar [8] who present a fully polyonimal-time approximation algorithm for the fractional *Steiner forest* problem. The algorithm also applies to the more general problem of finding a minimum-cost fractional hitting set of a given collection of *clutters*.

Organization

We first provide some more details on how to implement the iterative rounding algorithm of Jain. We continue and provide a detailed description of GK with approximate oracles in Section 3 for completeness, and describe the shortest-row oracles in Section 4. Finally, in Section 5 we put together all ingredients to prove our main result.

2 Iterative rounding

Recall that Jain's key structural insight was to observe that extreme points $x \in \mathbb{R}^E_+$ of (LP$_1$) have $x(e) \geq 1/2$ for at least one $e \in E$. Jain also noted that, in an implementation of his algorithm, the computation of extreme points may be circumvented. In fact, he suggests obtaining LP (LP$_g$) from (LP$_1$) by adding the constraint $x(g) \geq 1/2$ for some edge $g \in E$. Let opt_g be the objective function value of an optimum solution to (LP$_g$). Jain's structural lemma now implies that $\min_{g\in E} \mathrm{opt}_g$ is at most the optimum value of (LP$_1$). Jain's algorithm can now be implemented by replacing the computation of an optimum basic solution to the residual problem in each iteration, by computing optimal solutions to linear programs of type (LP$_g$) for all edges $g \in E$.

Of course, we can also replace computing an optimal solution to (LP$_g$) with computing an approximate one, at the expense of a slight increase of the final approximation factor. Jain's method in this case is summarized for completeness in Algorithm 1.

▶ **Claim 3.** *Given $\zeta > 0$, Algorithm 1 is a $2(1+\zeta)^{|E|}$-approximation algorithm for* (IP)*. Moreover, Algorithm 1 terminates after at most $|E|$ iterations of step 2.*

Algorithm 1 A $2(1+\zeta)^{|E|}$- approximation algorithm for (IP).

```
1: $I_0 \leftarrow \varnothing$, $z_0(e) \leftarrow 0$ for all $e \in E$, $k \leftarrow 0$
2: while $f(S) - z_k(\delta(S)) > 0$ for some $S \subseteq V$ do
3:     $k \leftarrow k + 1$
4:     for all $g \in E \setminus I_{k-1}$ do
5:         find a $(1+\zeta)$-approximation $x_{k,g}$ for (LP$_g$) with $z := z_{k-1}$ and $I := I_{k-1}$
6:     end for
7:     let $x_k$ be a vector $x_{k,g}$ corresponding to $\min_{g \in E \setminus I_{k-1}} \sum_{e \in E} c(e) x_{k,g}(e)$
8:     $I_k \leftarrow I_{k-1} \cup \{e \in E \setminus I_{k-1} \,|\, x_k(e) = 0 \text{ or } x_k(e) \geq 1/2\}$
9:     for all $e \in E$, let $z_k(e) \leftarrow \lceil x_k(E) \rceil$ if $x_k(e) \geq 1/2$, and $z_k(e) \leftarrow z_{k-1}(e)$ otherwise
10: end while
11: return $z_k$
```

Proof. Jain's structural lemma immediately implies that $I_{k-1} \subsetneq I_k \subseteq E$ for every k. Hence the total number of iterations is at most $|E|$. In iteration k we fix the values of variable $x(e)$, $e \in I_k \setminus I_{k-1}$, and due to the definition of $I_k \setminus I_{k-1}$ we have $z_k(e) \leq 2x_k(e)$, $e \in I_k \setminus I_{k-1}$. The remaining values $x_k(e)$, $e \in E \setminus I_k$ form a valid solution for (LP$_1$) with $z(e) := z_k(e)$, $e \in E$, which is solved with approximation guarantee $(1+\zeta)$ in the $(k+1)$-st iteration. Since there are at most $|E|$ iterations and in step 5 the found solution is a $(1+\zeta)$-approximation of (LP$_g$), we know that the objective value of the output is at most $2(1+\zeta)^{|E|}$ times the objective value of the linear relaxation of (IP), finishing the proof. ◀

Note that by Claim 3, if $\zeta \leq \ln(1+\varepsilon)/|E|$ then Algorithm 1 gives a $2(1+\varepsilon)$-approximation for (IP).

3 Multiplicative weights method

In this section, we briefly review the multiplicative weights method [10] of GK, when applied to a positive covering LP of the form

$$\begin{aligned} \min \quad & \sum_{j \in [n]} c(j)x(j) && \text{(LP}_2\text{)} \\ \text{s.t.} \quad & \sum_{j \in [n]} A(i,j)x(j) \geq b(i) \quad \forall i \in [m], \\ & x \geq 0\,, \end{aligned}$$

where $A(i,j) \geq 0$ for all $i \in [m]$, $j \in [n]$, $b(i) > 0$ for all $i \in [m]$ and $c(j) > 0$ for all $j \in [n]$. Note, that the linear program (LP$_g$) is a positive LP of the above form when simply eliminating variables $x(e)$ for $e \in I$. In the same way, we can exclude the inequalities corresponding to $S \subseteq V$ with $f(S) - z(\delta(S)) \leq 0$.

Given $i \in [m]$ and a vector $x \in \mathbb{R}^n_+$ we define the *length* $\operatorname{len}(i,x)$ of row i with respect to x as

$$\operatorname{len}(i,x) := \sum_{j \in [n]} A(i,j)x(j)/b(i)\,, \tag{1}$$

and we denote by $\operatorname{len}(x)$ the shortest length of a row in A with respect to x, i.e. $\operatorname{len}(x) := \min_{i \in [m]} \operatorname{len}(i,x)$. Now it is straightforward to reformulate (LP$_2$) as

$$\min_{x \geq 0, x \neq 0} \sum_{j \in [n]} c(j)x(j)/\operatorname{len}(x)\,. \tag{2}$$

Algorithm 2 The multiplicative weights algorithm to solve (LP$_2$).

```
1: $\delta \leftarrow (1+\zeta)\big((1+\zeta)n\big)^{-\frac{1}{\zeta}}$, $x_0(j) \leftarrow \delta/c(j)$ for all $j \in [n]$, $y_0(i) \leftarrow 0$ for all $i \in [m]$, $k \leftarrow 0$
2: while $\sum_{j\in[n]} c(j)x_k(j) < 1$ do
3:     $k \leftarrow k+1$
4:     determine a $(1+\zeta)$-approximation for the shortest row with respect to $x_{k-1}$, let it be row $q_k$
5:     determine $j \in [n]$ with the minimum value $c(j)/A(q_k, j)$, let it be column $p_k$
6:     for all $i \in [m]$ do
           $$y_k(i) \leftarrow \begin{cases} y_{k-1}(i) + c(p_k)/A(q_k,p_k) & \text{if} \quad i = q_k \\ y_{k-1}(i) & \text{otherwise}\,. \end{cases}$$
7:     end for
8:     $x_k(j) \leftarrow \big(1 + \zeta\frac{c(p_k)A(q_k,j)}{c(j)A(q_k,p_k)}\big)x_{k-1}(j)$ for all $j \in [n]$
9: end while
10: return $x_k/\operatorname{len}(x_k)$ corresponding to $\min_k \sum_{j\in[n]} c(j)x_k(j)/\operatorname{len}(x_k)$
```

The multiplicative weights method of GK applied to the dual of (LP$_2$) computes an approximate pair of primal and dual solutions in strongly-polynomial time, as long as it is provided with a strongly-polynomial time oracle for determining the row q of shortest length (the *shortest row*) with respect to given lengths $x \in \mathbb{R}^n_+$ as in (1).

It is implicit in the work of [10, 6] that exact oracles can be replaced by approximate ones (incurring a corresponding degradation in performance ratio, of course). Such a modification is described from a packing point of view in [4], for example. Algorithm 2 shows the pseudo code of the algorithm for completeness. In step 4 of the algorithm a $(1+\zeta)$-approximation q of the shortest row with respect to some vector $x \in \mathbb{R}^n_+$ is computed. That is, q is a row for which $\operatorname{len}(q,x) \leq (1+\zeta)\operatorname{len}(x)$. Section 4 describes how to obtain this approximation in strongly-polynomial time.

We give a proof of the next lemma for completeness.

▶ **Lemma 4** (implicit in [10, 6]). *Algorithm 2 is a* $(1+4\zeta)$*-approximation for* (LP$_2$). *Moreover, Algorithm 2 terminates after at most* $\frac{1}{\zeta}\log_{1+\zeta}(1+\zeta)n$ *iterations.*

Proof. Let us define $\beta := \min_k \frac{\sum_{j\in[n]} c(j)x_k(j)}{\operatorname{len}(x_k)}$, below we show that β provides a good approximation for the problem given by (2).

For every $k \geq 1$ we have

$$\sum_{j\in[n]} c(j)x_k(j) - \sum_{j\in[n]} c(j)x_{k-1}(j) =$$

$$\zeta\operatorname{len}(q_k, x_{k-1})c(p_k)b(q_k)/A(q_k,p_k) \leq \zeta(1+\zeta)\operatorname{len}(x_{k-1}) \times \sum_{i\in[m]} b(i)(y_k(i) - y_{k-1}(i))\,.$$

Hence,

$$\sum_{j\in[n]} c(j)x_k(j) \leq \sum_{j\in[n]} c(j)x_0(j) + \zeta(1+\zeta) \times \sum_{h\in[k]}\sum_{i\in[m]} b(i)(y_h(i) - y_{h-1}(i))\operatorname{len}(x_{h-1})\,.$$

Due to the definition of β we have $\mathrm{len}(x_{h-1}) \le \sum_{j\in[n]} c(j)x_{h-1}(j)/\beta$ and thus

$$\sum_{j\in[n]} c(j)x_k(j) \le \\ \sum_{j\in[n]} c(j)x_0(j) + \frac{\zeta(1+\zeta)}{\beta} \times \sum_{h\in[k]} \sum_{i\in[m]} b(i)(y_h(i) - y_{h-1}(i)) \sum_{j\in[n]} c(j)x_{h-1}(j)\,. \tag{3}$$

To show that the right-hand side of (3) is at most $n\delta\, \mathbf{e}^{\zeta(1+\zeta)\sum_{i\in[m]} b(i)y_k(i)/\beta}$, we use induction. Indeed, the case $k=0$ is clear, and to show the statement consider

$$\sum_{j\in[n]} c(j)x_0(j) + \frac{\zeta(1+\zeta)}{\beta} \times \sum_{h\in[k]} \sum_{i\in[m]} b(i)(y_h(i) - y_{h-1}(i)) \sum_{j\in[n]} c(j)x_{h-1}(j)\,,$$

which equals

$$\sum_{j\in[n]} c(j)x_0(j) + \frac{\zeta(1+\zeta)}{\beta} \times \Big(\sum_{h\in[k-1]} \sum_{i\in[m]} b(i)(y_h(i) - y_{h-1}(i)) \sum_{j\in[n]} c(j)x_{h-1}(j) + \\ \sum_{i\in[m]} b(i)\big(y_k(i) - y_{k-1}(i)\big) \sum_{j\in[n]} c(j)x_{k-1}(j)\Big)$$

Due to (3) we conclude that the last expression is at most

$$\Big(1 + \frac{\zeta(1+\zeta)}{\beta} \sum_{i\in[m]} b(i)\big(y_k(i) - y_{k-1}(i)\big)\Big) \times \Big(\sum_{j\in[n]} c(j)x_0(j) + \frac{\zeta(1+\zeta)}{\beta} \times \\ \sum_{h\in[k-1]} \sum_{i\in[m]} b(i)\big(y_h(i) - y_{h-1}(i)\big) \sum_{j\in[n]} c(j)x_{h-1}(j)\Big)\,.$$

Using the inequality $(1+\alpha) \le \mathbf{e}^{\alpha}$, $\alpha \in \mathbb{R}$ and the induction hypothesis we upper-bound the expression above by

$$\mathbf{e}^{\zeta(1+\zeta)\sum_{i\in[m]} b(i)\left(y_k(i)-y_{k-1}(i)\right)/\beta} \times n\delta\, \mathbf{e}^{\zeta(1+\zeta)\sum_{i\in[m]} b(i)y_{k-1}(i)/\beta} = n\delta\, \mathbf{e}^{\zeta(1+\zeta)\sum_{i\in[m]} b(i)y_k(i)/\beta}\,.$$

Now let us consider the last iteration t, where we have

$$1 \le \sum_{j\in[n]} c(j)x_t(j) \le n\delta\, \mathbf{e}^{\zeta(1+\zeta)\sum_{i\in[m]} b(i)y_t(i)/\beta}\,,$$

and thus

$$\frac{\beta}{\sum_{i\in[m]} b(i)y_t(i)} \le \frac{\zeta(1+\zeta)}{\ln((n\delta)^{-1})} \tag{4}$$

whenever $n\delta < 1$.

Now let us show that $y_t/\log_{1+\zeta}\left(\frac{1+\zeta}{\delta}\right)$ is a feasible solution for the dual of (LP$_2$). It is enough to show that

$$\max_j \sum_{i\in[m]} \frac{A(i,j)y_t(i)}{c(j)} \le \log_{1+\zeta} \frac{1+\zeta}{\delta}\,.$$

To see this note that for every $j \in [n]$ and every k

$$\sum_{i\in[m]} \frac{A(i,j)y_k(i)}{c(j)} - \sum_{i\in[m]} \frac{A(i,j)y_{k-1}(i)}{c(j)} = \frac{A(q_k,j)}{c(j)} \frac{c(p_k)}{A(q_k,p_k)} \leq 1\,,$$

and $\sum_{i\in[m]} \frac{A(i,j)y_0(i)}{c(j)} = 0$. On the other hand for every $j \in [n]$ and every k

$$\frac{x_k(j)}{x_{k-1}(j)} = 1 + \zeta \frac{c(p_k)A(q_k,j)}{c(j)A(q_k,p_k)} \leq 1 + \zeta\,,$$

$x_0(j) = \delta/c(j)$ and due to the termination condition $x_{t-1}(j) < 1/c(j)$ and hence $x_t(j) < (1+\zeta)/c(j)$. This implies that the algorithm terminates after at most $\log_{1+\zeta} \frac{1+\zeta}{\delta}$ iterations. Thus, $y_t / \log_{1+\zeta}\left(\frac{1+\zeta}{\delta}\right)$ is a feasible solution for the dual of (LP_2).

Hence, the algorithm provides a feasible solution for (2) with value β, which is an approximation with guarantee

$$\frac{\zeta(1+\zeta)}{\ln((n\delta)^{-1})} \log_{1+\zeta} \frac{1+\zeta}{\delta} = \frac{\zeta(1+\zeta)}{\ln(1+\zeta)} \frac{\ln \frac{1+\zeta}{\delta}}{\ln((n\delta)^{-1})}\,,$$

due to (4) and the fact that $y_t / \log_{1+\zeta}\left(\frac{1+\zeta}{\delta}\right)$ is a feasible solution for the dual of (LP_2). Thus, we obtain

$$\frac{\zeta(1+\zeta)}{\ln(1+\zeta)} \frac{\ln \frac{1+\zeta}{\delta}}{\ln((n\delta)^{-1})} = \frac{\zeta(1+\zeta)}{(1-\zeta)\ln(1+\zeta)} \leq \frac{\zeta(1+\zeta)}{(1-\zeta)(\zeta - \zeta^2/2)} \leq \frac{1+\zeta}{(1-\zeta)^2}\,,$$

which is at most $(1+4\zeta)$ for $\zeta \leq 0.15$. ◀

4 The shortest row problem

In this section we describe how to (approximately) solve the shortest row problem needed in Algorithm 2 when applied to (LP_1). We start by stating the following simple remark, that we will need at the end of our analysis.

▶ Remark 5. For every $k \geq 1$ and every $S \subseteq V$, we have $f(S) - z_k(\delta(S)) \leq |E|/2$ in Algorithm 1.

Proof. Define $x \in \mathbb{R}_+^E$ by letting $x(e) := x_k(e)$ whenever $x_k(e) < 1/2$, and let $x(e) := 0$ otherwise. Then note that $|E|/2 \geq x(\delta(S)) \geq f(S) - z_k(\delta(S))$, due to the feasibility of x in (LP_1) with $z := z_k$ and $I := I_k$. ◀

Let us recall that, for given $x \in \mathbb{R}_+^E$, $z \in \mathbb{Z}_+^E$, and proper function f, the shortest row problem we need to solve is the following:

$$\min_{\substack{f(S)-z(\delta(S))\geq 1 \\ S\subseteq V}} \frac{x(\delta(S))}{f(S) - z(\delta(S))}.$$

The above shortest row problem is quite easy to solve when $z = \nvdash$. In this case, Gabow et al. [7] give a strongly-polynomial time separation oracle based on the construction of *Gomory-Hu trees* [13], as we are now going to explain.

Given a graph $G = (V,E)$ and values $x(e) \in \mathbb{R}_+$ for each $e \in E$, a *Gomory-Hu tree* [13] is a capacitated tree $T = (V,J)$ such that for any two vertices $v, u \in V$ the minimum x-value of a cut in G separating v and u equals the minimum x-value among the u-v cuts induced by the edges of T. More concretely, let S_e and $V \setminus S_e$ induce connected components in T after removing e from T. We then have

$$\min_{\substack{u \in S, v \notin S \\ S \subseteq V}} x(\delta(S)) = \min_{\substack{u \in S_e, v \notin S_e \\ e \in J}} x(\delta(S_e))\,.$$

The next lemma shows that in order to find the shortest row in the first iteration of step 2 in Algorithm 1 (i.e. when $z = \nvdash$), it is enough to compute a Gomory-Hu tree with respect to values $x(e) \in \mathbb{R}_+$, $e \in E$.

▶ **Lemma 6** ([7]). *Given a graph $G = (V, E)$, a proper function $f : 2^V \to \mathbb{Z}_+$ and a Gomory-Hu tree $T = (V, J)$ with respect to values $x(e) \in \mathbb{R}_+$, $e \in E$, we have*

$$\min_{\substack{f(S) \neq 0 \\ S \subseteq V}} x(\delta(S))/f(S) = \min_{\substack{f(S_e) \neq 0 \\ e \in J}} x(\delta(S_e))/f(S_e)\,.$$

Proof. Consider $S \subseteq V$ and the edges $\delta_T(S)$ in the Gomory-Hu tree T defined by S. By definition of a Gomory-Hu tree $x(\delta(S)) \geq x(\delta(S_e))$ for every $e \in \delta_T(S)$, due to the cut in T incurred by the vertices incident to e. Thus, to prove the claim it is enough to show that

$$f(S) \leq \max_{e \in \delta_T(S)} f(S_e)\,. \tag{5}$$

To show the last inequality let $V_1, \ldots, V_k$ be vertex sets of the connected components after removing S in T. Thus, $V_1, \ldots,\ V_k$ form a partition of $V \setminus S$, and so $\max_{i \in [k]} f(V_i) \geq f(V \setminus S) = f(S)$. Choose $i \in [k]$. Replacing S_e by $V \setminus S_e$, we can assume that S_e and V_i are disjoint for every $e \in \delta_T(V_i)$. Thus the sets S_e with $e \in \delta_T(V_i)$ partition $V \setminus V_i$, showing that $f(V_i) \leq \max_{e \in \delta_T(V_i)} f(S_e)$. Since, $\delta_T(V_i)$, $i \in [k]$ partition $\delta(S)$ we get (5), finishing the proof. ◀

In the later iterations of steps 2 in Algorithm 1, the inequality corresponding to $S \subseteq V$ has the form $x(\delta(S)) \geq g(S)$, where $g : 2^V \to \mathbb{Z}_+$ is such that $g(S) = f(S) - z(\delta(S))$ for some $z(e) \in \mathbb{Z}_+$, $e \in E$, and a proper function f. Once $z \neq \nvdash$, $g(S)$ is not a proper function any more and unfortunately, Gabow et al.'s algorithm can not be used directly. We do not know how to solve this problem *exactly* in strongly-polynomial time, but we can approximate it using the following observation.

Fix a value $\gamma > 0$, and let us check whether the optimal solution of the shortest row problem has a value less than γ. The crucial fact is that given $x(e) \in \mathbb{R}_+$, $e \in E$ and $\gamma > 0$ checking whether

$$\min_{\substack{f(S) - z(\delta(S)) \geq 1 \\ S \subseteq V}} \frac{x(\delta(S))}{f(S) - z(\delta(S))} < \gamma\,.$$

is equivalent to checking whether

$$x(\delta(S))/\gamma + z(\delta(S)) < f(S)$$

for some $S \subseteq V$, i.e. it can be reduced to finding

$$\min_{\substack{f(S) \neq 0 \\ S \subseteq V}} \frac{x(\delta(S))/\gamma + z(\delta(S))}{f(S)}.$$

Therefore, we can apply Lemma 6 after replacing $x(e)$ with $x(e)/\gamma + z(e)$. This enables us to use binary search to find a $(1 + \zeta)$-approximation for the shortest row indexed by

Algorithm 3 Determining upper and lower bounds $\gamma_{\min}$, $\gamma_{\max}$.

```
$G_0 = (V_0, E_0) \leftarrow G = (V, E)$, $f_0(S) \leftarrow f(S)$ for all $S \subseteq V$, $k \leftarrow 0$
while $f_k(S) - z(\delta_{G_k}(S)) > 0$ for some $S \subseteq V_k$ do
    find a set $S \subseteq V_k$ such that $f_k(S) - z(\delta_{G_k}(S)) \geq 1$, let it be $S_k$
    determine $e \in \delta_{G_k}(S_k)$ with maximum $x(e)$, let it be $e_k \leftarrow \{u_k, v_k\}$
    contract $e_k$ in $G_k$ (keeping multiple copies of edges) to obtain $G_{k+1}$, and set
        $$f_{k+1}(S) \leftarrow \begin{cases} f_k(S \cup \{u_k, v_k\} \setminus w_k) & \text{if} \quad w_k \in S \\ f_k(S) & \text{otherwise}, \end{cases}$$
    for all $S \subseteq V_{k+1}$, where $w_k$ is the vertex in $G_{k+1}$ corresponding to the contracted edge $e_k$.
    $k \leftarrow k + 1$
end while
$\gamma_{\min} \leftarrow \min_k 2x(e_k)/|E|$, and let $p$ be the index for which this minimum is achieved
$\gamma_{\max} \leftarrow x(\delta(U_p))$, where $U_p$ is the vertex subset of $V$ corresponding to the vertex subset $S_p$ of $V_p$
return $\gamma_{\min}, \gamma_{\max}$
```

vertex subsets whenever we have a lower bound $\gamma_{\min}$ and an upper bound $\gamma_{\max}$ on the length of the shortest row. Giving trivial bounds on such a value (e.g. 1 and $(|E| \cdot \max_S f(S))$) is of course easy. However, given an interval $[\gamma_{\min}, \gamma_{\max}]$ for binary search we have to construct a Gomory-Hu tree $\lceil \log_{1+\zeta} \gamma_{\max}/\gamma_{\min} \rceil$ times, and therefore we need that $\gamma_{\max}/\gamma_{\min}$ is independent of the size of f in order to achieve strong polynomiality. To this aim, we propose Algorithm 3.

▶ **Lemma 7.** *Algorithm 3 computes an interval $[\gamma_{\min}, \gamma_{\max}]$, which contains the shortest row length with respect to $x(e) \in \mathbb{R}_+$, $e \in E$. Moreover, $\gamma_{\max}/\gamma_{\min} \leq |E|^2/2$, and the algorithm runs in strongly-polynomial time.*

Proof. Algorithm 3 works as follows. It does a sequence of at most $|V|$ iterations. In iteration k, it takes an arbitrary subset S corresponding to a violated cut, i.e. such that $f_k(S) - z(\delta_{G_k}(S)) > 0$, and contracts the edge e_k in this cut of maximum x-value. Contracting this edge naturally yields a graph G_{k+1} and a function f_{k+1} to use in the next iteration. Note that a violated subset S can be computed efficiently given that f is a proper function [12].

Our first claim is that $\gamma_{\min}$ is a valid lower bound on the shortest row length. In other words, we claim that for every $S : f(S) - z(\delta(S)) \geq 1$, we have

$$\frac{x(\delta(S))}{f(S) - z(\delta(S))} \geq \frac{x(e_p)}{|E|/2} = \gamma_{\min} .$$

Due to the termination condition, for every $S \subseteq V$ with $f(S) - z(\delta(S)) \geq 1$ the edge set $\delta(S)$ contains at least one of the edges $e_1,\dots,e_t$ selected by the algorithms during its t iterations. Therefore, $x(\delta(S)) \geq x(e_p)$, by the choice of p in step 8. Moreover, by Remark 5 $f(S) - z(\delta(S)) \leq |E|/2$. The claim then follows.

Our second claim is that $\gamma_{\max}$ is a valid upper bound on the shortest row length. To see this, note that $f(U_p) - z(\delta(U_p)) \geq 1$ because $f(U_p) = f_p(S_p)$ and $z(\delta(U_p)) = z(\delta_{G_p}(S_p))$, proving that $\gamma_{\max}$ is a valid upper bound for the shortest length of a row indexed by $S \subseteq V$.

Finally, recalling that e_p satisfies $x(e_p) = \max_{e \in \delta(U_p)} x(e)$ (step 4), we have

$$\gamma_{\max}/\gamma_{\min} = \frac{x(\delta(U_p))}{2x(e_p)/|E|} \leq |E|^2/2 . \qquad ◀$$

5 Concluding remarks

We are now ready to put all pieces together and give a proof of Theorem 1 and Corollary 2 stated in the introduction.

Proof of Theorem 1. Given an $\varepsilon > 0$, we apply Algorithm 2 to (LP$_1$) with $\zeta = \ln(1+\varepsilon)/|E|$. Algorithm 2 in its turn approximates the shortest row at most $O((\ln|V|)/\zeta^2)$ times[3]. It makes a call to Algorithm 3, computing at most $|V|$ Gomory-Hu trees and afterwards the binary search needs $O((\ln|E|)/\zeta)$ computations of a Gomory-Hu tree in $G = (V, E)$. Recall, that $\zeta = \ln(1+\varepsilon)/|E| = \Theta(\varepsilon/|E|)$ and hence each linear program appearing in Algorithm 1 is solved in time dominated by finding $O(|E|^3(\ln|E|)^2/\varepsilon^3)$ Gomory-Hu trees. Note that a Gomory-Hu tree for $G = (V, E)$ with respect to values $x(e) \in \mathbb{R}_+$, $e \in E$ can be found by $|V|$ computations of the minimum cut in G [13], so a Gomory-Hu tree can be found in strongly-polynomial time. ◀

The number of times our algorithm solves the Gomory-Hu tree problem is substantially smaller than the corresponding number for the Ellipsoid method given the classical estimation for the encoding length of vertices or given that $\max_S f(S)$ is sufficiently large, because this number for the Ellipsoid method grows proportionally with the logarithm of $\max_S f(S)$.

Proof of Corollary 2. To obtain a $(2+\varepsilon)$-approximation for (IP) we apply Algorithm 1. Algorithm 1 solves $O(|E|^2)$ linear programs, i.e. there are $O(|E|^2)$ calls of Algorithm 1 to Algorithm 2 to solve linear programs, and it makes at most $|E|$ roundings. Considering rounding as a basic operation, the result follows. ◀

We conclude the paper with some open questions. It remains open whether one is able to provide a 2-approximation algorithm for (IP), which does not need to solve linear programs. This question is among the top 10 open questions in the theory of approximation algorithms according to Shmoys and Williamson [23]. In our opinion, a good intermediate question is whether it is possible to give an algorithm with a constant approximation guarantee such that the number of linear programs solved in its course is bounded by a constant. One way to prove this could be to exploit that after each rounding in the algorithm of Jain [19] we have a sufficiently "good" feasible point for the new linear program.

References

1 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

2 Daniel Bienstock. *Potential function methods for approximately solving linear programming problems: theory and practice*, volume 53. Springer Science & Business Media, 2006.

3 Daniel Bienstock and Garud Iyengar. Solving fractional packing problems in o ast $(1/\varepsilon)$ iterations. In *Proc. of the 36th Annual ACM Symp. on Theory of Computing*, pages 146–155. ACM, 2004.

4 Khaled Elbassioni, Kurt Mehlhorn, and Fahimeh Ramezani. Towards more practical linear programming-based techniques for algorithmic mechanism design. *SAGT*, pages 98–109, 2015.

5 Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1001–1010. Society for Industrial and Applied Mathematics, 2004.

[3] Here, we use the inequality that $1 + \zeta < \mathbf{e}^{\zeta} < 1 + \frac{7}{4}\zeta$ for $0 < \zeta < 1$.

6 Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

7 Harold N. Gabow, Michel X. Goemans, and David P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming*, 82(1-2, Ser. B):13–40, 1998. Networks and matroids; Sequencing and scheduling.

8 Naveen Garg and Rohit Khandekar. Fast approximation algorithms for fractional steiner forest and related problems. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 500–509. IEEE, 2002.

9 Naveen Garg and Rohit Khandekar. Fractional covering with upper bounds on the variables: Solving lps with negative entries. In *Algorithms–ESA 2004*, pages 371–382. Springer, 2004.

10 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007. doi:10.1137/S0097539704446232.

11 M. X. Goemans, D. B. Shmoys, A. V. Goldberg, É. Tardos, S. Plotkin, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proc. of the 5th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'94)*, pages 223–232. ACM, 1994.

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

13 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.*, 9:551–570, 1961.

14 Michael D Grigoriadis and Leonid G Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4(1):86–107, 1994.

15 Michael D Grigoriadis and Leonid G Khachiyan. Approximate minimum-cost multicommodity flows in $\tilde{o}(\epsilon - 2knm)$ time. *Mathematical Programming*, 75(3):477–482, 1996.

16 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics: Study and Research Texts*. Springer-Verlag, Berlin, 1988. doi:10.1007/978-3-642-97881-4.

17 Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

18 Dorit S Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

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

20 Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 448–457. ACM, 1993.

21 Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.

22 Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

23 David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, Cambridge, 2011. doi:10.1017/CBO9780511921735.

24 Neal E Young. Randomized rounding without solving the linear program. In *SODA*, volume 95, pages 170–178, 1995.

25 Neal E Young. Sequential and parallel algorithms for mixed packing and covering. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 538–546. IEEE, 2001.

Space-Time Trade-Offs for the Shortest Unique Substring Problem

Arnab Ganguly*[1], Wing-Kai Hon†[2], Rahul Shah[3], and Sharma V. Thankachan[4]

1 **School of EECS, Louisiana State University, Baton Rouge, USA**
agangu4@lsu.edu

2 **Department of CS, National Tsing Hua University, Taiwan**
wkhon@cs.nthu.edu.tw

3 **School of EECS, Louisiana State University, Baton Rouge, USA; and National Science Foundation, USA**
rahul@csc.lsu.edu, rahul@nsf.gov

4 **Department of CS, University of Central Florida, Orlando, USA**
sharma.thankachan@gmail.com

Abstract

Given a string $\mathsf{X}[1,n]$ and a position $k \in [1,n]$, the *Shortest Unique Substring* of X covering k, denoted by S_k, is a substring $\mathsf{X}[i,j]$ of X which satisfies the following conditions: (i) $i \leq k \leq j$, (ii) i is the only position where there is an occurrence of $\mathsf{X}[i,j]$, and (iii) $j-i$ is minimized. The best-known algorithm [Hon et al., ISAAC 2015] can find S_k for all $k \in [1,n]$ in time $\mathcal{O}(n)$ using the string X and additional $2n$ words of working space. Let τ be a given parameter. We present the following new results. For any given $k \in [1,n]$, we can compute S_k via a deterministic algorithm in $\mathcal{O}(n\tau^2 \log \frac{n}{\tau})$ time using X and additional $\mathcal{O}(n/\tau)$ words of working space. For every $k \in [1,n]$, we can compute S_k via a deterministic algorithm in $\mathcal{O}(n\tau^2 \log n)$ time using X and additional $\mathcal{O}(n/\tau)$ words and $4n + o(n)$ bits of working space. For both problems above, we present an $\mathcal{O}(n\tau \log^{c+1} n)$-time randomized algorithm that uses $n/\log^c n$ words in addition to that mentioned above, where $c \geq 0$ is an arbitrary constant. In this case, the reported string is unique and covers k, but with probability at most $n^{-\mathcal{O}(1)}$, may not be the shortest. As a consequence of our techniques, we also obtain similar space-and-time tradeoffs for a related problem of finding Maximal Unique Matches of two strings [Delcher et al., Nucleic Acids Research 1999].

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases Suffix Tree, Sparsification, Rabin-Karp Fingerprint, Probabilistic z-Fast Trie, Succinct Data-Structures

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.34

1 Introduction

We consider a string $\mathsf{X}[1,n]$ with characters from an ordered alphabet Σ of cardinality σ. The ith character, $i \in [1,n]$, is denoted by $\mathsf{X}[i]$, and $\mathsf{X}[i,j]$, $1 \leq i \leq j \leq n$, is the substring $\mathsf{X}[i]\ \mathsf{X}[i+1] \ldots \mathsf{X}[j]$. We denote by $|\mathsf{X}[i,j]|$ the length $(j-i+1)$ of the substring $\mathsf{X}[i,j]$. A *suffix starting at* i is the string $\mathsf{X}[i,n]$ and a *prefix ending at* i is the string $\mathsf{X}[1,i]$. A *right*

* The work of Arnab Ganguly was supported by National Science Foundation Grants CCF–1218904 and CCF–1527435.

† The work of Wing-Kai Hon was supported by MOST Grant 105-2918-I-007-006 and MOST Grant 102-2221-E-007-068-MY3.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 34; pp. 34:1–34:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

extension of $\mathsf{X}[i,j]$ is a string $\mathsf{X}[i,j']$, where $j' > j$. A substring $\mathsf{X}[i,j]$ *covers a position* k iff $i \leq k \leq j$. A substring $\mathsf{X}[i,j]$ is *unique* iff $\mathsf{X}[i,n]$ is the only suffix having $\mathsf{X}[i,j]$ as a prefix. A substring $\mathsf{X}[i,j]$ is *repeating* iff there exists $i' \neq i$ such that $\mathsf{X}[i,j]$ is a prefix of $\mathsf{X}[i',n]$.

▶ **Definition 1** (Shortest Unique Substring Covering k)**.** A substring $\mathsf{X}[i,j]$ is a shortest unique substring covering a position k iff (i) $\mathsf{X}[i,j]$ covers k, and (ii) there does not exist a substring $\mathsf{X}[i',j']$ that covers k and satisfies $j'-i' < j-i$.

We now present the following problems that will be discussed in the rest of the paper. Throughout this paper, we will use S_k to denote any shortest unique substring of X covering k. Note that there may be multiple choices of S_k.

▶ **Problem 2** (Single k)**.** *Given* $\mathsf{X}[1,n]$ *and a position* $k \in [1,n]$*, find any* S_k*.*

▶ **Problem 3** (All k)**.** *Given* $\mathsf{X}[1,n]$*, find any* S_k *for every* $k \in [1,n]$*.*

Previous Works and Our Contribution. To the best of our knowledge, the formal definitions presented in Problems 2 and 3 were introduced by Pei et al. [19]. They also listed several potential applications, for e.g., document searching on the internet. Arguably, the most important applications lie in the field of Computational Biology. A few of them are (see [19] and references therein): finding unique DNA signatures between closely related organisms, aiding polymerase chain reaction (PCR) primer design, genome mapability, and next-generation short reads sequencing.

For Problem 2, Pei et al. [19] presented an $\mathcal{O}(n)$ time and $\Theta(n)$ space (in words) solution. For the second problem, their method incurred a time of $\mathcal{O}(n \cdot h)$, where h is a variable which for most practical purposes can be taken to be a constant. In the worst-case, however, h is $\mathcal{O}(n)$; therefore, their solution takes $\mathcal{O}(n^2)$ time, the space remains at $\Theta(n)$ words. This is the first drawback of their approach. More importantly though, their solution is intrinsically based on the *Suffix Tree* of the string X. A suffix tree [11] ST of a string $S[1,m]$ is a compacted trie on the set of all non-empty suffixes of the string S. The suffix tree has m leaves (one per each suffix), and at most $(m-1)$ internal nodes. The leaves in the suffix tree are arranged from left-to-right in the lexicographic order of the corresponding suffix they represent. The space occupied is $\Theta(m)$ words, or equivalently $\Theta(m \log m)$ bits. (We assume the standard Word-RAM model of computation, where the size of a machine word is $\Theta(\log m)$ bits. Also, all logarithms are in base 2.)

Unfortunately, for most practical purposes, the suffix tree of a string S occupies space much larger (15-50 times) compared to the $|S| \log |\Sigma_S|$ bits of space needed by S. Here, Σ_S is the alphabet from which the characters in S are drawn. (Typically, $\Sigma_S = \{1, 2, \ldots, |\Sigma_S|\}$.) The space occupancy issue becomes more profound in the case when strings are much larger in comparison to the size of the alphabet. An example is the DNA, in which the alphabet has size four, but the lengths of the strings (such as in Human Genome) are typically in the billions. Even with a space-efficient implementation, such as in [16], a suffix tree occupies 40 Gigabytes, whereas the input Human Genome occupies only 700 Megabytes. Since a primary application of the Shortest Unique Substring (SUS) problem involves DNA, this presents a serious bottleneck, as has been corroborated by the experimental results of Ileri et al. [14], who were unable to run the algorithm of Pei et al. [19] for massive data sizes.

To alleviate the running time of $\mathcal{O}(n^2)$ for Problem 3, Ileri et al. [14] introduced an $\mathcal{O}(n)$-time and $\Theta(n)$-word algorithm. More importantly, their algorithm is more space-efficient than the algorithm of Pei et al. [19]. They showed that their algorithm not only saves space by a factor of 20, but also attains a speedup by a factor of 4. The space efficiency is achieved by replacing the suffix tree with a combination of the *Suffix Array*, its inverse, and the *LCP array* of the string X. The suffix array [11] of $S[1,m]$ is an array $\mathsf{SA}_S[1,m]$ such that $\mathsf{SA}_S[i] = j$ iff $S[j,m]$ is the ith lexicographically smallest suffix. The inverse suffix array $\mathsf{SA}_S^{-1}[1,m]$ is an array such that $\mathsf{SA}_S^{-1}[j] = i$ iff $\mathsf{SA}_S[i] = j$. The Longest Common Prefix (LCP) array [11] of S is an array $\mathsf{LCP}[1,m]$ such that $\mathsf{LCP}[m] = -1$ and for $i < m$, $\mathsf{LCP}[i]$ equals the length of the LCP of the suffixes starting at $\mathsf{SA}_S[i]$ and $\mathsf{SA}_S[i+1]$. Hon et al. [13] achieved further space improvements by introducing an in-place framework. Specifically, their algorithm needs space $2n$ words in addition to that needed for storing the string X. Remarkably, the time needed to compute S_k for every k still remains $\mathcal{O}(n)$. Furthermore, they argued that $2n$ words is the minimum space needed to store S_k values explicitly, as we need to store the start position and length of each S_k.

Despite all the efforts that have been invested into the SUS problems, the current best solution of Hon et al. [13] still uses $2n$ words of space in addition to the space needed by the input string X. Therefore, an important question is whether we can solve the problems using $o(n)$ words of additional space. We consider the following sub-linear space setting. In addition to the input string X of length n, a parameter τ is provided. The task is to find S_k using space $\mathcal{O}(n/\tau)$ words in addition to the space needed for storing X. In this setting, we present the following solutions for Problems 2 and 3.

- For any given $k \in [1,n]$, we can compute S_k via a deterministic algorithm in $\mathcal{O}(n\tau^2 \log \frac{n}{\tau})$ time using $\mathcal{O}(n/\tau)$-words of additional working space.
- For every $k \in [1,n]$, we can compute S_k via a deterministic algorithm in $\mathcal{O}(n\tau^2 \log n)$ time using $\mathcal{O}(n/\tau)$-words and $4n + o(n)$-bits of additional working space.

We assume $\tau = \omega(1)$. Otherwise, we can simply use the algorithm of Hon et al. [13]. Thus, we present the first algorithm which needs $o(n)$ words of additional space for computing SUS. We also present a randomized algorithm which reduces the above running time to $\mathcal{O}(n\tau \log^{c+1} n)$ by using an additional $n/\log^c n$ words, where $c \geq 0$ is any arbitrary constant. Each computed S_k is unique and covers k, but with probability at most $n^{-\mathcal{O}(1)}$, may not be the shortest. Note that in this case, even by choosing $c = 0$, our space requirements are strictly better (in the asymptotic sense) than that of Hon et al. [13]. By choosing $\tau = \log n$, our algorithm achieves a space-factor improvement of $\mathcal{O}(\log n)$, while matching the best known running time of $\mathcal{O}(n)$ within poly-logarithmic factors.

We remark that our techniques imply (almost) the same results (compact space and succinct index) attained by Belazzougui and Cunial [2] for a related problem of finding the shortest unique prefix of every suffix of X. Our techniques also imply the first sub-linear space algorithm for the related problem of finding Maximal Unique Matches (MUM) of two strings [6].

Roadmap. We first present the two deterministic algorithms in Sections 2 and 3 respectively. Section 4 introduces the randomized algorithms. A brief discussion on the MUM problem [6] is presented Section 5.

2 Deterministic Algorithm for Single k

We begin with the following key observation.

▶ **Observation 4.** S_k *is either the shortest unique prefix of a suffix that starts at a position* $i \leq k$*, or is the smallest right extension till* k *of such a prefix.*

With this key intuition, we define LS_i as the shortest unique prefix of the suffix $\mathsf{X}[i,n]$.

▶ **Observation 5.** *LS_1 is defined, whereas LS_i for $i > 1$ may not be defined. If LS_i is not defined, then for any $i' > i$, $\mathsf{LS}_{i'}$ is also not defined.*

For any $i \leq k$, we define LS_i^k as LS_i if LS_i covers k; otherwise, LS_i^k is the right extension of LS_i up to the position k, i.e., $\mathsf{LS}_i^k = \mathsf{LS}_i \circ \mathsf{X}[i + |\mathsf{LS}_i|, k]$, where $\circ$ denotes concatenation. By this definition, S_k is a minimum length LS_i^k, where $i \leq k$ and LS_i is defined. Moving forward, we will represent S_k by two integers: the starting position of S_k and the length $|\mathsf{S}_k|$.

We first present the general idea behind the previous works. Once we know LS_i for every $i \leq k$, where defined, we first compute LS_i^k. Following this, S_k is computed simply by selecting a LS_i^k of minimum length. Specifically, start at $i = 1$, and compute the longest repeating prefix of $\mathsf{X}[i, n]$. Using the inverse suffix array and the LCP array, this can be easily computed. If the length of this prefix is $(n - i + 1)$, then clearly LS_i is not defined. Otherwise, compute LS_i^k from LS_i, and repeat the process with $(i + 1)$. Finally, compute the minimum length LS_i^k, once we reach an i such that either LS_i is not defined, or $i > k$.

In our case, we cannot construct the entire suffix array and LCP array, as it will violate our space constraints. Also, storing all the LS_i or LS_i^k values is not an option, as in the worst-case the space will become $\Theta(n)$ words. Therefore, we will compute LS_i for a carefully chosen set of $\mathcal{O}(n/\tau)$ suffixes. Based on this, we present the following crucial lemma.

▶ **Lemma 6.** *Let $\mathcal{I}_i = \{i, i + \tau, i + 2\tau, \dots\}$, $i \in [1, \tau]$, be a set of at most $\lceil n/\tau \rceil$ suffixes. For every $i' \in \mathcal{I}_i$, we can compute $\mathsf{LS}_{i'}$ in $\mathcal{O}(n\tau \log \frac{n}{\tau})$ time using X and additional $\mathcal{O}(n/\tau)$ words of working space.*

Using the above lemma, we prove the following theorem, which presents our first result.

▶ **Theorem 7.** *For any given $k \in [1, n]$, we can find S_k in $\mathcal{O}(n\tau^2 \log \frac{n}{\tau})$ time using X and additional $\mathcal{O}(n/\tau)$ words of working space.*

Proof. Initialize $S = n$ and $sp = 1$. Using Lemma 6, we first compute LS_j for every $j \in \mathcal{I}_i$ by choosing $i = 1$. Use LS_j to compute LS_j^k for every $j \in \mathcal{I}_1$. Assign $S = \min_{j \in \mathcal{I}_i}\{S, |\mathsf{LS}_j^k|\}$. If S is updated, then assign sp to the corresponding j. Repeat the process with $i = 2, 3, \dots, \tau$. Now, it remains to find LS_j for all suffixes with $j \in [n - \tau + 2, n]$. To find LS_j and LS_j^k for these suffixes simply use a brute-force approach. Since there are $\tau - 1$ suffixes, each of length at most $\tau - 1$, the time needed is $\mathcal{O}(n\tau^2)$. At each step, update S and sp accordingly. Finally, S_k is given by S and sp. Clearly, the claimed time and space bounds are met. ◀

2.1 Proof of Lemma 6

The central idea behind the proof is the use of a sparsification technique introduced by Hon et al. [12]. In particular, we create a sampled suffix tree ST_i by using a set of roughly n/τ regularly spaced suffixes, where the first suffix starts at position i. Now, we match the string X in ST_i, starting with the position $j = 1$ if $i > 1$, and with $j = 2$ otherwise. Using ST_i, we can find the longest repeating prefix of every sampled suffix w.r.t the positions $j, j + \tau, j + 2\tau, \dots$. Then the process is repeated with every value of $j \in [1, \tau]$, where $j \neq i$. Finally, we use the longest repeating prefix of each sampled suffix, and extend it by one character to find $\mathsf{LS}_{i'}$ for each $i' \in \mathcal{I}_i$. We now present the details.

Pre-process: Consider every substring of X of length τ that starts at a position which lies in the set $\mathcal{I}_i = \{i, i + \tau, i + 2\tau, \dots\}$. We first create a compacted trie $\mathcal{T}$ of these substrings, and ignore the last substring, say X'_i, of X if it has length less than τ. While creating $\mathcal{T}$, for every node u, store in a balanced binary search tree (BST) the first characters that label the edges starting from u. This BST will allow us to efficiently select a correct edge (or create a new one) when a new string is inserted. Since the number of strings considered is at most $\lceil n/\tau \rceil$, the number of nodes in $\mathcal{T}$ is at most $2\lceil n/\tau \rceil$. Likewise, at any moment, the number of nodes in all the BSTs combined is at most $2\lceil n/\tau \rceil$, implying a search or insert operation requires $\mathcal{O}(\log \frac{n}{\tau})$ time. Clearly, the space needed to create $\mathcal{T}$ is $\mathcal{O}(n/\tau)$, and the time required is $\mathcal{O}(n + \frac{n}{\tau} \log \frac{n}{\tau})$. Note that each τ-length substring corresponds to a (not necessarily unique) leaf in $\mathcal{T}$, where the leaves are numbered according to the lexicographic order of the substring they represent. We create a new (compressed) string X_i^τ by mapping each τ-length substring of X starting at a position in $\mathcal{I}_i$ to the corresponding leaf number. (We ignore the string X_i' and the characters before i while creating X_i^τ.) Let Σ_i denote the alphabet of X_i^τ. Note that $\Sigma_i^\tau = \{1, 2, \ldots, |\Sigma_i^\tau|\}$, where $|\Sigma_i^\tau| \leq \lceil \frac{n}{\tau} \rceil$ is the number of leaves in $\mathcal{T}$. Also note that for any two integers $p, q \in \Sigma_i^\tau$, $p < q$ iff the string corresponding to leaf p in $\mathcal{T}$ is lexicographically smaller than the one corresponding to q.

Construct a suffix tree ST_i of $\mathsf{X}_i^\tau\$$, where \$ is a unique special character. Since $|\mathsf{X}_i^\tau| \leq \lceil n/\tau \rceil$, the number of nodes in ST_i is at most $2\lceil n/\tau \rceil$. Append X_i' to the label (ignoring \$) on each edge from a leaf to its parent. We remark that the edge labels are not explicitly written, but are obtained using two pointers to the start and end positions of the label in X. Each non-root node u in ST_i has a *suffix link* pointing to a node $\Psi(u)$, such that the string (over Σ) obtained by concatenating the edge labels from root to $\Psi(u)$ is same as the string from root to u with the first τ characters truncated. By using the algorithm of Farach-Colton [8], constructing ST_i along with the suffix links requires $\mathcal{O}(n/\tau)$ time and space. Now, consider the set E_u of outgoing edges of a node u. We will order them from left-to-right according to the lexicographic order of the τ-length substring of X represented by the first character. Since the lexicographic rank of the τ-length strings can be compared directly in $\mathcal{O}(1)$ time based on its leaf index in $\mathcal{T}$ (i.e., based on its representative in Σ_i^τ), we can order the edges in all such sets E_u in $\sum_u \mathcal{O}(|E_u| \log |E_u|) = \mathcal{O}((n/\tau) \log \frac{n}{\tau})$ time using $\mathcal{O}(n/\tau)$ space. Each leaf in ST_i corresponds to a suffix of X with starting position in $\mathcal{I}_i$, where leaves are numbered from left-to-right in lexicographic order of the suffix they represent. For the pth leftmost leaf, denoted by ℓ_p, let $\mathsf{SA}_i[p]$ be the suffix array value, i.e., the starting position in $\mathcal{I}_i$ of the corresponding suffix. Summarizing, the time needed to construct ST_i is $\mathcal{O}(n + \frac{n}{\tau} \log \frac{n}{\tau})$, and the space usage is $\mathcal{O}(n/\tau)$ words. We create a compacted trie $\mathsf{ST}_i(u)$ with the edges in E_u by mapping the edge labels over Σ_i^τ to the corresponding τ-length string over Σ. Call this the *navigation trie of node u*. Note that each leaf in $\mathsf{ST}_i(u)$ corresponds to a unique child of u in ST_i. As before, the edge labels are obtained using two pointers to X. The outgoing edges of a node in the navigation trie are ordered based on the lexicographic order of the first character from Σ, such that given a character $x \in \Sigma$, we can find the outgoing edge (if any) beginning with x in $\mathcal{O}(\log \frac{n}{\tau})$ time. The number of nodes in all navigation tries combined is at most $2\lceil n/\tau \rceil$. Since the first τ-length strings labeling the outgoing edges of a node are distinct, all navigation tries can be created in $\mathcal{O}(n + \frac{n}{\tau} \log \frac{n}{\tau})$ time.

Equip ST_i with the data structures in [3, 4], such that in $\mathcal{O}(1)$ time, we can (i) find $\mathsf{lca}(u, v)$ i.e., the Lowest Common Ancestor (LCA) of two nodes u and v, and (ii) find $\mathsf{levelAncestor}(u, W)$ i.e., the ancestor of u which has node-depth W. Likewise, equip all navigation tries with these data structures. Using these, given two leaves ℓ_k and $\ell_{k'}$ in ST_i, we can easily find their LCA in a particular navigation trie in $\mathcal{O}(1)$ time. For any node u in ST_i, let $\mathsf{path}(u)$ denote the string formed by concatenating the edge labels over Σ from root to u. Likewise, for any node u^* in a navigation trie $\mathsf{ST}_i(u)$, let $\mathsf{path}(u^*)$ be the string $\mathsf{path}(u)$ appended with the edge labels from u to u^*. Store $|\mathsf{path}(u)|$ (resp. $|\mathsf{path}(u^*)|$) at each node u in ST_i (resp. u^* in $\mathsf{ST}_i(u)$). The space and time required for these pre-processing steps can both be bounded by $\mathcal{O}(n/\tau)$. With the aid of these pre-processing steps, in $\mathcal{O}(1)$ time, we can find $\mathsf{lcp}(\ell_k, \ell_{k'})$ i.e., the Longest Common Prefix (LCP) of $\mathsf{path}(\ell_k)$ and $\mathsf{path}(\ell_{k'})$.

Query: Initially, each node u^* in every navigation trie is *unmarked*; also, assign $\Delta(u^*) = 0$. Starting with $j = 1$ (if $i > 1$, and with $j = 2$ otherwise), we match successive symbols of the string $\mathsf{X}[j, n]$ in ST_i as follows. Suppose, we are at a node u in ST_i. Find the correct edge (if any) in $\mathsf{ST}_i(u)$ to traverse using the character $\mathsf{X}[j + |\mathsf{path}(u)|]$. Now, use the characters starting from $\mathsf{X}[j + |\mathsf{path}(u)| + 1]$ to traverse $\mathsf{ST}_i(u)$ until either we reach a leaf ℓ^* (which corresponds to a child u' of u in ST_i), or we find a mismatch. In the first case, mark the leaf ℓ^*, set $\Delta(\ell^*) = |\mathsf{path}(\ell^*)|$, and repeat the process from u'. Otherwise, suppose we find a failure on an edge to a node v^* in $\mathsf{ST}_i(u)$ after successfully matching D characters starting from u. *Mark* the node v^*, and store $\Delta(v^*) = \max\{\Delta(v^*), |\mathsf{path}(u)| + D\}$. Follow the suffix link of u to the node $\Psi(u)$. We have the following two cases to consider.

- If $D < \tau$, then use the string $\mathsf{X}[j+|\mathsf{path}(u)|, j+|\mathsf{path}(u)|+D-1]$ and traverse $\mathsf{ST}_i(\Psi(u))$. Then, we resume matching from the reached position using $\mathsf{X}[j + |\mathsf{path}(u)| + D, n]$.
- If $D \geq \tau$, then v^* is a leaf in $\mathsf{ST}_i(u)$ and represents a child v of u in ST_i. Use the string $\mathsf{X}[j + |\mathsf{path}(u)|, j + |\mathsf{path}(u)| + \tau - 1]$ and traverse $\mathsf{ST}_i(\Psi(u))$. At this point, we are on an edge from a node w^* to a leaf node in $\mathsf{ST}_i(\Psi(u))$. The desired position to resume matching on this edge is given by $(D - |\mathsf{path}(w^*)| + 1)$.

In either case, we compare at most τ characters. Observe that on following a suffix link we truncate τ characters starting from j, and are now trying to match $\mathsf{X}[j+\tau, n]$. Therefore, the total time needed to mark a node for $j, j+\tau, j+2\tau, \ldots$ is $\mathcal{O}(n \log \frac{n}{\tau})$. Repeat this process for every value of $j \in [1, \tau]$, $j \neq i$. The total time needed is $\mathcal{O}(n\tau \log \frac{n}{\tau})$.

We initialize an array LS of length $|\mathcal{I}_i|$ as follows. Assign $\mathsf{LS}[1] = \mathsf{lcp}(\ell_1, \ell_2)$, and $\mathsf{LS}[p] = \max\{\mathsf{lcp}(\ell_{p-1}, \ell_p), \mathsf{lcp}(\ell_p, \ell_{p+1})\}$, where $p \in [2, |\mathcal{I}_i| - 1]$. Finally, $\mathsf{LS}[|\mathcal{I}_i|] = \mathsf{lcp}(\ell_{|\mathcal{I}_i|-1}, \ell_{|\mathcal{I}_i|})$. Now, for each leaf ℓ_p in ST_i, we find its nearest marked ancestor ℓ_p^*. This is easily achieved in $\mathcal{O}(n/\tau)$ time and space by traversing ST_i and the navigation tries using lca and $\mathsf{levelAncestor}$ queries. Simply assign $\mathsf{LS}[p] = 1 + \max\{\mathsf{LS}[p], \Delta(\ell_p^*)\}$. If $\mathsf{LS}[p] = |\mathsf{path}(\ell_p)|$, then assign $\mathsf{LS}[p] = n + 1$. (This implies that LS value for the position $\mathsf{SA}_i[p]$ is not defined.) Clearly, $\mathsf{LS}[p]$ and $\mathsf{SA}_i[p]$ together give us $\mathsf{LS}_{\mathsf{SA}_i[p]}$. The time needed is $\mathcal{O}(n/\tau)$.

3 Deterministic Algorithm for All k

▶ **Observation 8** ([13, 14]). *$|\mathsf{LS}_k| \leq |\mathsf{LS}_{k+1}| + 1$, $k \in [1, n-1]$, where LS_k and LS_{k+1} are defined. S_1 is the same as LS_1. For any $k \in [2, n]$, if S_k is the right extension of some $\mathsf{LS}_{k'}$, $k' < k$, then* (i) *S_{k-1} ends at the position $(k-1)$, and* (ii) *$\mathsf{S}_k = \mathsf{S}_{k-1} \circ \mathsf{X}[k]$.*

Assume that S_{k-1}, $k > 1$, is computed, where $\mathsf{S}_1 = \mathsf{LS}_1$ is known. We want to compute S_k. The following are immediate from the above observation. If S_{k-1} does not end at $(k-1)$, then S_k is simply the shortest $\mathsf{LS}_{k'}$ that covers k. (Note that such an $\mathsf{LS}_{k'}$ must exist.) Otherwise, S_{k-1} ends at $(k-1)$, and S_k is simply the shorter of: (i) the shortest $\mathsf{LS}_{k'}$, $k' \leq k$, that covers k, if such a string exists, and (ii) $\mathsf{S}_{k-1} \circ \mathsf{X}[k]$. Thus the focus is to compute the shortest $\mathsf{LS}_{k'}$ that covers k, if such a string exists. We prove the following theorem.

▶ **Theorem 9.** *We can compute S_k for every $k \in [1, n]$ in $\mathcal{O}(n\tau^2 \log n)$ time using X and additional $\mathcal{O}(n/\tau)$ words and $4n + o(n)$ bits of working space.*

Following are a couple of well-known results that will be needed.

▶ **Fact 10** (Munro [18]). *Consider a binary string $B[1, m]$. By using a data structure occupying $o(m)$ bits, in $\mathcal{O}(1)$ time, we can find* (i) $\mathsf{rank}(i, c) = |\{j \leq i \mid B[j] = c\}|$ *and* (ii) $\mathsf{select}(j, c) = \min_i\{i \mid \mathsf{rank}(i, c) = j\}$, *where $c \in \{0, 1\}$. The data structure can be constructed in $\mathcal{O}(m)$ time using $o(m)$ bits of working space in addition to the string B.*

▶ **Fact 11** (Fischer and Heun [9]). *Consider an array A of m integers. By using a data structure occupying $2m + o(m)$ bits, in $\mathcal{O}(1)$ time, we can find $\mathsf{rmq}_A(i, j)$ i.e., a position $t \in [i, j]$ such that $A[t] = \min\{A[t'] \mid t' \in [i, j]\}$. The data structure can be constructed in $\mathcal{O}(m)$ time using $2m + o(m)$ bits of working space in addition to the array A.*

Proof of Theorem 9. The key idea is to compute LS_j for all values of j, where defined, and then store it in a compact way. Specifically, use Lemma 6 to compute LS_j for every $j \in \mathcal{I}_i = \{i, i + \tau, i + 2\tau, \dots\}$, first by choosing $i = 1$. Store these values explicitly, and initialize $|\mathcal{I}_1|$ empty binary strings $B_1, B_2, \dots, B_{|\mathcal{I}_1|}$. Compute LS_j for every $j \in \mathcal{I}_2$. For each $j \in \mathcal{I}_2$, append $(|\mathsf{LS}_j| + 1 - |\mathsf{LS}_{j-1}|)$ many 1s followed by a 0 to the binary string $B_{\lceil j/\tau\rceil}$. (Note that $(j - 1) \in \mathcal{I}_1$, and LS_{j-1} has already been computed.) Now, compute LS_j for every $j \in \mathcal{I}_3$. For each $j \in \mathcal{I}_3$, append $(|\mathsf{LS}_j| + 1 - |\mathsf{LS}_{j-1}|)$ many 1s followed by a 0 to the binary string $B_{\lceil j/\tau\rceil}$. Delete the LS_j values computed for $j \in \mathcal{I}_2$. Repeat the process with $i = 4, 5, \dots, \tau$. Suppose r is the last position such that LS_r is defined. We will ignore $\mathsf{LS}_{r+1}, \mathsf{LS}_{r+2}, \dots, \mathsf{LS}_n$ while creating the binary strings. Now, we create a binary string $B = B_1B'_1B_2B'_2 \dots B'_{|\mathcal{I}_1|-1}B_{|\mathcal{I}_1|}$, where B'_p, $p \in [1, |\mathcal{I}_1| - 1]$, is the string containing $(\mathsf{LS}_{p\tau+1} + 1 - \mathsf{LS}_{p\tau})$ many 1s followed by a 0. Delete the binary strings B_1 through $B_{|\mathcal{I}_1|}$. If $r > n - \tau + 1$, compute LS_j for $j \in [n - \tau + 1, r]$ in $O(n\tau^2)$ time using a brute-force approach. For each $j \in [n - \tau + 2, r]$, append $(|\mathsf{LS}_j| + 1 - |\mathsf{LS}_{j-1}|)$ many 1s followed by a 0 to the binary string B. Finally, construct the rank-select structure of Fact 10 over B.

Since we will make τ calls to Lemma 6, the time required is $\mathcal{O}(n\tau^2 \log \frac{n}{\tau})$. Note that $r + |\mathsf{LS}_r| \leq n + 1$. By Observation 8, for any $r' < r$, we have $r' + |\mathsf{LS}_{r'}| \leq r' + 1 + |\mathsf{LS}_{r'+1}|$. By Observation 5, $\mathsf{LS}_{r''}$ is not defined for any $r'' > r$. It immediately follows that $\sum_{p=1}^{r-1}(|\mathsf{LS}_{p+1}| + 1 - |\mathsf{LS}_p|) \leq n$. Observe that B is a binary string which is a concatenation of $(|\mathsf{LS}_{p+1}| + 1 - |\mathsf{LS}_p|)$ many 1s followed by 0 for all values of p from 1 to $(r - 1)$. Therefore, the total length of B is at most $2n$. Then, $|\mathsf{LS}_k| = |\mathsf{LS}_1| + \mathsf{rank}(\mathsf{select}(k - 1, 0), 1) - k + 1$, where $k \in [1, r]$. By storing $|\mathsf{LS}_1|$ and the position r explicitly in $\lceil 2 \log n\rceil = o(n)$ bits, LS_k can be retrieved in $\mathcal{O}(1)$ time. Since at any point we are storing LS_j values for at most $3\lceil n/\tau\rceil$ choices of j, the working space needed is $\mathcal{O}(n/\tau)$ words and $2n + o(n)$ bits.

Now, we build an RMQ data structure over a conceptual array A. The length of A is the number of zeroes in B i.e., $|A| = r \leq n$, and $A[p] = |\mathsf{LS}_p| = |\mathsf{LS}_1| + \mathsf{rank}(\mathsf{select}(p-1, 0), 1) - p + 1$. Using Fact 11, we can construct this data structure using $2n + o(n)$ bits of additional space.

Summarizing, the working space needed at any point is $\mathcal{O}(n/\tau)$ words and $4n + o(n)$ bits.

We return to the task of computing the shortest $\mathsf{LS}_{k'}$, $k' \leq k$, that covers k. First locate the smallest position $k'' \leq k$, such that $\mathsf{LS}_{k''}$ covers k. This is achieved in $\mathcal{O}(\log n)$ time via a binary search using LS_1, B and its associated rank-select structure. If k'' does not exist, then we are done. Otherwise, $k' = \mathsf{rmq}_A(k'', k)$. The total time needed for all such computations is $\mathcal{O}(n \log n)$, and the claimed space and time bounds are met. ◀

▶ **Corollary 12.** *Suppose, we can compute $\mathsf{LS}_1, \mathsf{LS}_2, \dots, \mathsf{LS}_n$ in that order. We can store $|\mathsf{LS}_k|$, $k \in [1, n]$, in total $2n + o(n)$ bits, such that a particular $|\mathsf{LS}_k|$ value can be accessed in $\mathcal{O}(1)$ time.* (This is the same result as obtained by Belazzougui and Cunial [2].) *Also, by maintaining an additional $2n + o(n)$-bit structure, for any k, in $\mathcal{O}(\log n)$ time, we can compute the shortest $\mathsf{LS}_{k'}$, $k' \leq k$, that covers k, or verify that no such k' exists. The total time (in addition to that for computing every LS_k value) to construct this $4n + o(n)$ data* *structure is $\mathcal{O}(n)$. The working space required is $4n + o(n)$ bits. Using this, we can compute S_k for every $k \in [1, n]$ in an additional $\mathcal{O}(n \log n)$ time and $\mathcal{O}(1)$ space.*

4 Randomized Algorithm

We prove the following theorem in this section.

▶ **Theorem 13.** *For a string X of length n, any given $k \in [1, n]$, and any arbitrary constant $c \geq 0$, we can find S_k in $\mathcal{O}(n\tau \log^{c+1} n)$ time using X and additional $n/\log^c n + \mathcal{O}(n/\tau)$-words of working space. By using additional $4n + o(n)$ bits, we can compute S_k for all values of k in $\mathcal{O}(n\tau \log^{c+1} n)$ time. Each S_k computed is correct with probability at least $1 - n^{-\mathcal{O}(1)}$.*

4.1 Proof of Theorem 13

The key idea to reduce the time from $\mathcal{O}(n\tau^2 \log \frac{n}{\tau})$ is to modify Lemma 6 so that we can carry out the same task in time $\mathcal{O}(n \log^{c+1} n)$ time, with $n/\log^c n$ words of additional space. In this context, we present the following lemma.

▶ **Lemma 14.** *Let $\mathcal{I}_i = \{i, i + \tau, i + 2\tau, \ldots\}$, $i \in [1, \tau]$, be a set of at most $\lceil n/\tau \rceil$ suffixes. For each $i' \in \mathcal{I}_i$, we can compute $\mathsf{LS}_{i'}$ correctly with high probability in $\mathcal{O}(n \log^{c+1} n)$ time using X and additional $n/\log^c n + \mathcal{O}(n/\tau)$-words of working space.*

Here and henceforth, by high probability, we mean that each computed $\mathsf{LS}_{i'}$ is unique, but with probability at most $n^{-\mathcal{O}(1)}$, may not be the shortest. Likewise, each computed S_k is unique and covers k, but with probability at most $n^{-\mathcal{O}(1)}$, may not be the shortest.

We observe that in Lemma 6 the $n\tau$-factor in the time complexity is due to matching X in the sampled suffix tree ST_i by passing the string τ times, each time with a different choice of $j \in [1, \tau]$, $j \neq i$. Each such pass costs us $\mathcal{O}(n \log \frac{n}{\tau})$ time. The idea is to reduce this by speeding up (i) the time to find the correct outgoing edge of a node, and (ii) the time to update the Δ value of a node in a navigation trie. We will show that (i) can be achieved in $\mathcal{O}(\log^c n)$ time, with a slight probability of a false positive using Rabin-Karp Fingerprint [15] and perfect hashing [10]. For achieving (ii), the rough idea is to use randomization to binary search on the navigation trie, along the path containing the longest repeating prefix. This will cost us $\mathcal{O}(\log^{c+1} n)$ time. We begin by revisiting a couple of important results.

▶ **Fact 15** (Rabin-Karp Fingerprint [15])**.** *Let S be a string, and $p > |S|$ be a prime number. Choose $q \in \mathbb{F}_p$ uniformly at random. The fingerprint of S is*

$$\Phi(S) = \sum_{k=0}^{|S|-1} S[k]q^k \mod p$$

The following are a few well-known properties of fingerprints [5]. The probability of $\Phi(S) = \Phi(S')$ for two distinct strings S and S' is at most $m^{-\lambda+1}$, where $m = |S| = |S'|$, $p \in \Theta(m^\lambda)$, and $\lambda \geq 4$ is a constant. The factor λ may be amplified by a constant number of computations. For two strings S and S', where $m = |S|$, we have (i) $\Phi(SS') = \Phi(S) + \Phi(S')q^m \mod p$, (ii) $\Phi(S) = \Phi(SS') - \Phi(S')q^m \mod p$, and (iii) $\Phi(S') = \big(\Phi(SS') - \Phi(S)\big)q^{-m} \mod p$. Therefore, for these three equations, given the value of $q^m \mod p$ and the FP values on right, we compute the FP value on the left in $\mathcal{O}(1)$ time.

▶ **Fact 16** (Probabilistic z-fast Trie, Theorem 4.1, Belazzougui et al. [1])**.** *Consider the compacted trie $\mathcal{T}$ of a set of t strings. Each string has length at most m. Given any string S,* *by using a probabilistic data structure, we can find the deepest node u (called the **exit node**) in $\mathcal{T}$ such that* path(u) *is a prefix of S. The chances of an error is at most $m^{-\lambda}$, where $\lambda > 0$ is an arbitrary constant. The space occupied by the probabilistic data structure is $\mathcal{O}(t)$ words, and the time required is $\mathcal{O}(\log(m+t))$.*

The main technique behind Fact 16 is to associate each node u in the trie with a signature function. Specifically, the signature function is based on path(u). If two strings are distinct, then their signatures match with very low probability. Now, given the signature of each prefix of S, the overall idea is to carry out a binary search on the signature of each node to locate the desired node. Furthermore, given the compacted trie, and the signature of every prefix of each of the t strings, the data structure of Fact 16 can be constructed in $\mathcal{O}(t)$ time.

▶ **Lemma 17.** *Consider the compacted trie $\mathcal{T}$ of a set of t suffixes of a string Y having length m. Given a string S and fingerprint of every $\log^c m$ prefix, by maintaining an $m/\log^c m+\mathcal{O}(t)$ word data structure, we can find the deepest point (possibly, on an edge) such that the string formed by concatenating edge labels from root to this point is a prefix of S. The time required is $\mathcal{O}(\log^{c+1} m)$, and the probability of an error is at most $m^{-\mathcal{O}(1)}$. The data structure can be constructed in $\mathcal{O}(m + t(\log t + \log^c m))$ time using $m/\log^c m + \mathcal{O}(t)$ words of space.*

Proof. We will use a different signature function as that of Belazzougui et al. [1]. (See [5] for a similar usage.) Specifically, each node w is labeled with the fingerprint (FP) of path(w). Each edge in $\mathcal{T}$ is labeled by a substring of Y. We maintain two pointers *sp* and *ep* to the start and end position in Y, and store the value of $q^{sp} \mod p$. Here, p and q are defined as in Fact 15. Also, at each node w, we store the value of $q^{|\mathsf{path}(w)|} \mod p$. To compute this simply sort the edges based on *sp* and the nodes w based on path(w) in $O(t \log t)$ time. The time needed is $\mathcal{O}(m + t\log t)$, and the space occupied at any point is $\mathcal{O}(t)$ words.

Now, compute the FP of each of the prefixes of Y ending at the positions $1, 1+\log^c m, 1+2\log^c m, \dots$ in $\mathcal{O}(m)$ time using Fact 15. The space needed to compute and store this information is at most $(1 + m/\log^c m)$ words. Using these, we can compute the FP of a prefix of an edge label in $\mathcal{O}(\log^c m)$ time by simply finding the nearest prefix of Y whose FP has been stored and then walking at most $\log^c m$ characters in Y. Also, by pre-processing the trie with levelAncestor queries [4], we can find the FP of a prefix of any of the t suffixes in $\mathcal{O}(\log t + \log^c m)$ time as follows. Binary search using levelAncestor queries and $|\mathsf{path}(u)|$ stored at each node u on the path from root to the leaf corresponding to the suffix. This binary search enables us to find the edge position corresponding to the prefix whose FP value we want to find. Then, the desired FP value is obtained using the edge pointers. Therefore, we can construct the z-fast trie of Fact 16 in $\mathcal{O}(t(\log t + \log^c m))$ time given $\mathcal{T}$ and the FP of each prefix of Y. The space at any point is bounded by $m/\log^c m + \mathcal{O}(t)$ words.

We return to our original task. Use the z-fast trie and the FP of each prefix of S to find the exit node u. Then, use the character $S[1 + |\mathsf{path}(u)|]$ to select an outgoing edge (u, v) of u. If no such edge exists, then the desired location is given by the node u. Otherwise, the desired location lies on the edge (u, v). Using the FP of a prefix of $S[|\mathsf{path}(u)| + 1, \mathsf{path}(v)]$, we binary search on the edge (u, v) to find the desired location. Each prefix computation needs $\mathcal{O}(\log^c m)$ time. Thus, the total time required is $\mathcal{O}(\log^{c+1} m)$. Since the number of FP comparisons is $\mathcal{O}(\log m)$, the probability of a false positive is $\mathcal{O}(\frac{\log m}{m^\lambda}) = m^{-\mathcal{O}(1)}$. ◀

Proof of Lemma 14. Construct the suffix tree ST_i for each suffix starting at a location lying in the set $\mathcal{I}_i = \{i, i+\tau, i+2\tau, \dots\}$. Now, create the navigation trie $\mathsf{ST}_i(u)$ of every node u. The total time needed is $\mathcal{O}(n + \frac{n}{\tau}\log\frac{n}{\tau})$. Each edge in ST_i or a navigation trie has pointers *sp* and *ep* to X. Use Lemma 17 to compute and store (i) $q^{|\mathsf{path}(w)|} \mod p$ for each node w in

ST_i, and (ii) $q^{sp} \mod p$ for each edge. Likewise, we compute and store the values for each node and edge in every navigation trie. We maintain an array A_j which stores the values $q^j \mod p$, $q^{j+\tau} \mod p, q^{j+2\tau} \mod p, \dots$. Initially, the array is maintained for $j = 2$ if $i = 1$, and for $j = 1$, otherwise. As described in Lemma 17, the total time needed is $\mathcal{O}(n)$. The construction space and that needed for storage are both bounded by $\mathcal{O}(n/\tau)$ words.

Using Fact 15, we compute and store $\Phi(\mathsf{X}[1, i])$ for every $i \in \{1, 1+\log^c n, 1+2\log^c n, \dots\}$ in $\mathcal{O}(n)$ time. The space needed is $1 + n/\log^c n$ words. Compute the FP of the first τ-characters of the edge label in ST_i; this is achieved in $\mathcal{O}(\log^c n)$ time. Use a perfect hash function [10] at each node for selecting the correct outgoing edge based on the computed FP. The total time and space needed to incorporate this information is $\mathcal{O}((n/\tau)\log^c n)$. Finally, we maintain the probabilistic z-fast trie of Fact 16 for each navigation trie. This can be created as described in Lemma 17. Incorporating the probabilistic data structure for all navigation tries requires $\mathcal{O}(n + \frac{n}{\tau}(\log \frac{n}{\tau} + \log^c n))$ time.

Summarizing, the space needed to maintain the data structure comprising of ST_i, the navigation tries and their adjoining z-fast tries is $n/\log^c n + \mathcal{O}(n/\tau)$ words. Moreover, the data structure is constructed in $\mathcal{O}(n + \frac{n}{\tau}(\log \frac{n}{\tau} + \log^c n))$ time using $n/\log^c n + \mathcal{O}(n/\tau)$ words.

Now, we start matching X in ST_i starting with $j = 1$ if $i > 1$, and with $j = 2$, otherwise. To traverse ST_i, suppose we are at a node u, and have read up to position j' in X. Use $\Phi(\mathsf{X}[j' + 1, j' + \tau])$ to select a correct outgoing edge (if any). This is achieved in $\mathcal{O}(\log^c n)$ time first by computing the FP using the array A_j, and then using perfect hashing. Similarly, we can traverse every τ characters on an edge in ST_i in $\mathcal{O}(\log^c n)$ time. We do this until we find a failure, or reach a child v of u. In the latter case, we mark v's corresponding leaf ℓ^* in $\mathsf{ST}_i(u)$, update $\Delta(\ell^*)$, and continue matching from v. In the former case, use Lemma 17 to mark the correct node w^* in $\mathsf{ST}_i(u)$ and update $\Delta(w^*)$ in $\mathcal{O}(\log^{c+1} n)$ time. Now, follow the suffix link of u. The correct position to start matching in an outgoing edge of $\Psi(u)$ in ST_i can be found in $\mathcal{O}(\log^c n)$ time. Continue, until the entire string X has been processed. The total time needed is $\mathcal{O}(\frac{n}{\tau}\log^{c+1} n)$. Now, we repeat the process with $(j + 1)$ if $i \neq (j + 1)$, and with $(j + 2)$, otherwise. The array A_j can be updated in $\mathcal{O}(n/\tau)$ time to A_{j+1} or A_{j+2}, as the case is. The number of times this process is repeated is $(\tau - 1)$. Finally, for each $i' \in \mathcal{I}_i$, we can compute $\mathsf{LS}_{i'}$ as described in Section 2.1 in $\mathcal{O}(n/\tau)$ time. Hence, the total time needed is $\mathcal{O}(n\log^{c+1} n + \frac{n}{\tau}(\log \frac{n}{\tau} + \log^c n)) = \mathcal{O}(n\log^{c+1} n)$.

Note that if two strings are identical, then their FP values are necessarily the same. Hence, each $\mathsf{LS}_{i'}$ is definitely unique, but may not be the shortest. The number of queries to a z-fast trie, or a FP comparison are both bounded by $\mathcal{O}(n\log n)$. Therefore, the probability of an error is $n^{-\mathcal{O}(1)}$ (achieved by appropriately choosing λ in Facts 15 and 16). ◀

Wrapping Up. As in Theorem 7, we will invoke Lemma 14 by rotating the choices of $i \in [1, \tau]$. Finally, we compute the LS_j values for $j \in [n - \tau + 2, n]$ as follows. Maintain the FP of every $\log^c n$ prefix of $\mathsf{X}[n - \tau + 2, n]$. Now to find LS_j, binary search at each position (other than j) of X with the suffix starting at j to find the longest repeating prefix. The FP of a prefix of any of these suffixes (resp. of a suffix in X) is obtained in $\mathcal{O}(\log^c \tau)$ time (resp. $\mathcal{O}(\log^c n)$ time). The number of binary search operations is $\mathcal{O}(\log \tau)$. Thus, the overall time is bounded by $\mathcal{O}(n\tau(\log^c n)\log \tau) = \mathcal{O}(n\tau\log^{c+1} n)$.

The discussion in this section and the techniques used in proving Theorems 7 prove the first part of Theorem 13 for computing S_k for a single k. The latter part of the theorem is a consequence of Corollary 12, which follows from the proof of Theorem 9. However, one needs to be a little more careful while carrying out the steps in Theorem 9 because the relation $|\mathsf{LS}_i| \leq |\mathsf{LS}_{i+1}| + 1$ in Observation 8 maybe violated due to false positives in FP matches.

Since no false negatives occur in FP matches, each computed $\mathsf{LS}_{i'}$ for any i' is definitely unique. Therefore, we can simply start from the rightmost i' where $|\mathsf{LS}_{i'}| \leq |\mathsf{LS}_{i'+1}| + 1$ is violated and set $|\mathsf{LS}_{i'}| = |\mathsf{LS}_{i'+1}| + 1$ for each successive i' from right to left. Observe that the total number of changes to the binary strings B_1 through $B_{|\mathcal{I}_1|}$ (see the proof of Theorem 9) is at most $2n$ for each invoking of Lemma 14. Therefore, the total time needed to affect these changes is $\mathcal{O}(n\tau)$. Finally, the binary string B is again computed in $\mathcal{O}(n)$ time. The rest of the steps remain the same as in the proof of Theorem 9. This completes the proof of Theorem 13.

5 Maximal Unique Matches Problem

Let X_1 and X_2 be two strings of length n_1 and n_2 respectively, where $n = n_1 + n_2$. Each character is drawn from a totally ordered alphabet Σ. We assume that X_1 and X_2 terminate in two special characters $\$_1$ and $\$_2$ that does not appear anywhere else.

▶ **Definition 18** (Maximal Unique Match)**.** A *Maximal Unique Match* (MUM) of two strings X_1 and X_2 is a string S that satisfies the following two properties: (i) S appears uniquely in each string X_1 and X_2, and (ii) a left or right extension of S in X_1 does not appear in X_2.

▶ **Problem 19.** *Given two strings X_1 and X_2, the task is to find the set $\mathcal{S}$ of all their maximal unique matches. Each match is represented by its starting position in X_1 and its length.*

To the best of our knowledge, Problem 19 was formulated by Delcher et al. [6]. The main motivation was its importance in aligning whole genome sequences consisting of millions of nucleotides. They presented a software known as MUMmer 1.0. Further improvements by Delcher et al. [7] and then by Kurtz et al. [17] lead to MUMmer 2.0 and MUMmer 3.0 respectively. The chief component of all these softwares (and underlying algorithm) is the (generalized) suffix trees (GST) – a compacted trie storing all the suffixes of X_1 and X_2, and occupying $\Theta(n)$ words. The following is the key observation.

▶ **Observation 20.** *Given two strings X_1 and X_2 and their GST, a string S is an MUM iff*

(a) *There exists a node v in the GST such that $S = \mathsf{path}(v)$. Moreover, v has exactly two children (leaves), each labeled by a suffix from X_1 and X_2.*

(b) *There does not exist a node u which simultaneously satisfies:* (i) *u has a suffix link to v, and* (ii) *u has exactly two children (leaves) that are labeled by suffixes from X_1 and X_2.*

The GST of X_1 and X_2 can be built in $\mathcal{O}(n)$ time using the algorithm of Farach-Colton [8], and leads to a simple $\mathcal{O}(n)$-space and $\mathcal{O}(n)$-time algorithm for Problem 19. The basic idea to reduce the space is to build a GST only on n_1/τ suffixes of X_1 and n_2/τ suffixes of X_2 at a time. This reduces the space to $\mathcal{O}(n/\tau)$ words. By rotating the choice of n_2/τ suffixes in X_2 roughly τ times, we will be able to determine the candidate set (i.e., a set containing the MUMs) among the n_1/τ suffixes of X_1. Using the next set of n_1/τ suffixes of X_1, we will be able to remove the incorrect choices from the candidate set. This idea, coupled with the techniques for the SUS problem, leads to the following theorem.

▶ **Theorem 21.** *Given X_1 and X_2, we can compute the set $\mathcal{S}$* (i) *in $\mathcal{O}(n\tau^2 \log \frac{n}{\tau})$ time using additional $\mathcal{O}(n/\tau)$ words of working space, and* (ii) *correctly with high probability in $\mathcal{O}(n\tau \log^{c+1} n)$ time using additional $n/\log^c n + \mathcal{O}(n/\tau)$ words of working space.*

References

1 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Monotone minimal perfect hashing: searching a sorted table with $O(1)$ accesses. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 785–794, 2009.

2 Djamal Belazzougui and Fabio Cunial. Indexed matching statistics and shortest unique substrings. In *String Processing and Information Retrieval – 21st International Symposium, SPIRE 2014, Ouro Preto, Brazil, October 20-22, 2014. Proceedings*, pages 179–190, 2014. doi:10.1007/978-3-319-11918-2_18.

3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, pages 88–94, 2000. doi:10.1007/10719839_9.

4 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. In *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, pages 508–515, 2002. doi:10.1007/3-540-45995-2_44.

5 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3_31.

6 Arthur L. Delcher, Simon Kasif, Robert D Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic acids research*, 27(11):2369–2376, 1999.

7 Arthur L. Delcher, Adam Phillippy, Jane Carlton, and Steven L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic acids research*, 30(11):2478–2483, 2002.

8 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS'97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

9 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, pages 459–470, 2007. doi:10.1007/978-3-540-74450-4_41.

10 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

11 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997.

12 Wing-Kai Hon, Tak Wah Lam, Rahul Shah, Siu-Lung Tam, and Jeffrey Scott Vitter. Compressed index for dictionary matching. In *2008 Data Compression Conference (DCC 2008), 25-27 March 2008, Snowbird, UT, USA*, pages 23–32, 2008. doi:10.1109/DCC.2008.62.

13 Wing-Kai Hon, Sharma V. Thankachan, and Bojian Xu. An in-place framework for exact and approximate shortest unique substring queries. In *Algorithms and Computation – 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 755–767, 2015. doi:10.1007/978-3-662-48971-0_63.

14 Atalay Mert Ileri, M. Oguzhan Külekci, and Bojian Xu. Shortest unique substring query revisited. In *Combinatorial Pattern Matching – 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, pages 172–181, 2014. doi:10.1007/978-3-319-07566-2_18.

15 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

16 Stefan Kurtz. Reducing the space requirement of suffix trees. *Softw., Pract. Exper.*, 29(13):1149–1171, 1999. doi:10.1002/(SICI)1097-024X(199911)29:13<1149::AID-SPE274>3.0.CO;2-O.

17 Stefan Kurtz, Adam Phillippy, Arthur L. Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L. Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004.

18 J. Ian Munro. Tables. In *Foundations of Software Technology and Theoretical Computer Science, 16th Conference, Hyderabad, India, December 18-20, 1996, Proceedings*, pages 37–42, 1996. doi:10.1007/3-540-62034-6_35.

19 Jian Pei, Wush Chi-Hsuan Wu, and Mi-Yen Yeh. On shortest unique substring queries. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 937–948, 2013. doi:10.1109/ICDE.2013.6544887.

The Subset Assignment Problem for Data Placement in Caches

Shahram Ghandeharizadeh[1], **Sandy Irani**[2], **and Jenny Lam**[3]

1 **Department of Computer Science, University of Southern California, Los Angeles, CA, USA**
`shahram@dblab.usc.edu`

2 **Department of Computer Science, University of California, Irvine, CA, USA**
`irani@ics.uci.edu`

3 **Department of Computer Science, San José State University, San José, CA, USA**
`jenny.lam01@sjsu.edu`

Abstract

We introduce the subset assignment problem in which items of varying sizes are placed in a set of bins with limited capacity. Items can be replicated and placed in any subset of the bins. Each (item, subset) pair has an associated cost. Not assigning an item to any of the bins is not free in general and can potentially be the most expensive option. The goal is to minimize the total cost of assigning items to subsets without exceeding the bin capacities. This problem is motivated by the design of caching systems composed of banks of memory with varying cost/performance specifications. The ability to replicate a data item in more than one memory bank can benefit the overall performance of the system with a faster recovery time in the event of a memory failure. For this setting, the number n of data objects (items) is very large and the number d of memory banks (bins) is a small constant (on the order of 3 or 4). Therefore, the goal is to determine an optimal assignment in time that minimizes dependence on n. The integral version of this problem is NP-hard since it is a generalization of the knapsack problem. We focus on an efficient solution to the LP relaxation as the number of fractionally assigned items will be at most d. If the data objects are small with respect to the size of the memory banks, the effect of excluding the fractionally assigned data items from the cache will be small. We give an algorithm that solves the LP relaxation and runs in time $O(\binom{3^d}{d+1} \operatorname{poly}(d) n \log(n) \log(nC) \log(Z))$, where Z is the maximum item size and C the maximum storage cost.

1998 ACM Subject Classification G.2.1 [Combinatorics] Combinatorial algorithms

Keywords and phrases Memory management, caching, simplex method, linear programming, min-cost flow

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.35

1 Introduction

We define a combinatorial optimization problem which we call the *subset assignment problem*. An instance of this problem consists of n items of varying sizes and d bins of varying capacities. Any item can be replicated and assigned to multiple bins. A problem instance also includes $n \cdot 2^d$ cost parameters which denote for each item and each subset of the bins, the cost of storing copies of the item on that subset of the bins. The objective is to find an assignment of items to subsets of bins which minimizes the total cost subject to the constraint that the sum of the sizes of items assigned to each bin does not exceed the capacity of the bin.

The costs do not necessarily exhibit any special properties, although we do assume that they are non-negative. For example, we do not assume that cost necessarily increases or decreases the more bins an item is assigned to. Assigning an item to the empty set, which corresponds to not assigning the item to any of the bins, is not free in general and can potentially be the most expensive option for an item.

The subset assignment problem is a natural generalization of the multiple knapsack problem (MKP), in which each item can only be stored on a single bin. The book by Martello and Toth [17] and the more recent book by Kellerer et al. [14] both devote a chapter to MKP. A restricted version of the subset assignment problem in which each item can only be stored in a single bin corresponds to the multiple knapsack problem (MKP). MKP is known to be NP-complete but does have a polynomial time approximation scheme [5]. For the application we are interested in, the number of items n is very large (on the order of billions) and the number of bins is a small constant (on the order of 3 or 4). Furthermore, the size of even the largest item is small with respect to the capacity of the bins. Since there is an optimal solution for the linear programming relaxation in which at most d items are fractionally placed, the effect of excluding the fractionally assigned items from the cache is negligible. Therefore, we focus on an efficient solution to the linear programming relaxation.

The linear relaxation of MKP can be expressed as a minimum cost flow on a bipartite graph, a classic and well studied problem in the literature [1]. Tighter analysis for the case of minimum cost flow on an imbalanced bipartite graph ($n >> d$) is given in [11] and improved in [2]. Naturally, the goal with highly imbalanced bipartite graphs (which corresponds to the situation in the subset assignment problem in which the number of items is much larger than the number of bins) is to minimize dependence on n, even at the expense of greater dependence on d.

We propose an algorithm for the linear programming relaxation of the subset assignment problem that is similar in structure to cycle canceling algorithms for min-cost flow and is also inspired by the concept of a bipush, which is central to the tighter analysis [11] and [2]. The analysis shows that our algorithm runs in $O(f(d)\operatorname{poly}(d)n\log(n)\log(nC)\log(Z))$, where C is the maximum cost of storing an item on any subset of the bins and Z is the maximum size of any item. The full details of this analysis are available in [10]. The function $f(d)$ is defined to be the number of distinct sets of vectors $\{\vec{v}_1, \ldots, \vec{v}_r\}$ where $\vec{v}_i \in \{-1, 0, 1\}^d$ and the solution $\vec{\alpha}$ to the equation $\sum_{i=1}^{r} \alpha_i \vec{v}_i = \vec{0}$ with $\alpha_1 = 1$ is unique and positive ($\vec{\alpha} > 0$). In order for the solution $\vec{\alpha}$ to be unique, the first $r - 1$ vectors must be linearly independent and therefore $r \leq d + 1$. If $r < d + 1$, the set can be uniquely expanded to a set of size $d + 1$ such that the solution to $\sum_{i=1}^{r} \alpha_i \vec{v}_i = \vec{0}$ with $\alpha_1 = 1$ is unique and non-negative ($\vec{\alpha} \geq 0$). This observation gives an upper bound of $\binom{3^d}{d+1}$ for $f(d)$, hence the running time of $O(\binom{3^d}{d+1}\operatorname{poly}(d)n\log(n)\log(nC)\log(Z))$. Numerical simulation has shown that $f(3) = 778$ and $f(4) = 531,319$. Since the problem specification requires $n \cdot 2^d$ cost parameters, an exponential dependence on d is unavoidable. A direction for future research is to reduce the dependence on d from exponential in d^2 to exponential in d.

A reasonable assumption for the cache assignment problem (the motivating application for the subset assignment problem) is that it is never advantageous to replicate an item in more than two bins. Under this assumption, the vectors can have at most 4 non-zero entries and therefore the function $f(d)$ is bounded by $d^{O(d)}\operatorname{poly}(d)$ resulting in an overall running time of $O(d^{O(d)}\operatorname{poly}(d)n\log(n)\log(nC)\log(Z))$.

In a linear programming formulation of the problem there are $2^d n$ variables and $n + d$ constraints. The best polynomial time algorithms to solve a general instance of linear programming require time at least cubic in n which for the values we consider is prohibitively large. (For example, Karmarkar's algorithm requires $O(N^{3.5}L)$ operations in which N is the number of variables and all input numbers can be encoded with $O(L)$ digits [13].) Our algorithm is closer to the Simplex algorithm in that after each iteration, the current solution is a basic feasible solution. However, the algorithm does not necessarily traverse the edges of the simplex. The algorithm selects an optimal local improvement which, in general, can result in a solution which is not a basic feasible solution and then restore the solution to a basic feasible solution without increasing the cost. It is not clear how to bound the running time of any implementation of the Simplex algorithm in which the algorithm is bound to traverse edges of the simplex. Since the problem can be formulated as a packing problem, there is a randomized approximation algorithm whose solution is within a factor of $1+\epsilon$ of optimal and whose running time is $O(2^d \operatorname{poly}(d) n \log n/\epsilon^2)$ [16].

1.1 Motivation

The subset assignment problem is motivated by the problem of managing a multi-level cache. Although caches are used in many different contexts, we are particularly interested in the use of caches to augment database management systems. Query results (called *key-value pairs*) are stored in the cache so that the next time the query is issued, the result can be retrieved from memory instead of recomputed from scratch. Typically key-value pairs vary in size as they contain different types of data. Furthermore, re-computation can vary dramatically, depending on the application. In some applications, a key-value pair is the result of hours of data-intensive computation. If the key-value pair does not reside in the cache (corresponding to allocating the item to the empty set in our formulation), this computation cost would be paid every time the key-value pair is accessed. Memcached, currently the most popular key-value store manager, is used by companies such as Facebook [19], Twitter and Wikipedia. Today's memcached uses DRAM for fast storage and retrieval of key-value pairs. However, using a cache that consists of a collection of memory banks with different characteristics can potentially improve cost or performance [7].

We model a sequence of requests to data items (key-value pairs) in the cache as a stream of independent events as do social networking benchmarks such as BG [4] and LinkBench [3]. Cache management (or paging) under i.i.d. request sequences is also well studied in the theory literature [20, 12]. If query and update (read and write) statistics are known in advance, the optimal policy is a static placement of data items in the memory banks that minimizes the expected time to service each request. A static placement can have much better performance over adaptive online algorithms if the request frequencies are stable [9, 8]. Since the popularity of queries do vary over time, a static placement would need to be recomputed periodically based on recent statistics followed with a reorganization of key-value pairs across memory banks.

With the advent of Non-Volatile Memory (NVM) such as PCM, STT-RAM, NAND Flash, and the (soon to be released) Intel X-Point, cache designers are provided with a wider selection of memory types with different performance, cost and reliability characteristics. The relative read/write latency and bandwidth for different memory types vary considerably.

An important challenge in computer system design is how to effectively design caching middleware that leverages these new choices [15, 7, 18]. The survey in [18] makes the case that the advent of new storage technologies significantly changes the standard assumptions in system design and leveraging such technologies will require more sophisticated workload-aware storage tiering.

In this paper, the cache is composed of a small number of memory banks each of which is a different type of memory. The goal is to find an optimal placement of data items in the cache. Our model also takes into account that a memory bank can fail due to a power outage or hardware failure. (Non-volatile memories do not lose their content during a power outage but they can experience hardware failures). If a memory bank fails, its contents must be restored, either all at once or over time. In this case, it may be advantageous to store a data item on more than one memory bank so that the data can be more easily recovered in the event that one or more memory banks fail. On the other hand, maintaining multiple copies of a key-value pair can be costly if they must be frequently updated. We express these different trade-offs in an optimization problem by allowing a key-value pair to be replicated and stored on any subset of the memory banks. The $\varnothing$ option represents not keeping the key-value pair in the cache at all and recomputing the result from the database at every query, an option that can be computationally very costly. Simulation results from [7] show that it can be advantageous to store copies of data items in more than one memory bank to speed up recovery time, although it depends on the read and write frequencies of the data items as well as the failure rates of the memory.

[7] gives a detailed description for how the memory parameters and request frequencies translate into costs and uses the model to study a closely related problem in which one is given a fixed budget as well as the price for the different types of memory. The goal is to determine the optimal amount of each type of memory to purchase as well as the optimal placement of key-value pairs to memory banks that minimizes expected service time subject to the overall budget constraint. The algorithm in [7] is implemented and evaluated using traces generated by a standard social networking benchmark [4]. In this paper, we consider the situation in which the design of the cache is already determined in that there is a set of memory banks whose capacities are given as part of the problem input. The goal is to place each item on a subset of the memory banks so that the capacities of each memory bank is not exceeded and the total cost is minimized.

In both cases, the objective function is expected service time for all the items. The cost of serving an item p located on subset S of the memory banks is a sum of three terms: the total expected time to serve read requests to p, the total expected time to serve write requests to p, and the total expected time needed to restore p in the event that one or more memory bank in S fails. More explicitly:

$$\begin{aligned}\text{cost}(p,S) = {} & \text{read-freq}(p)\cdot\text{read-time}(p,S)\\ & + \text{write-freq}(p)\cdot\text{write-time}(p,S)\\ & + \sum_F \text{fail-freq}(F)\cdot\big(\text{read-time}(p,S\setminus F)+\text{write-time}(p,F\cap S).\big)\end{aligned}$$

The functions read-freq(p) and write-freq(p) represent the probability of a read or write request to p. The function fail-freq(F) is the probability that all the memory banks in subset F fail. On a request to read an item p, item p can be obtained from any of the copies of p in the cache. Therefore, read-time(p,S) represents the time to read p from the memory bank in S that provides the fastest read time. The time to read a data item from a memory bank depends on the size of the item as well as the latency and bandwidth for reading from that type of memory. Updating an item, on the other hand, requires updating every copy of that item in the cache. Therefore, write-time(p,S) is the maximum time to write p to any of the memory banks in S, assuming that writing p to its multiple destinations is done in parallel. (If writing is done sequentially, then write-time(p,S) is the sum of the write times over all the memory banks in S). The time to write a data item to a memory bank depends on the size of the item as well as the latency and bandwidth for writing to that type of memory. Recovering from failure could involve reading from those memory banks that still have a copy of p and rewriting them to those that lost it.

Since the relative read/write frequencies for items and read/write times for memories vary significantly, there is no useful structure to exploit in modeling the cost of assigning an item to a subset which is why they are assumed to be arbitrary values given as part of the input to the subset assignment problem. However, based on the empirical failure rates of the memory technologies, it is reasonable to assume that two memory banks will never fail at the same time. Under this assumption, there is no need to keep more than two copies of an item in the cache and we can restrict the data placements for an item to subsets of size one or two. The problem is addressed in this paper in its full generality although a better bound on the running time can be obtained with this restriction.

2 Problem Definition

There are n items and each item p has a given size(p). There are d bins $\mathcal{B} = \{b_1, \dots, b_d\}$. Each bin b has a given capacity(b). An item can be replicated and placed on any subset of the memory banks $S \subseteq \mathcal{B}$. We call S a *placement option* for an item. Placing p on S has cost denoted by $\text{cost}(p, S) \geq 0$. A placement of items to memory banks is described by a set of $n \cdot 2^d$ variables $x(p, S) \geq 0$ with the constraint that for each p,

$$\sum_S x(p, S) = \text{size}(p). \tag{1}$$

Also the capacity of each bin cannot be exceeded, so for each b,

$$\sum_{S \ni b} \sum_p x(p, S) \leq \text{capacity}(b). \tag{2}$$

The goal is to minimize

$$\sum_p \sum_S \text{cost}(p, S) x(p, S),$$

subject to the condition that all $x(p, S) \geq 0$, (1) and (2) above.

The placement option $\varnothing$, corresponding to not placing an item in any of the bins, is an option for every p, so the problem always has a feasible solution. For each bin b, we will add an extra item p whose size is capacity(b). For each added p, $\text{cost}(p, \varnothing) = \text{cost}(p, \{b\}) = 0$. For all other $S \subseteq \mathcal{B}$, $\text{cost}(p, S) = \infty$. We assume that the pages are numbered so that the extra item for bin b_i is p_i. With the additional items, we can assume that every solution under consideration has every bin filled exactly to capacity since any extra space in b_i can be filled with p_i without changing the cost of the solution. Therefore we require that for each b, $\sum_{S \ni b} \sum_p x(p, S) = \text{capacity}(b)$. An assignment which satisfies the equality constraints on the bins is called *perfectly filled*.

3 Preliminaries

Our algorithm starts with a feasible, perfectly filled solution and improves the assignment in a series of small steps, called augmentations. The augmentations, a generalization of a negative cycle in min-cost flow, always maintain the condition that the current assignment is feasible and perfectly filled. In each iteration the algorithm finds an augmentation that approximates the best possible augmentation in terms of the overall improvement in cost. An augmentation is a linear combination of moves in which mass is moved from $x(p, S)$ to $x(p, T)$ for some item p. Each move gives rise to a d-dimensional vector over $\{-1, 0, 1\}$

that denotes the net increase or decrease to each bin as a result of the move. We require that the linear combination of vectors for an augmentation equal $\vec{0}$ in order to maintain the condition that the bins are perfectly filled. The profile for an augmentation is the set of vectors corresponding to the moves in that augmentation. In order to find a good augmentation, we exhaustively search over all profiles and then find a good set of actual moves that correspond to each profile. Exhaustively searching over all profiles introduces a factor of $f(d)$, the number of distinct profiles which is at most $\binom{3^d}{d+1}$.

In order to bound the number of iterations, we also need to establish that there is an augmentation that improves the cost by a significant factor. For flows, this is accomplished by showing that the difference between the current solution and the optimal solution can be decomposed into at most m simple cycles, where m is the number of edges in the network. If Δ is the difference between the current and optimal cost, then there is a cycle that improves the cost by at least Δ/m. We proceed in a similar way, showing that the difference between two assignments can be decomposed into at most $2(n+d)$ augmentations any of which can be applied to the current assignment. Therefore there is an augmentation that improves the cost by at least $\Delta/2(n+d)$.

3.1 Augmentations

For $S \subseteq \mathcal{B}$, $\vec{S}$ is a d-dimensional vector whose i^{th} coordinate is 1 if $b_i \in S$ and is 0 otherwise. Let $\mathcal{V}$ be the set of all length d vectors over $\{-1, 0, 1\}$. A set $V \subseteq \mathcal{V}$ is said to be *minimally dependent* if V is linearly dependent and no proper subset of V is linearly dependent. If $V = \{\vec{v}_1, \ldots, \vec{v}_r\}$ is minimally dependent, then the values $\alpha_1, \ldots, \alpha_r$ such that $\sum_{i=1}^{r} \alpha_i \vec{v}_i = \vec{0}$ are unique up to a global constant factor. In order to make a unique vector $\vec{\alpha}$, we always maintain the convention that $\alpha_1 = 1$. A minimally dependent set V is said to be *positive* if the associated vector $\vec{\alpha} > \vec{0}$.

A *move* is defined by a triplet (p, S, T) that represents the possibility of moving mass from $x(p, S)$ to $x(p, T)$. The *profile* for a set of moves $\{(p_1, S_1, T_1), \ldots, (p_r, S_r, T_r)\}$ is the set of vectors $\{(\vec{T}_1 - \vec{S}_1), \ldots, (\vec{T}_r - \vec{S}_r)\}$. Note that the vector $\vec{T} - \vec{S}$ represents the net increase or decrease to each bin that results from moving one unit of mass from $x(p, S)$ to $x(p, T)$ for some p. A set of moves is called an *augmentation* if the set of vectors in its profile is minimally dependent and positive. Note that an augmentation contains at most $d + 1$ moves.

An augmentation $\mathcal{A} = \{(p_1, S_1, T_1), \ldots, (p_r, S_r, T_r)\}$ can be applied to a particular assignment $\vec{x}$ if for every $i = 1, \ldots r$, $x(p_i, S_i) > 0$. Let $\vec{\alpha}$ be the unique vector of values such that $\alpha_1 = 1$ and $\sum_{j=1}^{r} \alpha_j (\vec{T}_j - \vec{S}_j) = \vec{0}$. If the augmentation is applied with magnitude a to $\vec{x}$, then for every $(p_j, S_j, T_j) \in \mathcal{A}$, $x(p_j, S_j)$ is replaced with $x(p_j, S_j) - a\alpha_j$ and $x(p_j, T_j)$ is replaced with $x(p_j, T_j) + a\alpha_j$. The cost vector for an augmentation is $\vec{c}$, where $c_j = \mathrm{cost}(p_j, T_j) - \mathrm{cost}(p_j, S_j)$. The cost associated with applying the augmentation with magnitude a is $a(\vec{c} \cdot \vec{\alpha})$. Since the goal is to minimize the cost, we only apply augmentations whose cost is negative.

For augmentation $\mathcal{A} = \{(p_1, S_1, T_1), \ldots, (p_r, S_r, T_r)\}$, let $\mathcal{S}(\mathcal{A})$ be the set of all pairs (p, S) such that for some i, $p = p_i$ and $S = S_i$. For each $(p, S) \in \mathcal{S}(\mathcal{A})$, define

$$\alpha(p, S) = \sum_{i: p_i = p, S_i = S} \alpha_i.$$

The maximum magnitude with which the augmentation $\mathcal{A}$ can be applied to $\vec{x}$ is

$$\min_{(p,S) \in \mathcal{S}(\mathcal{A})} \frac{x(p, S)}{\alpha(p, S)}.$$

The following lemma is analogous to the fact for flows that says there is always a cycle in the network representing the difference between two feasible flows. The proof is given in the full version of the paper.

▶ **Lemma 1.** *Let $\vec{x}$ and $\vec{y}$ be two feasible, perfectly filled assignments to the same instance of the subset assignment problem. Then there is an augmentation that can be applied to $\vec{x}$ that consists only of moves of the form (p, S, T) where $x(p, S) > y(p, S)$ and $x(p, T) < y(p, T)$.*

3.2 Basic Feasible Assignments

An item is said to be *fractionally assigned* if there are two subsets $S \neq S'$, such that $x(p, S) > 0$ and $x(p, S') > 0$. If items can only be assigned to single bins as in the standard assignment problem, then it follows from total unimodularity that the optimal solution is integral, assuming that all the input values are integers. For the subset assignment problem, the optimal solution may not be integral, even if all the input values are integers. Here is an example in which the item sizes and bin capacities are 1, but an optimal solution must have fractionally assigned items: we have two items p and q and two bins b and c with costs

$$\begin{aligned} &\text{cost}(p, \varnothing) = 1, \quad \text{cost}(p, \{b, c\}) = 0, && \text{cost}(q, \varnothing) = \text{cost}(q, \{b, c\}) = C, \\ &\text{cost}(p, \{b\}) = \text{cost}(p, \{c\}) = C, && \text{cost}(q, \{b\}) = \text{cost}(q, \{c\}) = 0, \end{aligned}$$

where C is a large number. The optimal assignment is to equally distribute p over $\{b, c\}$ and $\varnothing$, and to equally distribute q over $\{b\}$ and $\{c\}$.

The linear programming formulation of the subset assignment problem has $n+d$ constraints. n constraints enforce that each p must be assigned: $\sum_S x(p, S) = size(p)$. The other d constraints, say that each bin must be exactly filled to capacity. Therefore, any basic feasible solution to the linear programming formulation of the subset assignment problem has at most $n + d$ non-zero variables. Since for every p, there is at least one S such that $x(p, S) > 0$ and $n >> d$, we know at least $n - d$ of the items will not be fractionally assigned because they have only one S such that $x(p, S) > 0$. The number of variables $x(p, S)$ such that $0 < x(p, S) < \text{size}(p)$ is at most $2d$, so the number of fractionally assigned items is at most d.

The criteria for a feasible solution to be a basic feasible solution is that once the variables are chosen that will be positive, there is exactly one way to assign values to those variables so that all the constraints are satisfied. Suppose we have a feasible assignment $\vec{x}$. First place the items in bins that are not fractionally assigned. If $\vec{x}$ is a basic feasible solution, then there is a unique way to place the remaining items so that the bins are filled exactly to capacity. We rephrase the definition of a basic feasible solution in the language of the subset assignment problem and prove the same facts about the new definition.

Consider a feasible assignment $\vec{x}$. Let P_{frac} be the set of data items that are fractionally assigned. Let X_{frac} be the set of variables $x(p, S)$ such that $0 < x(p, S) < \text{size}(p)$. Let P_{int} be the set of items that are assigned to exactly one subset. That is $p \in P_{\text{int}}$ if $x(p, S) \in \{0, \text{size}(p)\}$ for all S.

▶ **Definition 2.** For each $p \in P_{\text{frac}}$ select one S such that $x(p, S) > 0$. Denote the selected set for p by S_p. Let X be the set of variables $x(p, S)$ such that $S \neq S_p$ and $0 < x(p, S) < \text{size}(p)$. Let V be the set of vectors $\vec{S} - \vec{S}_p$ for each $x(p, S) \in X$. Then $\vec{x}$ is a *basic feasible assignment* (*bfa*) if and only if V is linearly independent.

Although the definition for a basic feasible assignment was given in terms of a particular choice of S_p's, the property of being a *bfa* does not depend on this choice.

▶ **Lemma 3.** *The condition of being a* bfa *does not depend on the choice of S_p, for $p \in P_{frac}$.*

Proof. Let $p \in P_{\text{frac}}$ and let $\{S_1, \ldots, S_r\}$ be the subsets such that $x(p, S_j) > 0$. Suppose that S_p is chosen to be S_i. Select any two $S_j \neq S_k$. Since $(\vec{S_j} - \vec{S_k}) = (\vec{S_j} - \vec{S_p}) - (\vec{S_k} - \vec{S_p})$, the space spanned by all $(\vec{S_j} - \vec{S_k})$ for $S_j \neq S_k$ is equal to the space spanned by all $(\vec{S_j} - \vec{S_p})$ for $S_j \neq S_p$. The space spanned by all $(\vec{S_j} - \vec{S_k}) \neq \vec{0}$ is independent of the choice of S_p. ◀

▶ **Lemma 4.** *If $\vec{x}$ is a* bfa*, then the number of variables in X_{frac} is at most $2d$ and the number of fractionally assigned items is at most d.*

Proof. Since $|X| = |V|$, and V must be linearly independent for any *bfa*, it must be that if $\vec{x}$ is a *bfa*, then $|X| \leq d$. The set of fractionally assigned variables (X_{frac}) includes all the $x(p, S_p)$ for $p \in P_{\text{frac}}$ and X. For each $x(p, S_p)$, there is at least one variable in X. Therefore the number of variables such that $0 < x(p, S) < \text{size}(p)$ in any *bfa* is at most $2d$. ◀

The process Restore, given in [10], takes an assignment $\vec{x}$ which may not be a *bfa* and restores it to an assignment which is a *bfa*. The process maintains the condition that the current assignment is feasible and perfectly filled. If the set V is linearly dependent, a linear combination of the moves (p, S_p, S) is chosen for each $x(p, S) \in X$ such that applying the linear combination of moves keeps the bins perfectly filled. Since p has some weight on S_p and some weight on S, all the moves can be applied in either the forward or reverse direction. (A negative coefficient denotes applying a move in the reverse direction.) We pick a direction for the linear combination of moves such that the cost does not increase. The combination of moves is applied until either $x(p, S)$ or $x(p, S_p)$ becomes 0 for one of the moves represented in V. Thus, the cost of the assignment does not increase and the number of fractionally assigned variables decreases by at least one. The process continues until V is linearly independent.

▶ **Lemma 5.** *There is an optimal solution that is also a* bfa.

Proof. Start with an optimal assignment $\vec{x}$ which may not be a *bfa*. Apply Restore to $\vec{x}$. The resulting assignment is a *bfa*. And since the cost of $\vec{x}$ does not increase, $\vec{x}$ is still optimal. ◀

4 The Algorithm

The algorithm we present proceeds in a series of iterations. In each iteration, we apply an augmentation to the current assignment. Since the resulting assignment may no longer be a *bfa*, we then apply Restore to turn the solution back into a *bfa*.

Algorithm 1 MainLoop

```
$x(p, S) = 0$, for all $p$ and $S$.
$x(p_i, \{b_i\}) = \text{capacity}(b_i)$, for $i = 1, \ldots, d$. (Fill each bin with the "extra" items.)
$x(p_j, \varnothing) = \text{size}(p_j)$, for $j > d$. (All the "original" items start outside the bins.)
$\mathcal{P} = \text{Preprocess}(d)$
$\mathcal{A} = \text{FindAugmentation}(\vec{x})$
while $\mathcal{A} \neq \varnothing$
    Apply $\mathcal{A}$ to $\vec{x}$ with the largest possible magnitude
    Restore($\vec{x}$). (Transform $\vec{x}$ into a bfa.)
    $\mathcal{A} = \text{FindAugmentation}(\vec{x})$
```

Note that it is possible to find an augmentation that moves from a *bfa* to another *bfa* directly. This is essentially what the simplex algorithm does. However, not every augmentation results in a *bfa*. The augmentations that do result in a *bfa* must include the moves that shift mass between the fractionally assigned items. (These moves correspond to the vectors V described in the definition of a *bfa*). Restricting the augmentation in this way may result in a sub-optimal augmentation. For example, those augmentations could require decreasing a variable that is already very small in which case the augmentation can not be applied with very large magnitude. So we allow the algorithm to select from the set of all augmentations to get as much benefit as possible, and then move the assignment to a *bfa*.

4.1 Finding an augmentation that is close to the best possible

The first step is a preprocessing step in which every possible augmentation profile is generated. This consists of generating every minimally dependent subset V of $\mathcal{V}$ and its associated $\vec{\alpha}$. Preprocess (shown in the full version [10] of this paper) runs in time $O(\binom{3^d}{d+1}\,\mathrm{poly}(d))$. The number of distinct augmentation profiles returned by the procedure is at most $\binom{3^d}{d+1}$.

Given an augmentation profile $V = \{\vec{v}_1, \ldots, \vec{v}_r\}$, the goal is to find an augmentation whose profile matches V and can be applied with a magnitude that gives close to the best possible improvement. For each vector $\vec{v} \in \mathcal{V}$, we maintain a data structure with every move (p, S, T) such that $\vec{T} - \vec{S} = \vec{v}$ and $x(p, S) > 0$. We will call the set of all such moves *Moves*$(\vec{v})$. The data structure should be able to answer queries of the form: given x_0, find the move (p, S, T) such that $\mathrm{cost}(p, T) - \mathrm{cost}(p, S)$ is minimized subject to the condition that $x(p, S) \geq x_0$. These kind of queries can be handled by an augmented binary search tree in logarithmic time [6].

For a given *bfa* $\vec{x}$ and augmentation $\mathcal{A}$, one can calculate the maximum possible magnitude a with which $\mathcal{A}$ can be applied to $\vec{x}$. We will make use of upper and lower bounds for the value a for any augmentation and *bfa* combination. Call these values $a_{\max}$ and $a_{\min}$. Round $a_{\min}$ down so that $a_{\max}/a_{\min}$ is a power of 2. The while loop in procedure FindAugmentation runs for $\log(a_{\max}/a_{\min})$ iterations.

Algorithm 2 FindAugmentation($\vec{x}$)

```
$BestCost = 0$
$\mathcal{A} = \varnothing$
for each augmentation profile $V = \{\vec{v}_1, \ldots, \vec{v}_r\}$ and vector $\vec{\alpha}$
    $a = a_{\max}/2$
    while $a \geq a_{\min}$
        for $i = 1, \ldots, r$
            Let $(p_i, S_i, T_i)$ be the move with the smallest cost among moves in
                $Moves(\vec{v}_i)$ such that $x(p_i, S_i) \geq a \cdot \alpha_i$.
            $c_i = \mathrm{cost}(p_i, T_i) - \mathrm{cost}(p_i, S_i)$
        $CurrentCost = \sum_{i=1}^{r} a \cdot c_i \cdot \alpha_i$
        if $CurrentCost < BestCost$
            $BestCost = CurrentCost$
            $\mathcal{A} = \{(p_1, S_1, T_1), \ldots, (p_r, S_r, T_r)\}$
        $a = a/2$
return $\mathcal{A}$
```

For an augmentation $\mathcal{A}$ that can be applied to assignment $\vec{x}$ with magnitude a, the total change in cost is denoted by $\mathrm{cost}(\mathcal{A}, \vec{x}, a)$. Recall that since we are minimizing cost we will only apply an augmentation if the total change in cost is less than 0.

▶ **Lemma 6.** *Let $\mathcal{A}_1$ be the augmentation returned by FindAugmentation$(\vec{x})$ and $\mathcal{A}_2$ be another augmentation. If a_1 and a_2 are the maximum magnitudes with which $\mathcal{A}_1$ and $\mathcal{A}_2$ can be applied to $\vec{x}$, then $2d \cdot \mathrm{cost}(\mathcal{A}_1, \vec{x}, a_1) \leq \mathrm{cost}(\mathcal{A}_2, \vec{x}, a_2)$.*

Proof. Let V_2 be the profile for $\mathcal{A}_2$. Let $\vec{\alpha}$ be the vector associated with the profile V_2. Let $\bar{a}$ be the value of the form $a_{\max}/2^j$ such that $2\bar{a} > a_2 \geq \bar{a}$. There is an iteration inside the while loop of FindAugmentation($\vec{x}$) in which the augmentation profile is V_2 and the value for a is $\bar{a}$. The augmentation constructed in this iteration will be called V_3. The moves in $\mathcal{A}_2$ are $\{(p_1^{(2)}, S_1^{(2)}, T_1^{(2)}), \ldots, (p_r^{(2)}, S_r^{(2)}, T_r^{(2)})\}$. The moves in $\mathcal{A}_3$ are $\{(p_1^{(3)}, S_1^{(3)}, T_1^{(3)}), \ldots, (p_r^{(3)}, S_r^{(3)}, T_r^{(3)})\}$. Note that since V_2 can be applied to $\vec{x}$ with magnitude a_2, it must be the case that for $i = 1, \ldots, r$, $x(p_i^{(2)}, S_i^{(2)}) \geq \alpha_i a_2$ because applying the moves involves removing $\alpha_i a_2$ from $x(p_i^{(2)}, S_i^{(2)})$. Since $a_2 \geq \bar{a}$, $x(p_i^{(2)}, S_i^{(2)}) \geq \alpha_i \bar{a}$. The move $(p_i^{(3)}, S_i^{(3)}, T_i^{(3)})$ is chosen to be the move with minimum cost such that $x(p_i^{(3)}, S_i^{(3)}) \geq \alpha_i \bar{a}$. Therefore the cost of $(p_i^{(3)}, S_i^{(3)}, T_i^{(3)})$ is at most the cost of $(p_i^{(2)}, S_i^{(2)}, T_i^{(2)})$. The value of the variable *CurrentCost* for that iteration is

$$\begin{aligned}
\mathit{CurrentCost}_3 &= \bar{a} \sum_{i=1}^{r} \alpha_i \left[\operatorname{cost}(p_i^{(3)}, T_i^{(3)}) - \operatorname{cost}(p_i^{(2)}, S_i^{(3)})\right] \\
&\leq \bar{a} \sum_{i=1}^{r} \alpha_i \left[\operatorname{cost}(p_i^{(2)}, T_i^{(2)}) - \operatorname{cost}(p_i^{(2)}, S_i^{(2)})\right] \\
&\leq \frac{a_2}{2} \sum_{i=1}^{r} \alpha_i \left[\operatorname{cost}(p_i^{(2)}, T_i^{(2)} - \operatorname{cost}(p_i^{(2)}, S_i^{(2)})\right] = \frac{1}{2} \operatorname{cost}(\mathcal{A}_2, \vec{x}, a_2)
\end{aligned}$$

Let $\mathit{CurrentCost}_1$ be the value of the variable *CurrentCost* and a' the value of the variable a during the iteration in which the augmentation $\mathcal{A}_1$ is considered. Since $\mathcal{A}_1$ was selected by FindAugmentation, $\mathit{CurrentCost}_1 \leq \mathit{CurrentCost}_3$. It remains to show that the maximum magnitude with which $\mathcal{A}_1$ can be applied is at least a'/d and therefore the actual change in cost at most $\mathit{CurrentCost}_1/d$.

Let V_1 be the profile for $\mathcal{A}_1$ and let $\vec{\beta}$ be the vector associated with profile V_1. Since we are now only referring to one augmentation, we omit the subscripts and call the moves in $\mathcal{A} = \{(p_1, S_1, T_1), \ldots, (p_r, S_r, T_r)\}$. We are guaranteed by the selection of the move (p_i, S_i, T_i) that for every i, $x(p_i, S_i)/\beta_i \geq a'$. Let $\beta_{p,S}^{\text{sum}}$ and $\beta_{p,S}^{\max}$ denote the sum and maximum over all β_i such that $p_i = p$ and $S_i = S$. The value of a_1, the maximum value with which $\mathcal{A}$ can be applied, is equal to $x(p, S)/\beta_{p,S}$ for some pair (p, S). We have

$$a_1 = \frac{x(p, S)}{\beta_{p,S}^{\text{sum}}} \geq \frac{x(p, S)}{d \cdot \beta_{p,S}^{\max}} \geq \frac{a'}{d}.$$

◀

4.2 Number of iterations of the main loop

The procedure FindAugmentation takes a *bfa* $\vec{x}$ and returns an augmentation that reduces the cost of the current solution by an amount which is within $\Omega(1/d)$ of the best possible augmentation that can be applied to $\vec{x}$. In order to bound the number iterations of the main loop, we need to show that there always is a good augmentation that can be applied to $\vec{x}$ that moves it towards an optimal solution. The idea is that for any two assignments $\vec{x}$ and $\vec{y}$, $\vec{x}$ can be transformed into $\vec{y}$ by applying a sequence of augmentations. Each augmentation decreases the number of variables in which $\vec{x}$ and $\vec{y}$ differ by one. Since the number of non-zero variables in any *bfa* is at most $n + d$, there are at most $2(n + d)$ augmentations in the sequence. Thus, if the difference in cost between $\vec{y}$ and $\vec{x}$ is Δ, one of the augmentations will decrease the cost by at least $\Delta/2(n + d)$. The idea is analogous to the partitioning the difference between two min cost flows into a set of disjoint cycles. Some additional work is required to establish that the chosen augmentation can be applied with sufficient magnitude.

The proofs of Lemmas 7, 8, 9 and 10 are given in the full version [10] of the paper.

▶ **Lemma 7.** *Let $\vec{x}$ be a* bfa *for an instance of the subset assignment problem and let Δ be the difference in the objective function between $\vec{x}$ and the optimal solution. Then there is an augmentation $\mathcal{A}$ such that when $\mathcal{A}$ is applied to $\vec{x}$ with the maximum possible magnitude, the cost drops by at least $\Delta/2(n+d)$.*

In order to bound the number of iterations in the main loop, we need to know the smallest difference in cost between two assignments that have different cost.

▶ **Lemma 8.** *If A is an invertible $d \times d$ matrix with entries in $\{-1,0,1\}$ and $\vec{b}$ is a d-vector with integer entries, then there is an integer $\ell \leq d^{d/2}$ such that the solution $\vec{x}$ to $A\vec{x} = \vec{b}$ has entries of the form k/ℓ where k is an integer. Moreover, if $\vec{b}$ also has entries in $\{-1,0,1\}$, then the entries of x are at most equal to d.*

The following bound comes from the fact that the fractionally assigned values are the solution to a matrix equation with a $d \times d$ matrix over $\{-1,0,1\}$.

▶ **Lemma 9.** *If $\vec{x}$ is a* bfa*, then there is an integer $\ell \leq d^{d/2}$ such that every $x(p,S) = k/\ell$ for some integer k.*

With this result, we can bound the number of iterations in our algorithm.

▶ **Lemma 10.** *The number of iterations of the main loop is $O(nd^2 \log(dnC))$.*

4.3 Analysis of the running time

The running time of Preprocess is dominated by the running time of the main loop, so we just analyze the running time of the main loop. To bound the size of the augmented binary search trees $Moves(\vec{v})$, observe that for each S, there is at most one T such that $\vec{T} - \vec{S} = \vec{v}$. Therefore, the number of moves (p,S,T) that can be stored in a single tree is $O(2^d n)$. Updates are handled in logarithmic time, so the time per update to an entry in one of the trees is $O(d \log n)$. Every time a variable $x(p,S)$ changes, there are 2^d subsets T such that the move (p,S,T) must be updated. In each iteration of the main loop there are $O(d)$ variable changes, resulting in a total update time of $O(d^2 2^d \log n)$.

By Lemma 4, the *bfa* at the beginning of an iteration has at most $2d$ fractionally assigned variables. An augmentation consists of at most $d+1$ moves and therefore changes the value of at most $2(d+1)$ variables. Thus, the input to Restore is an assignment with $O(d)$ fractionally assigned variables. Each iteration of Restore reduces the number of fractionally assigned variables by at least one. Therefore, the number of iterations of Restore is bounded by $O(d)$ and the total time spent in Restore during an iteration is $\mathrm{poly}(d)$.

The inner loop of FindAugmentation requires $O(d)$ queries to one of the augmented binary search trees resulting in $O(d^2 \log n)$ time for each iteration of the inner loop. The number of times the inner loop is executed is $\log(a_{\max}/a_{\min})$ times the number of augmentation profiles, $f(d)$. Therefore the running time of FindAugmentation dominates the running time of an iteration of the main loop which is $O(f(d)d^2 \log n \log(a_{\max}/a_{\min}))$. By Lemma 10, the number of iterations of the main loop is $O(nd^2 \log(dnC))$, and since $f(d) \leq \binom{3^d}{d+1}$, the total running time is $O(\binom{3^d}{d+1} d^4 n \log n (\log n + \log C) \log(a_{\max}/a_{\min}))$. We now bound $a_{\max}/a_{\min}$:

▶ **Lemma 11.** *The values of a are bounded above by $a_{\max} = d^{d/2}Z$ and below by $a_{\min} = 1/d^{d/2+1}$, where $Z = \max_p \mathrm{size}(p)$.*

The proof of Lemma 11 is given in the full version [10]. Hence, we get that $\log(a_{\max}/a_{\min})$ is $O(d^2 \log d \log Z)$ and the total running time is $O(\binom{3^d}{d+1} \mathrm{poly}(d) n \log(n) \log(nC) \log(Z))$.

References

1 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1 edition, 2 1993. doi:10.1016/0166-218X(94)90171-6.

2 R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM J. Comput.*, 23(5):906–933, 1994. doi:10.1137/S0097539791199334.

3 T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. Linkbench: a database benchmark based on the Facebook social graph. In *SIGMOD*. ACM, 2013. doi:10.1145/2463676.2465296.

4 Sumita Barahmand and Shahram Ghandeharizadeh. BG: a benchmark to evaluate interactive social networking actions. In *CIDR*, January 2013.

5 Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *SODA*, pages 213–222. ACM, 2000. doi:10.1137/S0097539700382820.

6 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, third edition, 2009.

7 S. Ghandeharizadeh, S. Irani, and J. Lam. Memory hierarchy design for caching middleware in the age of NVM. Technical Report 2015-01, USC Database Laboratory, 2015. URL: http://dblab.usc.edu/Users/papers/CacheDesTR2.pdf.

8 S. Ghandeharizadeh, S. Irani, J. Lam, and J. Yap. CAMP: A cost adaptive multi-queue eviction policy for key-value stores. Technical Report 2014-07, USC Database Lab, 2014. URL: http://dblab.usc.edu/Users/papers/CAMPTR.pdf.

9 S. Ghandeharizadeh, S. Irani, J. Lam, and J. Yap. CAMP: a cost-aware multiqueue eviction policy. In *Middleware 2014*. Springer, 2014. doi:10.1145/2663165.2663317.

10 Shahram Ghandeharizadeh, Sandy Irani, and Jenny Lam. The subset assignment problem for data placement in caches. *ArXiv ePrint*, abs/1609.08767, 2016. URL: http://arxiv.org/abs/1609.08767.

11 D. Gusfield, C. Martel, and D. Fernández-Baca. Fast algorithms for bipartite network flow. *SIAM J. Comput.*, 16(2):237–251, 1987. doi:10.1137/0216020.

12 P. Jelenkovic and A. Radovanovic. Asymptotic insensitivity of least-recently-used caching to statistical dependency. In *INFOCOM 2003.*, pages 438–447 vol.1, March 2003. doi:10.1109/INFCOM.2003.1208695.

13 N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984. doi:10.1007/BF02579150.

14 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems.* Springer, 2004.

15 Hyojun Kim, Sangeetha Seshadri, Clement L. Dickey, and Lawrence Chiu. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. *Trans. Storage*, 10(4):15:1–15:21, October 2014. doi:10.1145/2668128.

16 Christos Koufogiannakis and Neal E. Young. A nearly linear-time PTAS for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, December 2014. doi:10.1007/s00453-013-9771-6.

17 Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Inc., 1990.

18 Mihir Nanavati, Malte Schwarzkopf, Jake Wires, and Andrew Warfield. Non-volatile storage. *Commun. ACM*, 59(1):56–63, December 2015. doi:10.1145/2814342.

19 R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, et al. Scaling Memcache at Facebook. *NSDI*, 13:385–398, 2013.

20 David Starobinski and David Tse. Probabilistic methods for web caching. *Performance Evaluation*, 46(2–3):125–137, 2001. Advanced Performance Modeling. doi:10.1016/S0166-5316(01)00045-1.

A Gap Trichotomy for Boolean Constraint Problems: Extending Schaefer's Theorem

Lucy Ham

Department of Mathematics and Statistics, La Trobe University, Melbourne, Australia
leham@students.latrobe.edu.au

Abstract

In this paper, we investigate "gap problems", which are promise problems where YES instances are flexibly satisfiable in a certain sense, and NO instances are not satisfiable at all. These gap problems generalise a family of constraint-related decision problems, including the constraint satisfaction problem itself, the *separation problem* (can distinct variables be validly assigned distinct values?) and the 2-*robust satisfiability problem* (does any assignment on two variables extend to a full satisfying assignment?). We establish a Gap Trichotomy Theorem, which on Boolean domains, completely classifies the complexity of the gap problems considered. As a consequence, we obtain several well-known dichotomy results, as well as dichotomies for the separation problem and the 2-robust satisfiability problem: all are either polynomial-time tractable or **NP**-complete. Schaefer's original dichotomy is a notable particular case.

1998 ACM Subject Classification F.1.3 [Complexity Measures and Classes] reducibility and completeness, F.4.1 [Mathematical Logic] Logic and constraint programming

Keywords and phrases Constraint Satisfaction Problem, Robust satisfiability, Clone theory, Dichotomy, Trichotomy, Boolean

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.36

1 Introduction

Constraint Satisfaction Problems (CSPs) occur widely in practice, both as natural problems, and as an underlying framework for constraint programming; see Tsang [23]. When the template is restricted to some fixed finite domain, these problems still cover many important practical problems as well as providing an important framework for theoretical considerations in computational complexity. In the case of Boolean (2-element) domains, constraint problems coincide with the SAT variants examined by Schaefer [20]. In his paper, Schaefer proved a famous dichotomy: he showed that the complexity of CSPs over a fixed Boolean constraint language is either decidable in polynomial time or is NP-complete. Since Schaefer's seminal contribution, there have been enormous advances toward a more general dichotomy for constraint satisfaction problems on non-Boolean domains. In [10], Feder and Vardi argue that fixed template CSPs emerge as the broadest natural class for which a dichotomy might hold and propose the well-known *Dichotomy Conjecture*. Numerous extensions of Schaefer's result are now known. Amongst the broadest of these include the case of three-element domains (Bulatov [7]), List Homomorphism Problems (Bulatov [8]), and the case of directed graphs without sources and sinks (Barto, Kozik and Niven [4]).

In addition to direct extensions of Schaefer's results, many variants of constraint satisfaction problems have been shown to experience dichotomies like that of Schaefer's, such as counting CSPs [9] and balanced CSPs [21]. We explore computational complexity for notions of "flexible satisfaction": instead of asking for the existence of a single solution,

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 36; pp. 36:1–36:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

one asks for enough solutions to satisfy a range of conditions. We focus in particular on *separability* and *robust satisfiability.* The separation problem SEP asks if it is true that, for every pair of distinct variables u and v, there is a solution giving u a different value to v. The $(k, \mathcal{F})$-robust satisfiability problem asks if every compatible partial assignment on k variables extends to a full solution. As explained in Jackson [13], the SEP condition arises naturally in universal algebraic considerations, but is also closely related to problems without a backbone: problems (typically SAT variants) where no variable is forced to take some fixed value. Implicit constraints such as these are widely associated with computational difficulty; see Monasson et al. [17] or Beacham and Culberson [5]. In the language of [5] for example, the SEP condition corresponds to the unfrozenness of equality. The $(k, \mathcal{F})$-robustness condition is an extension of a robustness condition of Abramsky, Gottlob and Kolaitis [1], who studied robust satisfiability in relation to hidden-variable models in quantum mechanics and explicitly invite the systematic study of the complexity of robust satisfiability for constraint problems. In the present article, we completely classify the complexity of the $(2, \mathcal{F})$-robust problem and the separation problem, in the case of Boolean domains.

Recall that for disjoint languages Y and N, the *promise problem* (Y, N) is the decision problem for Y, where instances are promised to lie in $Y \cup N$; see Goldreich [12] for example. The problem (Y, N) is **NP**-hard if it is **NP**-hard to decide membership in any language S containing Y and disjoint from N. In 2011, Gottlob proved **NP**-hardness of a promise problem relating to $(3k + 3)$SAT: if $\mathrm{Y}_{(k\text{-Rob})}$ denotes the set of all $(3k + 3)$SAT instances for which every possible partial assignment on k variables extends to a satisfying solution and $\mathrm{N}_{\mathrm{CSP}}$ is the set of all NO instances for $(3k + 3)$SAT, then $(\mathrm{Y}_{(k\text{-Rob})}, \mathrm{N}_{\mathrm{CSP}})$ is **NP**-hard. This promise problem can be more precisely described as a *gap problem*, because having no solutions at all is a strong shortfall relative to having k-robust satisfiability [12, p. 259].

Abramsky, Gottlob and Kolaitis [1] and then Jackson showed [13] that **NP**-hard gap problems are also to be found for some other well-known **NP**-complete problems, including 3 SAT, G3C, NAE3 SAT, and positive 1-in-3 SAT. We investigate gap problems in the Boolean case, establishing a Gap Trichotomy Theorem (Theorem 6) that provides dichotomies for these flexible satisfaction problems, as well as several known dichotomy results. A notable consequence is the recovery of Schaefer's Theorem in case of core relational structures. In addition to providing unified proofs for these dichotomies, the Gap Trichotomy Theorem reveals that whenever the constraint satisfaction problem is hard, the more general promise problem is also hard. The Gap Trichotomy Theorem also gives a continuum of examples in the style of the five examples mentioned above.

The fundamental tools used in the aforementioned extensions of Schaefer's dichotomy for Boolean CSPs to higher domains and other related computational problems has been the algebraic analysis of "polymorphisms" (see Definition 7 below). For SEP and robust satisfiability, it is necessary to move to partial polymorphisms. As a second main result, we show that the basic universal algebraic methods can nevertheless be established in this setting, see Theorem 8 below.

2 Preliminaries: Separation and Robustness

We introduce four computational problems that will be of primary focus in this article.

▶ **Definition 1.** Let Γ be a set of relation symbols, each with an associated finite arity. A *template* is a pair $\mathbb{A} = \langle A; \Gamma^{\mathbb{A}} \rangle$ consisting of a *finite* set A together with an interpretation of each n-ary relation symbol $r \in \Gamma$ as a subset $r^{\mathbb{A}}$ of A^n. The set $\Gamma^{\mathbb{A}} = \{r^{\mathbb{A}} \mid r \in \Gamma\}$ is often referred to as a *constraint language over domain A*.

We define a $\Gamma^{\mathbb{A}}$*-instance* to be a triple $I = (V; A; \mathcal{C})$ consisting of a set of *variables* V, the *domain* set A, and a set of *constraints* $\mathcal{C}$. Each constraint $c \in \mathcal{C}$ is a pair $\langle s, r^{\mathbb{A}} \rangle$, where $r^{\mathbb{A}}$ is a k-ary relation in $\Gamma^{\mathbb{A}}$ and $s = (v_1, \ldots, v_k)$ is a k-tuple involving variables from V. We define a *solution* of I to be any assignment $\phi \colon V \to A$ such that for each $c = \langle (v_1, \ldots, v_k), r^{\mathbb{A}} \rangle$ in $\mathcal{C}$, we have $(\phi(v_1), \ldots, \phi(v_k)) \in r^{\mathbb{A}}$.

Constraint satisfaction problem CSP($\mathbb{A}$) *over template* $\mathbb{A}$.
Instance: a $\Gamma^{\mathbb{A}}$-instance I.
Question: is there a solution of I?

Nontrivial satisfaction problem $\text{CSP}_{\text{NonTriv}}(\mathbb{A})$ *over template* $\mathbb{A}$.
Instance: a $\Gamma^{\mathbb{A}}$-instance I.
Question: is there a nontrivial solution of I?

Separation problem SEP($\mathbb{A}$) *over template* $\mathbb{A}$.
Instance: a $\Gamma^{\mathbb{A}}$-instance I.
Question: for every pair $\{v_1, v_2\}$ of distinct variables in V, is there is a solution $\phi \colon V \to A$ of I such that $\phi(v_1) \neq \phi(v_2)$?

Our fourth computational problem of interest requires some further definitions.

▶ **Definition 2.** Let R be a set of finitary relation symbols and let $X = \{x_i \mid i \in I\}$ be a set of pairwise distinct variables. A formula in the language of R is called a *primitive-positive formula* (abbreviated to pp-formula) if, for some $\ell \in \mathbb{N}_0$ and $m, n \in \mathbb{N}$, it is of the form:

$$(\exists w_1, \ldots, w_\ell) \bigwedge_{i=1}^{m} \alpha_i(x_1, \ldots, x_k, w_1, \ldots, w_\ell),$$

where $w_1, \ldots, w_\ell, x_1, \ldots, x_k$ are distinct variables, and each $\alpha_i(x_1, \ldots, x_k, w_1, \ldots, w_\ell)$ is either of the form $y \approx z$, where $\approx$ is the symbol for the equality relation and $y, z \in \{x_1, \ldots, x_k, w_1, \ldots, w_\ell\}$, or of the form $(y_1, \ldots, y_k) \in r$, for some k and relation $r \in R$ of arity k and $\{y_1, \ldots, y_k\} \subseteq \{x_1, \ldots, x_k, w_1, \ldots, w_\ell\}$.

The particular case where $\ell = 0$ (that is, no quantifiers) is used later, and is called a *conjunct-atomic formula.*

▶ **Definition 3.** Let $\Gamma^{\mathbb{A}}$ be a constraint language over a finite set A and let $\mathcal{F}$ be a finite set of pp-formulæ in the language of Γ. Let $(V; A; \mathcal{C})$ be a constraint instance for $\Gamma^{\mathbb{A}}$. For a subset $S \subseteq V$, we say that an assignment $f \colon S \to A$ is $\mathcal{F}$*-compatible* if it preserves $\mathcal{F}$.

In other words, if for some $\rho(x_1, \ldots, x_k) \in \mathcal{F}$ and some tuple $(s_1, \ldots, s_k) \in S^k$ the formula $\rho(s_1, \ldots, s_k)$ is true in $(V; A; \mathcal{C})$, then $\rho(f(s_1), \ldots, f(s_k))$ must be true in $\mathbb{A}$.

In the following, we let k be a nonnegative integer and $\mathcal{F}$ be a finite set of pp-formulæ in the language of Γ.

The $(k, \mathcal{F})$*-robust satisfiability problem* $(k, \mathcal{F})$-Robust($\mathbb{A}$) *over template* $\mathbb{A}$.
Instance: a $\Gamma^{\mathbb{A}}$-instance I.
Question: does every $\mathcal{F}$-compatible assignment on k variables extend to a solution of I?

In the case where $\mathcal{F}$ consists of the set of pp-formulæ defining all projections of relations in $\Gamma^{\mathbb{A}}$, the notion of $\mathcal{F}$-compatibility has been called "local compatibility" and $(k, \mathcal{F})$-robust satisfiability called "k-robust satisfiability", see [1, §2]. In [13, Lemma 3.1], Jackson proposes that $\mathcal{F}$-compatibility is the natural localness condition in general.

3 Main results: a gap trichotomy

The first main result presents a trichotomy of computational gap theorems for Boolean constraint languages. As consequences we obtain dichotomy theorems for each of the four computational problems described above.

▶ **Notation 4.** *Let Γ be a finite set of relations on $\{0, 1\}$, let $k \in \mathbb{N}$ and let $\mathcal{F}$ be a finite set of pp-formulæ in the language of Γ. We use*

- $\mathrm{N}_{\mathrm{CSP}}(\Gamma)$ *to denote the set of NO instances for* $\mathrm{CSP}(\Gamma)$*,*
- $\mathrm{N}_{\mathrm{NonTriv}}(\Gamma)$ *to denote the set of NO instances for* $\mathrm{CSP}_{\mathrm{NonTriv}}(\Gamma)$*,*
- $\mathrm{Y}_{(k,\mathcal{F})}(\Gamma)$ *to denote the set of YES instances for* $(k, \mathcal{F})$-$\mathrm{Robust}(\Gamma)$*,*
- $\mathrm{Y}_{\mathrm{SEP}}(\Gamma)$ *to denote the set of YES instances for* $\mathrm{SEP}(\Gamma)$*,*
- $\mathrm{Y}_{\mathrm{SEP}\cap(k,\mathcal{F})}(\Gamma)$ *to denote the set of instances in* $\mathrm{Y}_{\mathrm{SEP}}(\Gamma) \cap \mathrm{Y}_{(2,\mathcal{F})}(\Gamma)$*.*

When the context refers to a specific constraint language Γ, we omit Γ from this notation.

▶ **Definition 5.** If P and Q are disjoint sets of Γ-instances, we say that Γ *satisfies* $\mathrm{GAP}(P, Q)$ (or *has the gap property* $\mathrm{GAP}(P, Q)$) if the promise problem (P, Q) is **NP**-hard.

We can now state one of main results of the article.

▶ **Theorem 6** (Gap Trichotomy Theorem)**.** *Let Γ be a constraint language on $\{0, 1\}$. Exactly one of the following statements is true.*

1. Γ *satisfies* $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}} \cap \mathrm{Y}_{(2,\mathcal{F})}, \mathrm{N}_{\mathrm{CSP}})$ *for some finite set of pp-formulæ* $\mathcal{F}$*.*
2. $\mathrm{CSP}(\Gamma)$ *is trivial but* Γ *satisfies* $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}} \cap \mathrm{Y}_{(2,\mathcal{F})}, \mathrm{N}_{\mathrm{NonTriv}})$ *for some finite set of pp-formulæ* $\mathcal{F}$*.*
3. *The satisfiability problem,* $(2, \mathcal{F})$-$\mathrm{Robust}(\Gamma)$ *and the separation problem* $\mathrm{SEP}(\Gamma)$ *are solvable in polynomial-time, for any finite set of pp-formulæ* $\mathcal{F}$*.*

▶ Remark. The language of polymorphisms and clone theory can be used to express precise boundaries for when each condition of the three applies to a given Γ. We give full details including which co-clones give rise to which complexity condition below; see Figure 1 and the associated discussion. An overview of the proof of the Gap Dichotomy Theorem is given in Section 7.

A number of *dichotomy* theorems are immediate consequences of the Gap Trichotomy Theorem. We list four examples.

- (Schaefer's Dichotomy Theorem [20].) Observe that $\mathrm{CSP}(\mathbb{A})$ is **NP**-complete in case 1, and polynomial time solvable in cases 2 and 3.
- (Dichotomy Theorem for $\mathrm{CSP}_{\mathrm{NonTriv}}(\mathbb{A})$.) Observe that $\mathrm{CSP}_{\mathrm{NonTriv}}(\mathbb{A})$ is **NP**-complete in cases 1 and 2, and polynomial time solvable in case 3.
- (Dichotomy Theorem for $\mathrm{SEP}(\mathbb{A})$.) Observe that $\mathrm{SEP}(\mathbb{A})$ is **NP**-complete in cases 1 and 2, and polynomial time solvable in case 3.
- (Dichotomy Theorem for $(2, \mathcal{F})$-$\mathrm{Robust}(\Gamma)$.) Observe that $(2, \mathcal{F})$-$\mathrm{Robust}(\Gamma)$ is **NP**-complete for some $\mathcal{F}$ in cases 1 and 2, and polynomial time solvable for all $\mathcal{F}$ in case 3.

4 Main results: an algebraic approach

A pivotal development in the classification of fixed template CSP complexity was the introduction of universal algebraic methods, starting with the work of Jeavons [14], Jeavons, Cohen, Gyssens [15], with the full framework presented in Bulatov, Jeavons, Krokhin [6]. The algebraic method is fundamental in Bulatov's classification of tractable CSPs on 3-element domains [7], and of list homomorphism complexity [8], in the classification of tractable CSPs over digraphs without sources and sinks [4], the classification of CSPs solvable by local consistency check algorithm [2, 3], amongst others.

The algebraic approach concerns the analysis of *polymorphisms* of the template. For some computational problems, polymorphism analysis appears too coarse; this is true for problems considered in Schnoor and Schnoor [21] as well as the problems considered in the present article. Following [21], our results are based on methods relating to partial polymorphisms.

▶ **Definition 7.** Let $k, n \in \mathbb{N}$, let $f\colon \operatorname{dom}(f) \to A$ be a n-ary partial operation, where $\operatorname{dom}(f) \subseteq A^n$, and let r be a k-ary relation on the set A. We say that f *preserves* r or r is *invariant under* f or f is a *partial polymorphism* of r, if whenever $a_1 = (a_{11}, \dots, a_{1n}), a_2 = (a_{21}, \dots, a_{2n}), \dots, a_k = (a_{k1}, \dots, a_{kn})$ are tuples in $\operatorname{dom}(f)$, then

$$(\forall i \in \{1, \dots n\}\ (a_{1i}, a_{2i}, \dots a_{ki}) \in r) \implies (f(a_1), f(a_2), \dots, f(a_k)) \in r.$$

If f is a total operation, then f is called a *polymorphism* of r. If F is set of partial operations then we say that r is *invariant under* F if r is invariant under every operation in F. We let $\mathcal{P}_A$ be the set of all non-empty, non-nullary finitary partial operations on A and $\mathcal{R}_A$ be the set of all non-empty, non-nullary finitary relations on A. Define

$$\mathrm{pPol}(R) := \{f \in \mathcal{P}_A \mid f \text{ preserves each } r \in R\}$$

for any set $R \subseteq \mathcal{R}_A$.

The following theorems are analogous to some of the main contributions in Bulatov et al. [6, Theorems 5.2 & 5.4], but now in the context of SEP and $(k, \mathcal{F})$-Robust and the algebra of partial polymorphisms.

For any partial algebra $\mathbf{A}$ we let $\mathsf{HS}(\mathbf{A})$ be the smallest class of partial algebras in the same signature closed under the formation of homomorphic images (H) and subalgebras (S).

▶ **Theorem 8** (HS Theorem). *Let $\mathbb{A} = \langle A; R\rangle$ and $\mathbb{B} = \langle B; S\rangle$ be templates and let $\mathcal{F}$ be a finite set of pp-formulæ in the language of R. If S satisfies* $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}\cap(k,\mathcal{F})}, \mathrm{N}_{\mathrm{CSP}})$ *and there exist partial algebras $\mathbf{A} = \langle A; F^{\mathbf{A}}\rangle$ and $\mathbf{B} = \langle B; F^{\mathbf{B}}\rangle$ such that*

1. $F^{\mathbf{B}} \subseteq \mathrm{pPol}(S)$,
2. $\mathbf{B} \in \mathsf{HS}(\mathbf{A})$, *and*
3. $\mathrm{pPol}(R) \subseteq F^{\mathbb{A}}$,

then R satisfies $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}\cap(k,\mathcal{F})}, \mathrm{N}_{\mathrm{CSP}})$.

The case where $\mathbf{A} = \mathbf{B}$ corresponds to the preservation of complexity of the gap property under conjunct atomic reductions, which is critical to the proof of the Gap Trichotomy Theorem. When $\mathbf{A} \neq \mathbf{B}$, the theorem lifts the complete classification given by the Gap Trichotomy Theorem on Boolean domains to many problems on templates with non-Boolean domains. With further effort, direct products can be incorporated into Item 2 of Theorem 8, but the full version is beyond the scope of the present article, and will appear in subsequent work.

In the full version of the present article, we show that HS theorems can be obtained for other variants of the constraint satisfaction problem, namely the equivalence problem and the implication problem, but whose definitions are not given due to space constraints.

A further useful simplification in the standard CSP setting has been the restriction to so-called idempotent polymorphisms; see [6, Theorem 4.7]. A partial polymorphism $f : \operatorname{dom}(f) \to \mathbb{A}$ is *idempotent* if $f(a, \dots, a) = a$ for every $a \in A$ for which $(a, \dots, a) \in \operatorname{dom}(f)$. As a final result we show that when analysing the complexity of SEP problems we may restrict to idempotent partial polymorphisms. The full statement of this theorem can be found in the complete version of the present article.

5 Weak co-clones and strong partial clones

We now give more technical definitions that are required for the main arguments.

▶ **Definition 9.** Let $m, n \in \mathbb{N}$, let $f \in \mathcal{P}_A$ be m-ary and let $g_1, \dots, g_m \in \mathcal{P}_A$ be n-ary. The *composition* $f(g_1, \dots, g_m)$ is an n-ary partial operation defined by

$$f(g_1, \dots, g_m)(x_1, \dots, x_n) := f(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)),$$

where $\mathrm{dom}(f(g_1, \dots, g_m))$ is the set

$$\left\{(x_1, \dots, x_n) \in \bigcap_{i=1}^{m} \mathrm{dom}(g_i) \mid (g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)) \in \mathrm{dom}(f)\right\}.$$

▶ **Definition 10.** Let $f, g \in \mathcal{P}_A$. We say that f is a *restriction* of g if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and f agrees with g on $\mathrm{dom}(f)$.

We let $\mathcal{O}_A$ denote the subset of $\mathcal{P}_A$ consisting of total operations.

▶ **Definition 11.** Let A be a non-empty set and let $\mathcal{C} \subseteq \mathcal{O}_A$. Then $\mathcal{C}$ is a *clone* on the set A if the following two conditions hold:

1. $\mathcal{C}$ contains all projection operations: that is, for all $n \in \mathbb{N}$, the ith projection $\pi_i \colon A^n \to A$ given by $\pi_i(x_1, \dots, x_n) = x_i$ belongs to $\mathcal{C}$;
2. $\mathcal{C}$ is closed under compositions.

For a set F of total operations, $[F]$ will denote the smallest clone containing F and we refer to $[F]$ as the *clone generated by* F. The set F is sometimes called a *base* for the clone $[F]$.

▶ **Definition 12.** Let A be a non-empty set. A subset R of $\mathcal{R}_A$ is called a *co-clone* or *relational clone* if it is closed under the formation of pp-definable relations. We define $\langle R \rangle$ to be the smallest co-clone containing R and we refer to $\langle R \rangle$ as the *co-clone generated by* R. The set R is sometimes called a *base* for $\langle R \rangle$.

The sets $\wp(\mathcal{O}_A)$ and $\wp(\mathcal{R}_A)$ are complete lattices, where $\wp()$ is the powerset operator. A well-known result of Geiger [11] states that pair of maps $\mathrm{Inv} \colon \wp(\mathcal{O}_A) \to \wp(\mathcal{R}_A)$ and $\mathrm{Pol} \colon \wp(\mathcal{R}_A) \to \wp(\mathcal{P}_A)$ form a Galois correspondence between $\wp(\mathcal{O}_A)$ and $\wp(\mathcal{R}_A)$. In particular, we have

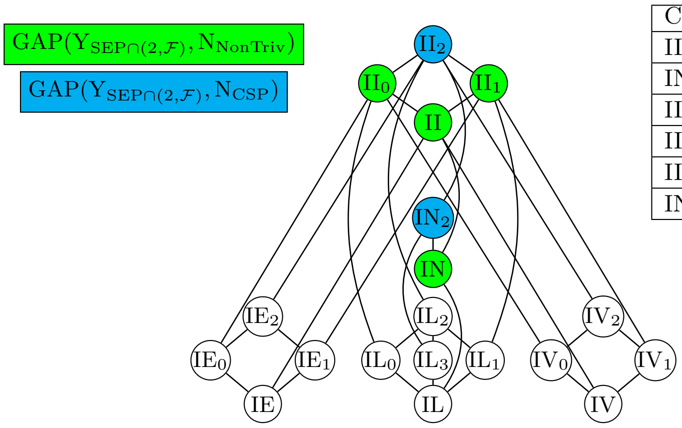
$$\begin{aligned}&\mathrm{Inv}(F) := \{r \in \mathcal{R}_A \mid r \text{ is invariant under each } f \in F\} \text{ and}\\&\mathrm{Pol}(R) := \{f \in \mathcal{O}_A \mid f \text{ preserves each } r \in R\},\end{aligned}$$

for each $F \subseteq \mathcal{O}_A$ and each $R \subseteq \mathcal{R}_A$.

Clones on $\{0, 1\}$ were characterised by Post [18] and are usually called "Boolean clones". An upset of Post's lattice is given in Figure 1; the table included gives definitions of the shaded vertices in terms of relations invariant under basic operations. The operations c_0 and c_1 are the constant unary functions to 0 and 1, respectively, and $\neg$ is the usual negation operation on $\{0, 1\}$. Shaded vertices in Figure 1 give the precise information for the Gap Trichotomy Theorem:

- Statement 1 applies when Γ generates a co-clone containing IN_2(blue/dark grey);
- Statement 2 applies when Γ generates a co-clone containing IN, but not containing IN_2 (green/light grey);
- Statement 3 holds otherwise.

In general, it seems difficult to use pp-formulæ to transfer the complexity of problems such as SEP and $(k, \mathcal{F})$-Robust. Instead we use conjunct-atomic formulæ.


GAP(Y$_{\mathrm{SEP}\cap(2,\mathcal{F})}$, N$_{\mathrm{NonTriv}}$)
GAP(Y$_{\mathrm{SEP}\cap(2,\mathcal{F})}$, N$_{\mathrm{CSP}}$)
II$_2$
II$_0$
II$_1$
II
IN$_2$
IN
IE$_2$
IL$_2$
IV$_2$
IE$_0$
IE$_1$
IL$_0$
IL$_3$
IL$_1$
IV$_0$
IV$_1$
IE
IL
IV

Co-clone	Definition
II$_2$	all Boolean relations
IN$_2$	$\mathrm{Inv}(\{\neg\})$
II$_0$	$\mathrm{Inv}(\{c_0\})$
II$_1$	$\mathrm{Inv}(\{c_1\})$
II	$\mathrm{Inv}(\{c_0, c_1\})$
IN	$\mathrm{Inv}(\{\neg, c_0\})$

Figure 1 An upset in the Boolean co-clone lattice, with a table of polymorphism definitions for the shaded co-clones; I$\mathcal{C}$ abbreviates $\mathrm{Inv}(\mathcal{C})$, for each Boolean clone $\mathcal{C}$.

▶ **Definition 13.** A subset R of $\mathcal{R}_A$ is called a *weak co-clone* or *weak system* if it is closed under the formation of conjunct-atomic definable relations. We can define $\langle R\rangle_{\not\exists}$ to be the smallest weak co-clone containing R and we refer to $\langle R\rangle_{\not\exists}$ as the *weak co-clone generated by* R. The set R is sometimes called a *base* for the weak system $\langle R\rangle_{\not\exists}$.

If we restrict further to conjunct atomic formulæ without equality, then we write instead $\langle R\rangle_{\not\exists,\neq}$ for the smallest system containing R and say that $\langle R\rangle_{\not\exists,\neq}$ is the *equality-free weak system generated by* R.

If we weaken the operators Inv and Pol to allow partial operations to be included in the definition, we obtain a refined Galois connection between the complete lattices $\wp(\mathcal{P}_A)$ and $\wp(\mathcal{R}_A)$ (see Romov [19]). In particular, sets of the form $\mathrm{Inv}(F)$ are precisely the weak co-clones, for $F \subseteq \mathcal{P}_A$. Sets of the form $\mathrm{pPol}(R)$, for $R \subseteq \mathcal{R}_A$, are called *strong partial clones*, and coincide with those subsets of $\mathcal{P}_A$ including all total projections and that are closed under composition and domain restriction. Post's lattice provides a useful approximation to the lattice of strong partial clones in the Boolean setting: for each Boolean clone $\mathcal{C}$, is it known that the set of all strong partial clones whose total operations agree with $\mathcal{C}$ forms an interval, and there are known generators for the top and bottom element in each of these intervals [21]; these are critical in the main proofs to come.

▶ **Definition 14.** Let A be a non-empty set, let $\mathcal{C}$ be a clone on A and let Γ be a set of finitary relations on A. We call Γ a *weak base* for the co-clone $\mathrm{Inv}(\mathcal{C})$ if $\mathcal{I}_\cup(\mathcal{C}) = \mathrm{pPol}(\Gamma)$.

We will often present relations in a matrix form. The representation is not unique, but it is succinct. For a k-ary relation $r = \{a_1, \dots, a_m\}$ on a non-empty set A with $|r| = m$, the *matrix representation of* r is the $m \times k$ matrix $M = (a_{ij})$ over A whose ith row is the tuple a_i. (Non-uniqueness follows because the ordering $a_1, \dots, a_m$ is arbitrary.)

▶ **Definition 15.** Define Cols_3 to be the following 8-ary relation over $\{0, 1\}$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

▶ **Definition 16.** Let $\mathcal{C}$ be a Boolean clone and let r be a relation on A. Define $\mathcal{C}(r)$ to be the smallest relation containing r that is invariant under every operation in $\mathcal{C}$. We refer to $\mathcal{C}(r)$ as the $\mathcal{C}$-*closure* of the relation r.

Using the work of Schnoor and Schnoor [21] and Schnoor [22, Table 3.1], the following construction gives weak-bases for each of the Boolean co-clones shaded in Figure 1.

▶ **Proposition 17** ([21, Theorem 4.11], [22, Table 3.1])**.** *Let* $\mathrm{I}\mathcal{C}$ *be any of the Boolean co-clones listed in the table within Figure 1. Then* $\mathcal{C}(\mathrm{Cols}_3)$ *is a weak-base for* $\mathrm{I}\mathcal{C}$*.*

For example, to construct a weak base for the Boolean co-clone $\mathrm{IN}_2 = \mathrm{Inv}(\{\neg\})$, we simply close the relation Cols_3 under $\neg$. Thus,

$$\mathrm{N}_2(\mathrm{Cols}_3) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Weak bases that generate not only the smallest weak system but also the smallest equality-free weak-system of relations generating the same co-clone will be crucial in the classification of the problems considered. Schnoor and Schnoor [21, Definition 5.1] give an irredundancy condition ensuring conjunct-atomic definability without equality. We omit the definition, but observe that all six of the relations in Proposition 17 are irredundant.

▶ **Theorem 18** ([21, Corollary 5.6])**.** *Let* A *be a non-empty set, let* $\mathcal{C}$ *be a clone on* A *and let* Γ *be an irredundant weak base for the co-clone* $\mathrm{Inv}(\mathcal{C})$*. If* Γ' *is set of relations on* A *such that* $\langle\Gamma'\rangle = \mathrm{Inv}(\mathcal{C})$*, then* $\langle\Gamma\rangle_{\nexists,\neq} \subseteq \langle\Gamma'\rangle_{\nexists,\neq}$*.*

The next two sections are dedicated to establishing the Gap Trichotomy Theorem.

6 Towards a dichotomy: gap properties

We begin with three results that are crucial for establishing gap properties. The first result is an abridged version of [13, Theorem 6.1]. We let $\mathbb{2}$ denote the positive 1-in-3 SAT template $\langle\{0,1\}; \{(1,0,0),(0,1,0),(0,0,1)\}\rangle$.

▶ **Theorem 19** ([13])**.** *Let* $\mathcal{K}$ *be the set consisting of all positive* 1*-in-*3 SAT *instances* I *with the following properties*:

- *no variable appears more than once in each constraint tuple of* I*,*
- I *is* 2*-robustly positive* 1*-in-*3 *satisfiable.*

Then the positive 1*-in-*3 SAT *relation has* $\mathrm{GAP}(\mathcal{K}, \mathrm{N}_{\mathrm{CSP}})$*.*

The next lemma summarises the basic method employed in Abramsky, Gottlob and Kolaitis [1] and Jackson [13]. It is essentially the definition of reduction for promise problems; see [12, Definition 3], for example.

▶ **Lemma 20.** *Let* Γ *and* Γ' *be finite sets of relations on* $\{0,1\}$*. Let* A *and* B *be disjoint sets of* Γ*-instances and let* X *and* Y *be disjoint sets of* Γ'*-instances. Further, let* Γ *have the gap property* $\mathrm{GAP}(A,B)$*. If there is a polynomial-time computable function* $f\colon \mathcal{I}_\Gamma \to \mathcal{I}_{\Gamma'}$ *satisfying:*

1. $I \in A \Rightarrow f(I) \in X$,
2. $I \in B \Rightarrow f(I) \in Y$,

then Γ' *satisfies* $\mathrm{GAP}(X,Y)$*. In particular,* Γ' *has the gap property* $\mathrm{GAP}(f(A), f(B))$*.*

It is well known that the complexity of CSP(Γ) depends only on the co-clone generated by Γ, see [14, Theorem 3.4] or alternatively [6, Theorem 2.16] for a proof explicitly using pp-formulæ. We now give an analogous result that says when analysing the the complexity of SEP and $(2, \mathcal{F})$-robust satisfiability we need only consider relations up to equality-free conjunct-atomic definability. We first give a preliminary lemma.

▶ **Lemma 21.** *Let $\Gamma^{\mathbb{A}}$ be a constraint language over a finite set A and let $R^{\mathbb{A}}$ be a finite set of relations in $\langle \Gamma^{\mathbb{A}} \rangle_{\not\exists}$. There is a polynomial-time construction that transforms any instance $I = (V; A; \mathcal{C})$ of $\mathrm{CSP}(R^{\mathbb{A}})$ into an instance I' of $\mathrm{CSP}(\Gamma^{\mathbb{A}})$ on the same variables, and moreover, the solutions of I are exactly the solutions of I'.*

▶ **Theorem 22.** *Let $\Gamma^{\mathbb{A}}$ be a constraint language over a set A, let $R^{\mathbb{A}}$ be any finite set of relations in $\langle \Gamma^{\mathbb{A}} \rangle_{\not\exists, \neq}$, let $\mathcal{F}$ be a finite set of pp-formulæ in the language of R and let $k \in \mathbb{N}$. There is a polynomial-time computable function that reduces*

1. $\mathrm{CSP}(R^{\mathbb{A}})$ *to* $\mathrm{CSP}(\Gamma^{\mathbb{A}})$*,*
2. $\mathrm{SEP}(R^{\mathbb{A}})$ *to* $\mathrm{SEP}(\Gamma^{\mathbb{A}})$*, and*
3. $(k, \mathcal{F})\text{-Robust}(R^{\mathbb{A}})$ *to* $(k, \mathcal{G})\text{-Robust}(\Gamma^{\mathbb{A}})$*, for some finite set $\mathcal{G}$ of pp-formulæ in the language of Γ.*

Proof. The reduction from $\mathrm{CSP}(R^{\mathbb{A}})$ to $\mathrm{CSP}(\Gamma^{\mathbb{A}})$ is obtained immediately from Lemma 21. This proves (1).

Since the solutions of I in $\mathrm{CSP}(R^{\mathbb{A}})$ are precisely the solutions of I' in $\mathrm{CSP}(\Gamma^{\mathbb{A}})$, it follows that separating solutions of I are exactly the separating solutions of I'. Hence I is a YES instance of $\mathrm{SEP}(R^{\mathbb{A}})$ if and only if I' is a YES instance of $\mathrm{SEP}(\Gamma^{\mathbb{A}})$. This establishes (2).

For (3), consider $r \in R$ of arity ℓ and abstractly expressible by an equality-free conjunct-atomic formula $r(x_1, \dots, x_\ell)$ in the language of Γ. For each pp-formula $\rho(w_1, \dots, w_m) \in \mathcal{F}$, we construct a pp-formula $\rho_\Gamma(w_1, \dots, w_m)$ in the language of Γ in the following way: replace every occurrence of an ℓ-ary relation symbol r in ρ with its conjunct-atomic defining formula $r(x_1, \dots, x_\ell)$. Let $\mathcal{G} = \{\rho_\Gamma \mid \rho \in \mathcal{F}\}$. Then since $\rho(a_1, \dots, a_m)$ is true in $\langle A; R^{\mathbb{A}} \rangle$ if and only if $\rho_\Gamma(a_1, \dots, a_m)$ is true in $\langle A; \Gamma^{\mathbb{A}} \rangle$ for $(a_1, \dots, a_m) \in A^m$ and $\rho(w_1, \dots, w_m) \in \mathcal{F}$, it follows that the $\mathcal{F}$-compatible assignments on k variables of I are exactly the $\mathcal{G}$-compatible assignments on k variables of I'. Thus, since the solutions of I are precisely the solutions of I', by Lemma 21, it then follows that I is a YES instance of $(k, \mathcal{F})\text{-Robust}(R^{\mathbb{A}})$ if and only if I' is a YES instance of $(k, \mathcal{G})\text{-Robust}(\Gamma^{\mathbb{A}})$. ◀

Theorem 22 holds more generally: with some caveats and proper amendment to the proof, the assumption that $R \subseteq \langle \Gamma \rangle_{\not\exists, \neq}$ can be weakened to $R \subseteq \langle \Gamma \rangle_{\not\exists}$. However, this result is not required for establishing our main theorems.

7 Proof of the Gap Trichotomy Theorem

In this section, we establish gap properties for relations generating the Boolean co-clones II_2, IN_2, II_0, II_1, II or IN. These co-clones are shaded in Figure 1. Each co-clone must be considered separately, however the proofs follow the same structure: we first establish a gap property for the irredundant weak-base and then use the fact that gap properties are preserved by the $\langle - \rangle_{\not\exists, \neq}$-operator. We sketch details only in the case of II_2.

7.1 The Boolean co-clone II_2 and IN_2

By Proposition 17, the relation $\mathrm{I}_2(\mathrm{Cols}_3)$ of Definition 15 is an irredundant weak base for II_2.

▶ **Proposition 23.** *The relation* $\mathrm{I}_2(\mathrm{Cols}_3)$ *satisfies* $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}\cap\text{2-Rob}}, \mathrm{N}_{\mathrm{CSP}})$.

Proof outline. We apply Lemma 20, reducing from $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}\cap\text{2-Rob}}, \mathrm{N}_{\mathrm{CSP}})$ for positive 1-in-3SAT. The result will then follow from Theorem 19.

Given an instance $I = (V; \{0,1\}; \mathcal{C})$ of positive 1-in-3 SAT, construct an instance $I^\star = (V^\star; \{0,1\}; \mathcal{C}^\star)$ of II_2 - SAT in the following way.

1. First let $\overline{V} = \{\bar{v} \mid v \in V\}$ be a disjoint copy of V, and construct $V^\star = V \cup \overline{V} \cup \{\top, \bot\}$, where $\top, \bot \notin V \cup \overline{V}$,
2. for each constraint $\langle (x,y,z), +1\mathrm{in}3\,\mathrm{SAT}\rangle$ in $\mathcal{C}$, we include the constraint $\langle (x,y,z,\bar{x},\bar{y},\bar{z},\bot,\top), \mathrm{I}_2(\mathrm{Cols}_3)\rangle$ in $\mathcal{C}^\star$.

Any solution φ of I in CSP(2) can be extended to a solution $\varphi^\star$ of $I^\star$ in the following way. For each $v \in V$, define $\varphi^\star(v) := \varphi(v)$, $\varphi^\star(\bar{v}) = \neg \circ \varphi(v)$, $\varphi^\star(\bot) = 0$ and $\varphi^\star(\top) = 1$, where $\neg$ is the usual Boolean complement. For the converse direction, observe that the projection $\pi_{\{1,2,3\}}(\mathrm{I}_2(\mathrm{Cols}_3)) = +1\mathrm{in}3\,\mathrm{SAT}$, thus if ψ is a solution of $I^\star$ in II_2 - SAT, then the restriction $\psi{\restriction_V}$ is a solution of I in CSP(2). Hence we have shown that any solution ϕ of I extends uniquely to a solution $\phi^\star$ of $I^\star$.

Now assume that I lies in the class $\mathcal{K}$ of Theorem 19. Since I is 2-robustly satisfiable and no variable appears more than once in each constraint tuple, it follows that I has the following properties.

(♡) For every pair of distinct variables x and y in V, there are solutions φ_1, φ_2 and φ_3 of I satisfying

$$(\varphi_1(x), \varphi_1(y)) = (0,0),$$
$$(\varphi_2(x), \varphi_2(y)) = (0,1),$$
$$(\varphi_3(x), \varphi_3(y)) = (1,0).$$

(♢) If x and y do not appear in a common constraint tuple, then there is a solution φ_4 of I satisfying $(\varphi_4(x), \varphi_4(y)) = (1,1)$.

These conditions can be used to show that $I^\star \in \mathrm{Y}_{\mathrm{SEP}\cap\text{2-Rob}}(\mathrm{II}_2\text{ - SAT})$: there are a number of cases according to the different combinations of containments in $V, \overline{V}, \{\top, \bot\}$ for a given pair of variables $\{u, v\} \subseteq V^\star$. ◀

The following theorem now follows from Theorems 18 and 22.

▶ **Theorem 24.** *Let* Γ *be a finite constraint language on* $\{0,1\}$ *such that* $\langle\Gamma\rangle = \mathrm{II}_2$. *Then* Γ *has the gap property* $\mathrm{GAP}(\mathrm{Y}_{\mathrm{SEP}\cap(2,\mathcal{G})}, \mathrm{N}_{\mathrm{CSP}})$, *for some finite set* $\mathcal{G}$ *of pp-formulæ in the language of* Γ, *and consequently both* $(2,\mathcal{G})$-Robust(Γ) *and* SEP(Γ) *are* **NP**-*complete.*

A similar approach gives an analogous theorem for constraint languages generating IN_2: the reduction can be taken from either of positive 1-in-3SAT directly, or by following from the reduction in Proposition 23. Together with Theorem 24, these two cases cover Statement 1 of the Gap Trichotomy Theorem 6.

7.2 The Boolean co-clones with trivial CSPs but hard $\mathrm{CSP}_{\mathrm{NonTriv}}$

This section relates to the clones $\mathrm{II}_0, \mathrm{II}_1, \mathrm{II}, \mathrm{IN}$ corresponding to Statement 2 in the Gap Dichotomy Theorem 6. In these cases, we can reuse the same fundamental construction used for II_2 and IN_2. The proof proceeds as follows: to lie in $\mathrm{Y}_{\mathrm{SEP}}$ or $\mathrm{Y}_{(2,\mathcal{F})}$, it is necessary to have an assignment in which $\bot$ and $\top$ take different values. We then argue that this forces solutions into II_2 or IN_2.

7.3 Proving tractablity

We establish a theorem that covers all cases that are solvable in polynomial-time. The proof relies on a result of Jackson [13, Proposition 3.2], which says that a constraint language Γ on a finite set A is polynomial-time equivalent to Γ_{Con}, with respect to Turing reductions, for each of $\mathrm{SEP}(\mathbb{A})$ and $(k, \mathcal{F})$-Robust.

▶ **Theorem 25.** *Let Γ be a constraint language on $\{0, 1\}$. If $\mathrm{IN} \not\subseteq \langle \Gamma \rangle$, then the computational problems $\mathrm{SEP}(\Gamma)$ and $(2, \mathcal{F})$-Robust(Γ) are solvable in polynomial-time.*

Proof. When $\mathrm{IN} \not\subseteq \langle \Gamma \rangle$, it follows from Post's co-clone lattice (see Figure 1), that $\mathrm{IN} \not\subseteq \langle \Gamma \cup \{(0), (1)\} \rangle$ and then it is known that the constraint problem $\mathrm{CSP}(\Gamma \cup \{(0), (1)\})$ is tractable; this can be found in Schaefer's original argument for example; see [20, Lemma 4.1]. Then from [13, Proposition 3.2], the problems $\mathrm{SEP}(\Gamma)$ and $(2, \mathcal{F})$-Robust(Γ) are solvable in polynomial-time. ◀

8 Proof of the HS Theorem

We give a brief overview of the proof for the HS Theorem. The result is established by carrying the gap property through items 1, 2 and 3 of Theorem 8. The main difficulties arise from items 2 (HS) and 3 (restricted pp-definability), requiring a series of polynomial-time reductions. The constructions used for substructures and homomorphisms are based on those in [6, 16], given in the standard CSP setting. In the case of taking substructures, SEP and $(2, \mathcal{F})$-Robust carry through using the standard construction (the local compatibility condition $\mathcal{F}$ is changed during the reduction). The homomorphism case however requires proper amendment, including the addition of extra relations and non-trivial usage of the gap property; the main complication arising from SEP. For item 3, we require a more general version of Theorem 22. The addition of equality presents complications for $(2, \mathcal{F})$-robust satisfiability, and we again require the use of a gap property to carry through the reduction. The proof for SEP is similar however, a further slight variation of the proof is necessary.

Acknowledgements. The author is indebted to Marcel Jackson for many useful suggestions and discussions, as well as her PhD supervisors Brian Davey and Tomasz Kowalski, for their guidance and feedback.

References

1 Samson Abramsky, Georg Gottlob, and Phokion G. Kolaitis. Robust constraint satisfaction and local hidden variables in quantum mechanics. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

2 Libor Barto and Marcin Kozik. Robust satisfiability of constraint satisfaction problems. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19-22, 2012*, pages 931–940, 2012.

3 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, 2014.

4 Libor Barto, Marcin Kozik, and Todd Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (A positive answer to a conjecture of bang-jensen and hell). *SIAM J. Comput.*, 38(5):1782–1802, 2009.

5 Adam Beacham and Joseph C. Culberson. On the complexity of unfrozen problems. *Discrete Applied Mathematics*, 153(1-3):3–24, 2005.

6 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

7 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120, January 2006.

8 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Logic*, 12(4):24:1–24:66, July 2011.

9 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34, 2013.

10 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

11 David Geiger. Closed systems of functions and predicates. *Pacific J. Math.*, 27:95–100, 1968.

12 Oded Goldreich. On promise problems: A survey. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, pages 254–290, 2006.

13 M. Jackson. Flexible constraint satisfiability and a problem in semigroup theory. *ArXiv e-prints*, 2015. `arXiv:1512.03127`.

14 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, 1998.

15 Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

16 Benoit Larose and Pascal Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theor. Comput. Sci.*, 410(18):1629–1647, 2009.

17 R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic phase transitions. *Nature*, 400:133–137, 1998.

18 E. L. Post. *The two-valued iterative systems of mathematical logic*, volume no. 5 of *Annals of Mathematics studies*. Princeton University Press, Princeton, 1941.

19 B. A. Romov. The algebras of partial functions and their invariants. *Cybernetics*, 17:157–167, 1981.

20 Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978.

21 Henning Schnoor and Ilka Schnoor. Partial polymorphisms and constraint satisfaction problems. In *Complexity of Constraints – An Overview of Current Research Themes [Result of a Dagstuhl Seminar].*, pages 229–254, 2008.

22 Ilka Schnoor. *The weak base method for constraint satisfaction.* PhD thesis, University of Hanover, 2008.

23 Edward P. K. Tsang. *Foundations of constraint satisfaction.* Computation in cognitive science. Academic Press, 1993.

Sliding Tokens on a Cactus*

Duc A. Hoang[1] **and Ryuhei Uehara**[2]

1 **Japan Advanced Institute of Science and Technology, Japan**
`hoanganhduc@jaist.ac.jp`

2 **Japan Advanced Institute of Science and Technology, Japan**
`uehara@jaist.ac.jp`

Abstract

Given two independent sets **I** and **J** of a graph G, imagine that a token (coin) is placed on each vertex in **I**. Then, the Sliding Token problem asks if one could transforms **I** to **J** using a sequence of elementary steps, where each step requires sliding a token from one vertex to one of its neighbors, such that the resulting set of vertices where tokens are placed still remains independent. In this paper, we describe a polynomial-time algorithm for solving Sliding Token in case the graph G is a cactus. Our algorithm is designed based on two observations. First, all structures that forbid the existence of a sequence of token slidings between **I** and **J**, if exist, can be found in polynomial time. A no-instance may be easily deduced using this characterization. Second, without such forbidden structures, a sequence of token slidings between **I** and **J** does exist.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases reconfiguration problem, token sliding, independent set, cactus

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.37

1 Introduction

A reconfiguration problem arises when we wish to find a step-by-step transformation between two feasible solutions of a problem. In each transformation, each intermediate result is also feasible, and each transformation step abides by a fixed reconfiguration rule. The *reconfiguration problems* attract the attention recently from the viewpoint of theoretical computer science, and have been studied extensively for several well-known problems, including satisfiability [8, 12], independent set [9, 10, 11, 14], set cover, clique, matching [10], and so on. For an overview of this research area, we refer the readers to [17].

Although the problems above might seem to be artificial, from the viewpoint of recreational mathematics, the reconfiguration problems have already been played long time, and partially well investigated. One of the most famous classic examples is the so-called *15 puzzle* (see Figure 1). If rectangles are allowed, we obtain a more general classic puzzle called "sliding block puzzle" and its variants (see Figure 1). In 1964, Gardner said that "These puzzles are very much in want of a theory" [7]. After 40 years, Hearn and Demaine gave the theory. Using their proposed *nondeterministic constraint logic* model [9], they proved that the general sliding block puzzle is PSPACE-complete, while it is linear time solvable if all pieces are unit squares. We remind that finding an optimal solution is NP-complete for yes-instance of this linear time solvable case. In this way, we can characterize three

* This work was partially supported by MEXT/JSPS Kakenhi (Grant Numbers 26330009 and 24106004).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 37; pp. 37:1–37:26
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

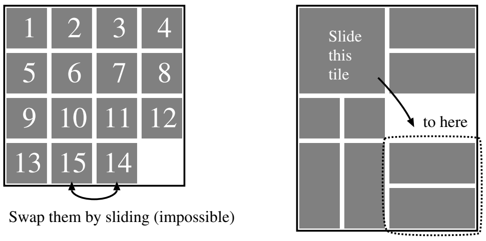


Figure 1 The 15 puzzle and sliding block puzzle.

familiar complexity classes P, NP, and PSPACE using the model of the sliding block puzzle, a representative reconfiguration problem.

From the viewpoint of theoretical computer science, one of the most important problems is the 3SAT. Even in the reconfiguration problem, the computational complexity of the 3SAT has been investigated, and it is shown to be PSPACE-complete [8]. Recently, for the 3SAT, an interesting trichotomy for the complexity of finding a shortest sequence has been shown; that is, for the reconfiguration problem, finding a shortest sequence between two satisfiable assignments is in P, NP-complete, or PSPACE-complete in certain conditions [13]. In general, the reconfiguration problems tend to be PSPACE-complete, and some polynomial time algorithms are shown in restricted cases. However, we have to mind that it may potentially have different computational complexity for deciding two configurations are reconfigurable, for finding a sequence of feasible solutions between two configurations, or for finding a shortest sequence of feasible solutions between two configurations. Especially, since some problems are PSPACE-complete, we may have some case that the length of the sequence of solutions can be super-polynomial even if the decision problem is in NP.

Beside the 3SAT, one of the most important problems in theoretical computer science is the INDEPENDENT SET problem. For this notion, the natural reconfiguration problem is called the SLIDING TOKEN problem introduced by Hearn and Demaine [9]: Suppose that we are given two independent sets $\mathbf{I}$ and $\mathbf{J}$ of a graph $G = (V, E)$ and imagine that a token (or coin) is placed on each vertex in $\mathbf{I}$. Then, the SLIDING TOKEN problem asks if there exists a sequence $\mathcal{S} = \langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ of independent sets of G such that (a) $\mathbf{I}_1 = \mathbf{I}$, $\mathbf{I}_\ell = \mathbf{J}$, and $|\mathbf{I}| = |\mathbf{I}_i|$ for all i with $1 \leq i \leq \ell$; and (b) for each i, $2 \leq i \leq \ell$, there is an edge uv in E such that $\mathbf{I}_{i-1} \setminus \mathbf{I}_i = \{u\}$ and $\mathbf{I}_i \setminus \mathbf{I}_{i-1} = \{v\}$. If such a sequence $\mathcal{S}$ exists, we call $\mathcal{S}$ a TS-*sequence* and say that $\mathcal{S}$ *reconfigures* $\mathbf{I}$ to $\mathbf{J}$ in G and write $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$. Figure 2 illustrates a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_5 \rangle$ of independent sets which reconfigures $\mathbf{I} = \mathbf{I}_1$ into $\mathbf{J} = \mathbf{I}_5$. Hearn and Demaine proved that the SLIDING TOKEN problem is PSPACE-complete for planar graphs as an example of the application of their nondeterministic constraint logic model, which can be used to prove PSPACE-hardness of many puzzles and games [9]. (We note that the reconfiguration problem for INDEPENDENT SET has some variants. In [11], the reconfiguration problem for INDEPENDENT SET is studied under three reconfiguration rules called "token sliding," "token jumping," and "token addition and removal." In this paper, we only consider the token sliding model, and see [11] for the other models.)

For the SLIDING TOKEN problem, some polynomial-time algorithms have been shown recently for bipartite permutation graphs [6] and claw-free graphs [2]. Linear-time algorithms have been shown for cographs [11] and trees [4]. Even a shortest TS-sequence can be found in polynomial time for a caterpillar [18]. On the other hand, PSPACE-completeness is also shown for graphs of bounded tree-width [15] and planar graphs [9]. Recently, hardness results for split graphs, and polynomial-time algorithm for interval graphs have been annouced by Bonamy and Bousquet [1].

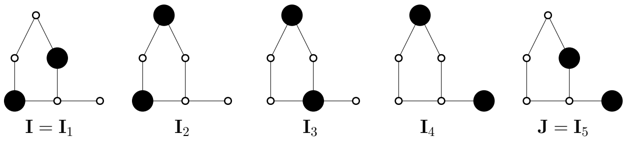


Figure 2 A sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \dots, \mathbf{I}_5 \rangle$ of independent sets of the same graph, where the vertices in independent sets are depicted by black circles (tokens).

In this paper, we give a polynomial-time algorithm for the Sliding Token problem for a cactus. Intuitively, a cactus is a graph that is obtained by joining cycles. When we solve the Sliding Token problem, there are three major points to be considered. First, we have to decide a correspondence between the tokens in $\mathbf{I}$ and $\mathbf{J}$. That is, we have to decide the goal in $\mathbf{J}$ for each token in $\mathbf{I}$, which is called *target-assignments*. Next, we design the route for each token. In some graph class, say, a tree, the second one is easy since any pair of vertices on a tree has unique path for joining them. However, even in this case, some token is required to make "detours" to open its position to admit other tokens to go through its neighbors (see [18] for the details). When the graph contains a cycle, since the route for a token is not unique any more, we have to "choose" the route. Therefore, for each token, we may have exponentially many choices and possibly super polynomial detours in general. Especially, if a graph contains an odd cycle, the Sliding Token problem is quite difficult.

The idea of our algorithm is to characterize all structures that forbid the existence of a TS-sequence between $\mathbf{I}$ and $\mathbf{J}$ first, and then prove the existence of a TS-sequence between them when no such forbidden structures exist. A trivial forbidden structure is clearly the sizes of $\mathbf{I}$ and $\mathbf{J}$, i.e., if $|\mathbf{I}| \neq |\mathbf{J}|$ then $\mathbf{I}$ cannot be reconfigured to $\mathbf{J}$ (and vice versa) using TS rule. In case of cacti, two more forbidden structures, named *rigid token* and *confined cycle*, are characterized (see Section 4). We claim that these structures (if exist) can be found in polynomial time. For a cactus that does not contain these forbidden structures, we show that a TS-sequence between $\mathbf{I}$ and $\mathbf{J}$ exists (Lemma 16). Despite of the non-trivial tasks of identifying forbidden structures and designing reconfiguration sequences, this technique was proved to be powerful for developing polynomial-time algorithms for solving several reconfiguration problems [3, 4, 6, 16].

In this paper, some proof details are omitted due to the space restriction. For the statements marked with $(*)$, one can find the corresponding proof details in the appendix.

2 Preliminaries

In this section, we define some notions that will be used in this paper. For the notions which are not mentioned here, the readers are referred to [5].

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. For a vertex v, let $N_G(v)$ be the set of all neighbors of v in G. Let $N_G[v] = N_G(v) \cup \{v\}$ and $\deg_G(v) = |N_G(v)|$. For a subset $X \subseteq V(G)$, we simply write $N_G[X] = \bigcup_{v \in X} N_G[v]$. For two vertices u, v, denote by $\mathsf{dist}_G(u, v)$ the length of a shortest uv-path in G. G is *connected* if any pair of vertices in G are joined by at least one path; otherwise, we say that G is *disconnected*. For $X \subseteq V(G)$, denote by $G[X]$ the subgraph of G induced by vertices of X. We write $G - X$ to indicate the graph $G[V(G) \setminus X]$. Similarly, for a subgraph H of G, we denote by $G - H$ the graph $G[V(G) \setminus V(H)]$, and we say that the graph $G - H$ is obtained by *removing H from G*. An *independent set* $\mathbf{I}$ of a graph G is a subset of $V(G)$ in which for every $u, v \in \mathbf{I}$, uv is not an edge of G. For a subgraph H of G, sometimes we write $\mathbf{I} \cap H$ and $\mathbf{I} - H$ to indicate the

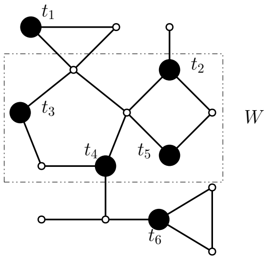


Figure 3 The tokens t_3 and t_5 are $(G, \mathbf{I}, W)$-confined, while t_2 and t_4 are not.

sets $\mathbf{I} \cap V(H)$ and $\mathbf{I} \setminus V(H)$, respectively. A vertex v of G is called a *cut vertex* if $G - v$ is disconnected; otherwise, we say that v is a *non-cut vertex*. A *block* of G is a maximal connected subgraph (i.e., a subgraph with as many edges as possible) with no cut vertex. G is called a *cactus* if every block of G is either K_2 or a simple cycle.

Let G be a graph and $\mathbf{I}$ an independent set of G. For a TS-sequence $\mathcal{S}$, we write $\mathbf{I} \in \mathcal{S}$ if $\mathbf{I}$ appears in $\mathcal{S}$. We say that $\mathcal{S}$ *involves* a vertex v if there exists some independent set $\mathbf{I} \in \mathcal{S}$ such that $v \in \mathbf{I}$. We say that $\mathcal{S} = \langle \mathbf{I}_1, \mathbf{I}_2, \dots, \mathbf{I}_\ell \rangle$ *slides* (or *moves*) the token t placed at $u \in \mathbf{I}_1$ to $v \notin \mathbf{I}_1$ in G if after applying the sliding steps described in $\mathcal{S}$, the token t is placed at $v \in \mathbf{I}_\ell$. Observe that a TS-sequence is *reversible*, i.e., $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ if and only if $\mathbf{J} \overset{G}{\leftrightsquigarrow} \mathbf{I}$. The *length* of a TS-sequence $\mathcal{S}$ is defined as the number of independent sets contained in $\mathcal{S}$.

One of the non-trivial structures that forbid the existence of a TS-sequence between any two independent sets of a graph is the so-called *rigid token*. Let $u \in \mathbf{I}$ be a vertex of G. The token t placed at u is called $(G, \mathbf{I})$-*rigid* if for any $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, $u \in \mathbf{J}$. The set of vertices where $(G, \mathbf{I})$-rigid tokens are placed is denoted by $\mathscr{R}(G, \mathbf{I})$. If t is not $(G, \mathbf{I})$-rigid, we say that it is $(G, \mathbf{I})$-*movable*. Decide if a token is $(G, \mathbf{I})$-rigid is PSPACE-complete for a general graph G [6].

Naturally, one can generalize the notion of rigid tokens in the following way. Let $W \subseteq V(G)$ be a subset of vertices of G. We say that t is $(G, \mathbf{I}, W)$-*confined* if for every $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, t is always placed at some vertex of W (see Figure 3). In other words, t can only be slid along edges of $G[W]$. Observe that a token t placed at some vertex $u \in \mathbf{I}$ is $(G, \mathbf{I})$-rigid if and only if it is $(G, \mathbf{I}, \{u\})$-confined.

Let H be an induced subgraph of G. H is called $(G, \mathbf{I})$-*confined* if $\mathbf{I} \cap H$ is a maximum independent set of H and all tokens in $\mathbf{I} \cap H$ are $(G, \mathbf{I}, V(H))$-confined. In particular, if H is a cycle (resp. a path) of G, we say that it is a $(G, \mathbf{I})$-*confined cycle* (resp. $(G, \mathbf{I})$-*confined path*). We denote by $\mathscr{C}(G, \mathbf{I})$ the set of all $(G, \mathbf{I})$-confined cycles of G. We will see later that $(G, \mathbf{I})$-confined cycles indeed form a structure that forbids the existence of a TS-sequence when G is a cactus. For a vertex $v \in V(H)$, we define G^v_H to be the (connected) component of G_H containing v, where G_H is obtained from G by removing all edges of H. Observe that if G is a cactus then for a cycle H of G and two distinct vertices $u, v \in V(H)$, $V(G^u_H) \cap V(G^v_H) = \emptyset$.

3 Some useful observations

In this section, we prove some useful observations. These observations will be implicitly used in many statements of this paper. The next lemma describes some equivalent conditions of being a $(G, \mathbf{I})$-confined induced subgraph, where $\mathbf{I}$ is a given independent set of a graph G. Intuitively, the structure of a $(G, \mathbf{I})$-confined induced subgraph H guarantees that the tokens *inside* (resp. *outside*) of H cannot be moved *out* (resp. *in*).

▶ **Lemma 1** (∗). *Let $\mathbf{I}$ be an independent set of a graph G. Let H be an induced subgraph of G. Then the following conditions are equivalent.*

(i) *H is $(G, \mathbf{I})$-confined.*

(ii) *For every independent set $\mathbf{J}$ satisfying $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, $\mathbf{J} \cap H$ is a maximum independent set of H.*

(iii) *$\mathbf{I} \cap H$ is a maximum independent set of H and for every $\mathbf{J}$ satisfying $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, any token t_x placed at $x \in \mathbf{J} \cap H$ is $(G^x_H, \mathbf{J} \cap G^x_H)$-rigid.*

The next proposition says that if the given graph G is not connected, then one can deal with each component separately.

▶ **Proposition 2** (∗). *Let $\mathbf{I}$, $\mathbf{J}$ be two given independent set of G. Assume that $G_1, \dots, G_k$ are the components of G. Then $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ if and only if $\mathbf{I} \cap G_i \overset{G_i}{\leftrightsquigarrow} \mathbf{J} \cap G_i$ for $i = 1, 2, \dots, k$.*

Thus, when dealing with Sliding Token, one can assume without loss of generality that the given graph is connected. Next, we claim that in certain conditions, a TS-sequence in a subgraph G' of G can be somehow "extended" to a sequence in G, and vice versa.

▶ **Proposition 3** (∗). *Let u be a vertex of a graph G. Let $\mathcal{S} = \langle \mathbf{I}_1, \mathbf{I}_2, \dots, \mathbf{I}_\ell \rangle$ be a* TS*-sequence in G such that for any $\mathbf{I}_i \in \mathcal{S}$, $u \in \mathbf{I}_i$, where $i \in \{1, 2, \dots, \ell\}$. Let $G' = G - N_G[u]$. Then $\mathbf{I}_1 \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{I}_\ell \cap G'$. Moreover, for any* TS*-sequence $\mathcal{S}' = \langle \mathbf{I}'_1, \dots, \mathbf{I}'_l \rangle$ in G', $\mathbf{I}'_1 \cup \{u\} \overset{G}{\leftrightsquigarrow} \mathbf{I}'_l \cup \{u\}$.*

Finally, we claim that if $\mathscr{R}(C, \mathbf{I}) = \mathscr{R}(C, \mathbf{J}) = \emptyset$, where C is a cycle and $\mathbf{I}$, $\mathbf{J}$ are two independent sets of C, then $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{J}$ if and only if $|\mathbf{I}| = |\mathbf{J}|$. In particular, if $\mathscr{R}(C, \mathbf{I}) = \emptyset$, starting from a given independent set $\mathbf{I}$, using token sliding, one can obtain any target independent set $\mathbf{J}$ of the same cardinality.

▶ **Lemma 4** (∗). *Let C be a cycle. Let $\mathbf{I}$ and $\mathbf{J}$ be two given independent sets of C. Assume that there are no $(C, \mathbf{I})$-rigid and $(C, \mathbf{J})$-rigid tokens. Then $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{J}$ if and only if $|\mathbf{I}| = |\mathbf{J}|$.*

4 The forbidden structures

In this section, we describe two non-trivial structures that forbid the existence of a TS-sequence between any two independent sets of a cactus G. The first structure is the $(G, \mathbf{I})$-rigid tokens, i.e., the tokens in $\mathbf{I}$ that cannot be slid along any edge of G.

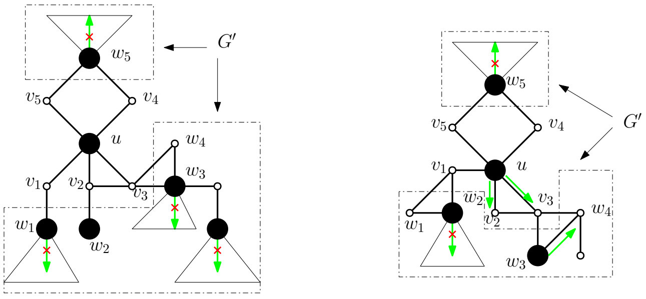
▶ **Lemma 5.** *Let $\mathbf{I}$ be an independent set of a cactus G. For any vertex $u \in \mathbf{I}$, the token t placed at u is $(G, \mathbf{I})$-rigid (see Figure 4(a)) if and only if for every vertex $v \in N_G(u)$, there exists a vertex $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$ satisfying one of the following conditions:*

(i) *The token t_w on w is $(G', \mathbf{I} \cap G')$-rigid, where $G' = G - N_G[u]$.*

(ii) *The token t_w on w is $(G', \mathbf{I} \cap G')$-movable; and there exists a cycle C in G such that $u \notin V(C)$, $\{v, w\} \subseteq V(C)$, and the path $P = C - v$ is $(G', \mathbf{I} \cap G')$-confined.*

Proof. First of all, we show the only-if-part. Let $v \in N_G(u)$. Assume that there exists $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$ such that either (i) or (ii) holds. We claim that in both cases, t cannot be slid to v.

- If (i) holds then clearly there is no TS-sequence in G' which slides t_w to a vertex in $N_{G'}(w) = N_G(w) \setminus \{v\}$. Hence, t cannot be slid to v.



(a) (b)

Figure 4 The token placed at $u \in \mathbf{I}$ is (a) $(G, \mathbf{I})$-rigid or (b) $(G, \mathbf{I})$-movable.

- When (ii) holds. Since t_w is $(G', \mathbf{I} \cap G')$-movable, it can be (at least) slid in G' to a vertex $x \in N_{G'}(w)$ by some TS-sequence $\mathcal{S}$. Since P is $(G', \mathbf{I} \cap G')$-confined, there is no TS-sequence in G' that slides a token from $G' - P$ to P and vice versa. Clearly, this also holds for $\mathcal{S}$. Let $w' \in N_G(v) \cap V(C)$ such that $w' \neq w$. Hence, if $w' \notin \mathbf{I}$ then before sliding any other token in P, $\mathcal{S}$ must move a token in $N_P(w') \cap \mathbf{I}$ (because $\mathbf{I} \cap P$ is a maximum independent set of P) to w'. Clearly, $N_G(v) \cap \mathbf{I}' \neq \emptyset$ for any $\mathbf{I}'$ such that $\mathbf{I} \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{I}'$, which means that t cannot be slid to v.

We have shown that if either (i) or (ii) holds, t cannot be slid to v. Since this holds for any $v \in N_G(u)$, it follows that t is $(G, \mathbf{I})$-rigid.

Next, we show the if-part. More precisely, we claim that if both (i) and (ii) do not hold, then t is $(G, \mathbf{I})$-movable (see Figure 4(b)).

Case 1: There exists $v \in N_G(u)$ such that $\big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I} = \emptyset$. Clearly, t can be slid to v and hence is $(G, \mathbf{I})$-movable.

Case 2: For all $v \in N_G(u)$, $\big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I} \neq \emptyset$. Let $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$. Since (i) does not hold, we can assume that t_w is $(G', \mathbf{I} \cap G')$-movable. Since (ii) does not hold, for any cycle C of G, (at least) one of the following conditions does not hold: (a) $u \notin V(C)$; (b) $\{v, w\} \subseteq V(C)$; (c) P is $(G, \mathbf{I})$-confined. Note that by definition, $w \neq u$. Additionally, since G is a cactus, there is at most one cycle C that contains both v and w. Let $H(G', w)$ be the (connected) component of G' containing w. We claim that for each such w above, one can slide t_w to a vertex in $N_{H(G',w)}(w)$ without sliding another token to a vertex in $N_G(v)$ beforehand. Eventually, there are no tokens in $N_G(v)$ other than t. Consider the following cases:

Case 2-1: Any cycle C contains either v or w but not both of them. Since t_w is $(G, \mathbf{I})$-movable, it is also $(H(G', w), \mathbf{I} \cap H(G', w))$-movable. Assume that there exists a vertex $x \in N_G(v) \cap H(G', w)$, $x \neq w$. Since $H(G', w)$ is connected, there exists a wx-path Q in $H(G', w)$. Note that Q, vw and vx form a cycle in G that contains both v and w, which contradicts our assumption. Hence, $N_G(v) \cap H(G', w) = \{w\}$. Therefore, one can simply slides t_w to a vertex in $N_{H(G',w)}(w)$ without sliding another token to a vertex in $N_G(v)$ beforehand.

Case 2-2: There is a (unique) cycle C that contains both v and w. When $u \in V(C)$ holds. As before, $N_G(v) \cap H(G', w) = \{w\}$. Otherwise, using the same argument as before, we have that the wx-path Q, vw and vx form a cycle C' in G that contains both v and w, where $x \in N_G(v) \cap H(G', w)$ and $x \neq w$. Because Q (a subgraph of G') does not contain u, it follows that $C' \neq C$, which is a contradiction. Since $N_G(v) \cap H(G', w) = \{w\}$, one can simply slides t_w to a vertex in $N_{H(G',w)}(w)$ without sliding another token to a vertex in $N_G(v)$ beforehand.

When $u \notin V(C)$ holds. Let $w' \in N_C(v)$, $w' \neq w$. By definition of a cactus and our assumption, $N_C(v) \cap H(G', w) = \{w, w'\}$. Since $\{v, w\} \subseteq V(C)$, it must happen that the condition (c) does not hold. By Lemma 1, there exists an independent set $\mathbf{I}'$ with $\mathbf{I} \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{I}'$ such that $|\mathbf{I} \cap P| < \lfloor k/2 \rfloor$, where $P = C - v$ and k is the length of C. (A maximum independent set of P must be of size $\lfloor k/2 \rfloor$.) Suppose that both w and w' are in $\mathbf{I}'$. Note that both t_w and $t_{w'}$ are $(G', \mathbf{I}')$-movable. Let $\mathcal{S}_w$ be a TS-sequence in G' that slides t_w to a vertex $x \in N_{H(G',w)}(w)$. Similarly, let $\mathcal{S}_{w'}$ be a TS-sequence in G' that slides $t_{w'}$ to a vertex $y \in N_{H(G',w)}(w')$. Since $|\mathbf{I}' \cap P| \leq \lfloor k/2 \rfloor - 1$, $\mathcal{S}_w$ (resp. $\mathcal{S}_{w'}$) does not involve any vertex in $\mathbf{I} \cap G^x_C$ where $x \in N_C[w']$ (resp. $x \in N_C[w]$). Note that by Proposition 3, $\mathcal{S}_w$ and $\mathcal{S}_{w'}$ can indeed be performed in G. Clearly, after applying both $\mathcal{S}_w$ and $\mathcal{S}_{w'}$, the number of tokens in $N_G(v)$ is reduced. Next, if either w or w' is in $\mathbf{I}'$, we can simply perform either $\mathcal{S}_w$ or $\mathcal{S}_{w'}$, respectively. If none of them is in $\mathbf{I}'$, nothing needs to be done.

We have shown that in any case, the number of tokens in $N_G(v)$ is reduced each time we slide the $(G', \mathbf{I} \cap G')$-movable token in $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$ to a vertex not in $N_G(v)$, and all such slidings can be performed independently (in each component of G'). Eventually, $N_G(v) \cap \mathbf{I} = \{u\}$, and hence we can slide t to v immediately, which implies that t is $(G, \mathbf{I})$-movable. ◀

We note that if an induced path P of a cactus G is of even length k, then by Lemma 1, it follows that P is $(G, \mathbf{I})$-confined if and only if $\mathbf{I} \cap P$ is a maximum independent set of P and any token placed at $x \in \mathbf{I} \cap P$ is $(G^x_P, \mathbf{I} \cap G^x_P)$-rigid. Since k is even and $\mathbf{I} \cap P$ is a maximum independent set of P, no token can be slid along any edge of P, i.e., the second condition is equivalent to saying that any token placed at $x \in \mathbf{I} \cap P$ is $(G, \mathbf{I})$-rigid. Now, we consider the case k is odd.

▶ **Lemma 6** (∗). *Let G be a cactus. Let $P = p_1 p_2 \dots p_l$ be an induced path in G. Let $\mathbf{I}$ be an independent set of G satisfying that $\mathbf{I} \cap P$ is a maximum independent set of P. Assume that for any $x \in \mathbf{I} \cap P$, the token placed at x is $(G, \mathbf{I})$-movable.*

Then, P is $(G, \mathbf{I})$-confined if and only if l is even (i.e., the length $k = l - 1$ of P is odd) and there exist two independent sets $\mathbf{I}'_1$ and $\mathbf{I}'_2$ such that

(i) *$\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{I}'$, where $\mathbf{I}' \in \{\mathbf{I}, \mathbf{I}'_1, \mathbf{I}'_2\}$,*
(ii) *$\mathbf{I}'_1 \cap P = \{p_1, p_3, \dots, p_{l-1}\}$, $\mathbf{I}'_2 \cap P = \{p_2, p_4, \dots, p_l\}$, and*
(iii) *for every $x \in \mathbf{I}' \cap P$, the token placed at x is $(G^x_P, \mathbf{I}' \cap G^x_P)$-rigid.*

The next lemma says that one can decide if the token t placed on u is $(G, \mathbf{I})$-rigid in linear time. Consequently, $\mathscr{R}(G, \mathbf{I})$ can be computed in polynomial time.

▶ **Lemma 7.** *Let $\mathbf{I}$ be an independent set of a cactus G. Let $u \in \mathbf{I}$. One can check if the token t placed on u is $(G, \mathbf{I})$-rigid in $O(n)$ time, where $n = |V(G)|$. Consequently, one can determine all $(G, \mathbf{I})$-rigid tokens in $O(n^2)$ time.*

Proof. We describe a recursive function CheckRigid(G, $\mathbf{I} \cap G$, u) for checking if t is $(G, \mathbf{I})$-rigid[1]. Clearly, if $N_G(u) = \emptyset$ then (by definition) t is $(G, \mathbf{I})$-rigid. We then consider the case when $N_G(u) \neq \emptyset$. We want to analyze the cases when t is not $(G, \mathbf{I})$-rigid using Lemma 5. If there exists $v \in N_G(u)$ such that $\big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I} = \emptyset$ then clearly t is not $(G, \mathbf{I})$-rigid. Otherwise, for each $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$, we need to check if the token t_w at w is $(G', \mathbf{I} \cap G')$-rigid, where $G' = G - N_G[u]$. It suffices to check if t_w is $(H(G', w), \mathbf{I} \cap H(G', w))$-rigid, where

[1] A pseudo-code of this algorithm is described in Algorithm 1 in the appendix.

$H(G', w)$ is the (connected) component of G' containing w. Note that by the definition of a cactus, it must happen that $1 \leq |N_G(v) \cap H(G', w)| \leq 2$.

Case 1: $N_G(v) \cap H(G', w) = \{w\}$. In this case, the cycle C mentioned in Lemma 5(ii) does not exist. Hence, if for all $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$, t_w is not $(H(G', w), \mathbf{I} \cap H(G', w))$-rigid, we can immediately conclude that t is not $(G, \mathbf{I})$-rigid, because we can slide all t_w to a vertex in $N_{H(G',w)}(w)$ and slide t to v.

Case 2: $N_G(v) \cap H(G', w) = \{w, w'\}$, $(w' \neq w)$. In this case, the cycle C mentioned in Lemma 5(ii) does exist. If for all $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$, t_w is not $(H(G', w), \mathbf{I} \cap H(G', w))$-rigid, we need to check if Lemma 5(ii) holds. If for all component $H(G', w)$ satisfying $N_G(v) \cap H(G', w) = \{w, w'\}$, Lemma 5(ii) does not hold, then we can conclude that t is not $(G, \mathbf{I})$-rigid, because we can slide all t_w to a vertex in $N_{H(G',w)}(w)$ (no token is slid to w' during this process) and slide t to v.

We now describe the function CheckConfinedPath for checking if Lemma 5(ii) holds. Let C be the (unique) cycle in G (of length k) containing v, w (and also w'). Let $P = C - v = p_1 p_2 \dots p_{k-1}$ with $p_1 = w, p_{k-1} = w'$. By the definition of G', it follows that $u \notin V(C) \subseteq V(G') \cup \{v\}$. Note that for each $x \in V(C) \setminus \{v\} = V(P)$, the graph G^x_C is a subgraph of $H(G', w)$. If $|\mathbf{I} \cap P| < \lfloor k/2 \rfloor$, Lemma 5(ii) clearly does not hold. If k is even then it also does not hold, since t_w is not $(H(G', w), \mathbf{I} \cap H(G', w))$-rigid. If $|\mathbf{I} \cap P| = \lfloor k/2 \rfloor$, we consider the set of tokens in $\mathbf{I} \cap P$. If there exists a vertex $x \in \mathbf{I} \cap P$ such that the token t_x placed at x is $(G^x_C, \mathbf{I} \cap G^x_C)$-movable, we can conclude that Lemma 5(ii) does not hold since by moving t_x to a vertex in G^x_C, we also obtain an independent set $\mathbf{I}'$ satisfying $\mathbf{I} \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{I}'$ and $|\mathbf{I}' \cap P| < \lfloor k/2 \rfloor$ (see Lemma 1). Thus, we can now consider the case when all t_x $(x \in \mathbf{I} \cap P)$ are $(G^x_C, \mathbf{I} \cap G^x_C)$-rigid. Note that from Lemma 5 and the assumption that t_w (and $t_{w'}$ if $w' \in \mathbf{I}$) is $(H(G', w), \mathbf{I} \cap H(G', w))$-movable, it follows that for each $x \in \mathbf{I} \cap P$, t_x must be $(H(G', w), \mathbf{I} \cap H(G', w))$-movable, and thus $(G', \mathbf{I} \cap G')$-movable (see Proposition 2). Thus, one can now apply Lemma 6. One can construct the independent sets $\mathbf{I}'_1, \mathbf{I}'_2$ described in Lemma 6 from $\mathbf{I} \cap G'$ by sliding tokens in G' (which can also be extended to a TS-sequence in G) as follows. Let i be the smallest index such that $p_i \in \mathbf{I}'_1 \setminus \mathbf{I}$. From the definition of $\mathbf{I}'_1 \cap P$, i must be even. Since $\mathbf{I} \cap P$ is a maximum independent set of P, it follows that $p_j \in \mathbf{I}'_1$ for j odd, $j < i - 1$, and $p_j \in \mathbf{I} \setminus \mathbf{I}'_1$ for j even, $j \geq i$. By Lemma 1, any token placed at $x \in \mathbf{I} \cap P$ must be $(G^x_P, \mathbf{I} \cap G^x_P)$-rigid. Since the token t_{p_i} on p_i is $(G', \mathbf{I} \cap G')$-movable but $(G^{p_i}_P, \mathbf{I} \cap G^{p_i}_P)$-rigid, it can only be slid to p_{i-1}. In other words, there exists a TS-sequence $\mathcal{S}_{p_i}$ in G' which slides t_{p_i} to p_{i-1}. Note that $\mathcal{S}_{p_i}$ can be constructed recursively as follows. From Lemma 5, if $\big(N_{G'}(p_{i-1}) \setminus \{p_i\}\big) \cap \mathbf{I} = \emptyset$, $\mathcal{S}_{p_i}$ contains only a single step of sliding t_{p_i} to p_{i-1}. On the other hand, if $\big(N_{G'}(p_{i-1}) \setminus \{p_i\}\big) \cap \mathbf{I} \neq \emptyset$, there must be a TS-sequence $\mathcal{S}'_{p_i}$ in $G'' = G' - N_{G'}[p_i]$ which slides any token in $\big(N_{G'}(p_{i-1}) \setminus \{p_i\}\big) \cap \mathbf{I}$ to some vertex not in $N_{G'}(p_{i-1}) \setminus \{p_i\}$ without having to move a new token to $N_{G'}(p_{i-1}) \setminus \{p_i\}$ beforehand. From Proposition 3, $\mathcal{S}'_{p_i}$ can be extended to a TS-sequence in G'. Hence, $\mathcal{S}_{p_i}$ is constructed by simply performing $\mathcal{S}'_{p_i}$ first, then performing a single sliding step which moves t_{p_i} to p_{i-1}. Repeat the described steps, we finally obtain an independent set $\mathbf{I}'_1$ which satisfies $\mathbf{I} \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{I}'_1$ and $\mathbf{I}'_1 \cap P = \{p_1, p_3, \dots\}$. Note that the recursive construction of $\mathcal{S}_{p_i}$ can indeed be derived from the recursive process of checking rigidity which we are describing. A similar procedure can be applied for constructing $\mathbf{I}'_2$. Once we constructed $\mathbf{I}'_1$ and $\mathbf{I}'_2$, we need to check for all $y \in P \cap (\mathbf{I}'_i \setminus \mathbf{I})$ $(i = 1, 2)$ whether the token t_y placed at y is $(G^y_C, \mathbf{I}'_i \cap G^y_C)$-rigid. If all of such t_y are $(G^y_C, \mathbf{I}'_i \cap G^y_C)$-rigid, by Lemma 6, we conclude that Lemma 5(ii) holds.

Next, we analyze the complexity of our algorithm. Note that the time complexity of this recursive algorithm is proportional to the number of callings of the CheckRigid function. Observe that for any vertex $u \in V(G)$, the function CheckRigid is called for u at most three times: at most one time during the process of checking Lemma 5(i) (the results of this checking can be used for constructing the sets $\mathbf{I}'_1$ and $\mathbf{I}'_2$ described in Lemma 6), and at most two times during the process of checking if Lemma 5(ii) holds. Hence, it takes at most $O(n)$ time to check if a token is $(G, \mathbf{I})$-rigid. Therefore, $\mathscr{R}(G, \mathbf{I})$ can be computed in $O(n^2)$ time. ◀

In the remaining part of this section, we consider the second forbidden structure – the $(G, \mathbf{I})$-confined cycles. Analogously to the case of confined paths, one can also derive (using Lemma 1) that if a cycle C is of even length k, then it is $(G, \mathbf{I})$-confined if and only if $\mathbf{I} \cap C$ is a maximum independent set of C and any token placed at $x \in \mathbf{I} \cap C$ is $(G, \mathbf{I})$-rigid. Similar to Lemma 6, we have

▶ **Lemma 8** (∗)**.** *Let G be a cactus. Let $C = c_1c_2 \dots c_kc_1$ be a cycle in G. Let $\mathbf{I}$ be an independent set of G satisfying that $\mathbf{I} \cap C$ is a maximum independent set of C. Assume that for any $x \in \mathbf{I} \cap C$, the token placed at x is $(G, \mathbf{I})$-movable.*

Then, C is $(G, \mathbf{I})$-confined if and only if k is odd and there exist three independent sets $\mathbf{I}'_1$, $\mathbf{I}'_2$ and $\mathbf{I}'_3$ such that

(i) *$\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{I}'$, where $\mathbf{I}' \in \{\mathbf{I}, \mathbf{I}'_1, \mathbf{I}'_2, \mathbf{I}'_3\}$,*
(ii) *$\mathbf{I}'_1 \cap C = \{c_1, c_3, \dots, c_{k-2}\}$, $\mathbf{I}'_2 \cap C = \{c_2, c_4, \dots, c_{k-1}\}$, $\mathbf{I}'_3 \cap C = \{c_3, c_5, \dots, c_k\}$, and*
(iii) *for every $x \in \mathbf{I}' \cap C$, the token placed at x is $(G^x_C, \mathbf{I}' \cap G^x_C)$-rigid.*

Using Lemma 8, we have

▶ **Lemma 9** (∗)**.** *Let G be a cactus. Let $\mathbf{I}$ be an independent set of G. Assume that $\mathscr{R}(G, \mathbf{I}) = \emptyset$. Then for any cycle C in G, one can decide if C is $(G, \mathbf{I})$-confined in $O(n)$ time, where $n = |V(G)|$. Consequently, computing $\mathscr{C}(G, \mathbf{I})$ takes at most $O(n^2)$ time.*

Proof sketch. By modifying the function CheckConfinedPath in the proof of Lemma 7, one can obtain an algorithm for checking if a length-k-cycle $C = c_1c_2 \dots c_kc_1$ in G is $(G, \mathbf{I})$-confined. Keep in mind that C must satisfy the conditions given in Lemma 8. Moreover, since $\mathscr{R}(G, \mathbf{I}) = \emptyset$, it suffices to consider only cycles of odd length. The condition $\mathscr{R}(G, \mathbf{I}) = \emptyset$ also implies that for any $x \in \mathbf{I} \cap C$, the token placed at x is $(G, \mathbf{I})$-movable. ◀

5 Sliding tokens on a cactus

In this section, we describe a polynomial-time algorithm for solving Sliding Token for cacti and prove its correctness. More precisely, we claim that:

▶ **Theorem 10.** *Let $(G, \mathbf{I}, \mathbf{J})$ be an instance of* Sliding Token *where G is a cactus and $\mathbf{I}$, $\mathbf{J}$ are two independent sets of G. Then, it takes at most $O(n^2)$ time to decide if $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, where $n = |V(G)|$.*

Let $(G, \mathbf{I}, \mathbf{J})$ be an instance of Sliding Token where G is a cactus and $\mathbf{I}$, $\mathbf{J}$ are two independent sets of G. The following algorithm decides if $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$.

Step 1:
Step 1-1: If $\mathscr{R}(G, \mathbf{I}) \neq \mathscr{R}(G, \mathbf{J})$, return no.

Step 1-2: Otherwise, remove all vertices in $N_G[\mathscr{R}(G,\mathbf{I})]$ and go to **Step 2**. Let G' be the resulting graph.

Step 2:

Step 2-1: If $\mathscr{C}(G',\mathbf{I}\cap G') \neq \mathscr{C}(G',\mathbf{J}\cap G')$, return NO

Step 2-2: Otherwise, remove all cycles in $\mathscr{C}(G',\mathbf{I}\cap G')$ and go to **Step 3**. Let G'' be the resulting graph.

Step 3: If $|\mathbf{I}\cap F| \neq |\mathbf{J}\cap F|$ for some component F of G'' then return NO. Otherwise, return YES.

We now estimate the running time of this algorithm. First of all, Lemma 7 ensures that **Step 1-1** can be performed in $O(n^2)$ time. **Step 1-2** clearly can be performed in $O(n)$ time. Thus, **Step 1** takes at most $O(n^2)$ time. **Step 2** also takes at most $O(n^2)$ time since by Lemma 9, **Step 2-1** takes $O(n^2)$ time, and **Step 2-2** can be performed in $O(n)$ time. Finally, **Step 3** clearly runs in $O(n)$ time. In total, the algorithm runs in $O(n^2)$ time.

It remains to show the correctness of our algorithm. First of all, we prove an useful observation.

▶ **Lemma 11** (∗). *Let $\mathbf{I}$ be an independent set of a cactus G. Let $v \notin \mathbf{I}$. Assume that $\mathscr{R}(G,\mathbf{I}) = \emptyset$, and $N_G(v)\cap\mathbf{I} \neq \emptyset$. Then, there is at most one $(G',\mathbf{I}\cap G')$-rigid token in $N_G(v)\cap\mathbf{I}$, where $G' = G - v$. On the other hand, if there exists a cycle C containing v such that the path $P = C - v$ is $(G',\mathbf{I}\cap G')$-confined, then all tokens in $N_G(v)\cap\mathbf{I}$ must be $(G',\mathbf{I}\cap G')$-movable. Moreover, if $\mathscr{C}(G,\mathbf{I}) = \emptyset$ then there is at most one cycle C with the above described property.*

The next lemma claims that **Step 1-1** and **Step 2-1** are correct.

▶ **Lemma 12** (∗). *Let $\mathbf{I}$ and $\mathbf{J}$ be independent sets of a cactus G. If $\mathscr{R}(G,\mathbf{I}) \neq \mathscr{R}(R,\mathbf{J})$, then there is no* TS*-sequence in G which reconfigures $\mathbf{I}$ to $\mathbf{J}$.*

Assume that $\mathscr{R}(G,\mathbf{I}) = \mathscr{R}(G,\mathbf{J}) = \emptyset$. If $\mathscr{C}(G,\mathbf{I}) \neq \mathscr{C}(G,\mathbf{J})$ then there is no TS*-sequence in G which reconfigures $\mathbf{I}$ to $\mathbf{J}$.*

The next lemma ensures the correctness of **Step 1-2** and **Step 2-2**.

▶ **Lemma 13** (∗). *Suppose that $\mathscr{R}(G,\mathbf{I}) = \mathscr{R}(G,\mathbf{J})$ for two given independent sets $\mathbf{I}$ and $\mathbf{J}$ of a cactus G, and let G' be the graph obtained from G by deleting the vertices in $N_G[\mathscr{R}(G,\mathbf{I})] = N_G[\mathscr{R}(G,\mathbf{J})]$. Then $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ if and only if $\mathbf{I}\cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{J}\cap G'$. Furthermore, $\mathscr{R}(G',\mathbf{I}\cap G') = \mathscr{R}(G',\mathbf{J}\cap G') = \emptyset$.*

Suppose that $\mathscr{C}(G',\mathbf{I}\cap G') = \mathscr{C}(G',\mathbf{I}\cap G') \neq \emptyset$. Let G'' be the graph obtained by removing all cycles in $\mathscr{C}(G',\mathbf{I}\cap G')$. Then $\mathbf{I}\cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{J}\cap G'$ if and only if $\mathbf{I}\cap G'' \overset{G''}{\leftrightsquigarrow} \mathbf{J}\cap G''$. Furthermore, $\mathscr{R}(G'',\mathbf{I}\cap G'') = \mathscr{R}(G'',\mathbf{J}\cap G'') = \emptyset$ and $\mathscr{C}(G'',\mathbf{I}\cap G'') = \mathscr{C}(G'',\mathbf{J}\cap G'') = \emptyset$.

Before proving the correctness of **Step 3**, we need some extra definitions. Let B_1, B_2 be two blocks of a cactus G. We say that B_1 is a *neighbor* of B_2 if $V(B_1)\cap V(B_2) \neq \emptyset$. A block B is *safe* if it has at most one cut vertex and at most one neighbor containing more than one cut vertex. For example, the blocks marked with black color in Figure 5 are safe. A vertex $v \in V(G)$ is *safe* if it is a non-cut vertex of some safe block B of G.

For each cut vertex w of G, let $\mathcal{B}_w$ be the smallest subgraph of G such that $\mathcal{B}_w$ contains all safe blocks of G containing w (see Figure 5). $\mathcal{B}_w$ can also be viewed as a collection of safe blocks sharing the same cut vertex w. Observe that for two distinct cut vertices w_1, w_2, $V(\mathcal{B}_{w_1})\cap V(\mathcal{B}_{w_2}) = \emptyset$. If no safe block contains w, we define $\mathcal{B}_w = \emptyset$.

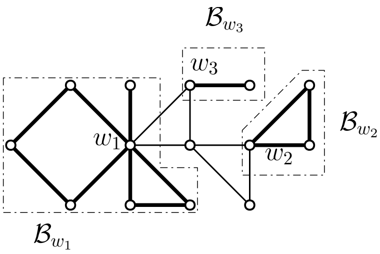


Figure 5 Examples of safe blocks.

Let w be a cut vertex of a cactus G such that $\mathcal{B}_w \neq \emptyset$. For each block $B \in \mathcal{B}_w$, since each block of G is either K_2 or a simple cycle and all blocks in $\mathcal{B}_w$ share the same (unique) cut vertex w, without loss of generality, assume that the vertices of B are labeled as $v_0[B], v_1[B], \dots, v_{|B|-1}[B]$ such that $v_0[B] = w$; $v_i[B]$ is adjacent to $v_{i+1}[B]$, $i \in \{1, 2, \dots, |B| - 2\}$; and $v_0[B]$ is adjacent to $v_{|B|-1}[B]$.

▶ **Lemma 14** ($*$). *Let* $\mathbf{I}$ *be an independent set of a given cactus* G. *Assume that* $\mathscr{R}(G, \mathbf{I}) = \emptyset$ *and* $\mathscr{C}(G, \mathbf{I}) = \emptyset$. *Let* w *be a cut vertex of* G *such that* $\mathcal{B}_w \neq \emptyset$. *Assume that* $|\mathbf{I}| \geq \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$.

(i) *If* $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) = 0$, *then there exists an independent set* $\mathbf{I}'$ *satisfying that* $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{I}'$ *and* $v \in \mathbf{I}'$, *where* $v \in V(\mathcal{B}_w)$ *is some safe vertex of* G *and* $|B|$ *denotes the number of vertices of* $B \in \mathcal{B}_w$.

(ii) *If* $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) \geq 1$, *then there exists an independent set* $\mathbf{I}'$ *satisfying that* $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{I}'$, $N_{\mathcal{B}_w}(w) \cap \mathbf{I}' = \emptyset$, *and* $|\mathbf{I}' \cap (\mathcal{B}_w - w)| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$.

▶ **Lemma 15** ($*$). *Let* $\mathbf{I}$ *be an independent set of a given cactus* G. *Assume that* $\mathscr{R}(G, \mathbf{I}) = \emptyset$, *and* $\mathscr{C}(G, \mathbf{I}) = \emptyset$. *Let* w *be a cut vertex of* G *such that* $\mathcal{B}_w \neq \emptyset$.

(i) *If* $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) = 0$. *Let* $v \in V(\mathcal{B}_w)$ *be a safe vertex of* G. *Assume that* $v \in \mathbf{I}$. *Then,* $\mathscr{R}(G^*, \mathbf{I}^*) = \emptyset$, *where* G^* *is the graph obtained from* G *by removing all vertices in* $\mathcal{B}_w$ *and* $\mathbf{I}^* = \mathbf{I} \cap G^*$. *Moreover,* $\mathscr{C}(G^*, \mathbf{I}^*) = \emptyset$.

(ii) *If* $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) \geq 1$. *Assume that* $\mathbf{I} \cap (\mathcal{B}_w - w) = \mathbf{I} \cap \bigcup_{B \in \mathcal{B}_w} \{v_2[B], v_4[B], \dots\}$, $|\mathbf{I} \cap (\mathcal{B}_w - w)| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$ *and* $N_{\mathcal{B}_w}(w) \cap \mathbf{I} = \emptyset$. *Let* G^* *be the graph obtained from* G *by removing all vertices in* $N_G[\mathbf{I} \cap (\mathcal{B}_w - w)]$ *and* $\mathbf{I}^* = \mathbf{I} \cap G^*$. *Then* $\mathscr{R}(G^*, \mathbf{I}^*) = \emptyset$ *and* $\mathscr{C}(G^*, \mathbf{I}^*) = \emptyset$.

The next lemma ensures the correctness of **Step 3**.

▶ **Lemma 16** ($*$). *Let* G *be a cactus. Let* $\mathbf{I}$ *and* $\mathbf{J}$ *be two given independent sets of* G. *Assume that* $\mathscr{R}(G, \mathbf{I}) = \mathscr{R}(G, \mathbf{J}) = \emptyset$ *and* $\mathscr{C}(G, \mathbf{I}) = \mathscr{C}(G, \mathbf{I}) = \emptyset$. *Then* $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ *if and only if* $|\mathbf{I}| = |\mathbf{J}|$.

Acknowledgements. We would like to thank Yota Otachi, Takehiro Ito, Tatsuhiko Hatanaka, Haruka Mizuta and Yuhei Moriya for their useful comments and discussions.

References

1 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. *arXiv preprint*, 2016. `arXiv:1605.00442`.

2 Paul Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In R. Ravi and IngeLi Gørtz, editors, *Algorithm Theory – SWAT 2014*, volume 8503 of *LNCS*, pages 86–97. Springer, 2014.

3 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.

4 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015.

5 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.

6 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In Khaled Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation – ISAAC 2015*, volume 9472 of *LNCS*, pages 237–247. Springer, 2015.

7 Martin Gardner. The hypnotic fascination of sliding-block puzzles. *Scientific American*, 210(2):122, 1964.

8 Parikshit Gopalan, Phokion G. Kolaitis, Elitza Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.

9 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1):72–96, 2005.

10 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12):1054–1065, 2011.

11 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

12 Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. An exact algorithm for the Boolean connectivity problem for k-CNF. *Theoretical Computer Science*, 412(35):4613–4618, 2011.

13 Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of Boolean formulas. In M. Magnús Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – ICALP 2015*, volume 9134 of *LNCS*. Springer, 2015.

14 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation – IPEC 2013*, volume 8246 of *LNCS*, pages 281–294. Springer, 2013.

15 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation – IPEC 2014*, volume 8894 of *LNCS*, pages 246–257. Springer, 2014.

16 Moritz Mühlenthaler. Degree-constrained subgraph reconfiguration is in P. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Mathematical Foundations of Computer Science – MFCS 2015*, volume 9235 of *LNCS*, pages 505–516. Springer, 2015.

17 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.

18 Takeshi Yamada and Ryuhei Uehara. Shortest reconfiguration of sliding tokens on a caterpillar. In Mohammad Kaykobad and Rossella Petreschi, editors, *Algorithms and Computation – WALCOM 2016*, volume 9627 of *LNCS*, pages 236–248. Springer, 2016.

A Details of Section 3

Proof of Lemma 1. We claim that (i) ⇔ (ii) and (ii) ⇔ (iii).

- (i) ⇔ (ii).
  - (i) ⇒ (ii). Assume that (i) holds, i.e., $\mathbf{I} \cap H$ is maximum and all tokens placed at vertices in $\mathbf{I} \cap H$ are $(G, \mathbf{I}, V(H))$-confined. Clearly, this implies (ii).
  - (ii) ⇒ (i). Assume that (ii) holds. i.e., for every independent set $\mathbf{J}$ satisfying $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, $\mathbf{J} \cap H$ is a maximum independent set of H. It follows that no token can be slid from a vertex in H to a vertex in $G - H$. Moreover, since $\mathbf{J} \cap H$ is always maximum, no token can be slid from a vertex in $G - H$ to H. Thus, any token placed at a vertex in $\mathbf{I} \cap H$ can only be slid along edges of H, i.e., it is $(G, \mathbf{I}, V(H))$-confined.
- (ii) ⇔ (iii).
  - (ii) ⇒ (iii). Assume that (ii) holds. First of all, it is clear that $\mathbf{I} \cap H$ is maximum. Assume that there exists an independent set $\mathbf{J}$, $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, and a vertex $x \in \mathbf{J} \cap H$ such that the token t_x placed at x is $(G^x_H, \mathbf{J} \cap G^x_H)$-movable, i.e., (iii) does not hold. Let $\mathcal{S} = \langle \mathbf{I}_1 = \mathbf{I}, \mathbf{I}_2, \dots, \mathbf{I}_\ell = \mathbf{J} \rangle$ be a TS-sequence in G which reconfigures $\mathbf{I}$ to $\mathbf{J}$. Let $\mathcal{S}' = \langle \mathbf{I}'_1 = \mathbf{J} \cap G^x_H, \mathbf{I}'_2, \dots, \mathbf{I}'_k \rangle$ be a TS-sequence in G^x_H which slides x to a vertex $y \in N_{G^x_H}(x)$. By definition of G^x_H, $y \notin V(H)$. Without loss of generality, assume that $x \in \mathbf{I}'_j \setminus \mathbf{I}'_k$ and $y \in \mathbf{I}'_k \setminus \mathbf{I}'_j$, where $j = 1, 2, \dots, k-1$. For any independent set $\mathbf{I}$ of G, $\mathbf{I} \cap G^x_H$ is also an independent set of G^x_H. Therefore, one can construct the TS-sequence $\langle \mathbf{I}_1 \cap G^x_H, \mathbf{I}_2 \cap G^x_H, \dots, \mathbf{I}_\ell \cap G^x_H \rangle$ from $\mathcal{S}$. Thus, we have $\mathbf{I} \cap G^x_H \overset{G^x_H}{\leftrightsquigarrow} \mathbf{J} \cap G^x_H \overset{G^x_H}{\leftrightsquigarrow} \mathbf{I}'_{k-1}$. Note that for any independent set $\mathbf{I}'$ of G^x_H, since $V(G^x_H) \cap (\mathbf{I} - G^x_H) = \emptyset$ the set $\mathbf{I}' \cup (\mathbf{I} - G^x_H)$ is also independent. Therefore, $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J} \overset{G}{\leftrightsquigarrow} \mathbf{I}'_{k-1} \cup (\mathbf{I} - G^x_H)$. Let $\mathbf{J}' = \mathbf{I}'_{k-1} \cup (\mathbf{I} - G^x_H)$ then by our assumption $\mathbf{J}' \cap H$ is a maximum independent set of H. Let $\mathbf{J}'' = \mathbf{I}'_k \cup (\mathbf{I} - G^x_H)$. Similarly, we also have $\mathbf{J}'' \cap H$ must be a maximum independent set of H. Since $\mathbf{J}'' \setminus \mathbf{J}' = \{y\}$, $\mathbf{J}' \setminus \mathbf{J}'' = \{x\}$, and $y \notin V(H)$, this is a contradiction.
  - (iii) ⇒ (ii). Assume that (iii) holds. Assume that there exists an independent set $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ but $\mathbf{J} \cap H$ is not a maximum independent set of H, i.e., (ii) does not hold. Let $\mathcal{S} = \langle \mathbf{I}_1 = \mathbf{I}, \mathbf{I}_2, \dots, \mathbf{I}_\ell = \mathbf{J} \rangle$ be a TS-sequence which reconfigures $\mathbf{I}$ to $\mathbf{J}$. Without loss of generality, assume that $\mathbf{I}_i \cap H$ is a maximum independent set of H for $i = 1, 2, \dots, \ell - 1$. Let $x \in \mathbf{I}_{\ell-1} \setminus \mathbf{I}_\ell$ and $y \in \mathbf{I}_\ell \setminus \mathbf{I}_{\ell-1}$. Since $\mathbf{I}_\ell \cap H$ is not a maximum independent set of H, $|\mathbf{I}_\ell \cap H| < |\mathbf{I}_i \cap H|$ for $i = 1, 2, \dots, \ell - 1$. Hence, $y \notin V(H)$. Since $N_G(x) = N_{G^x_H}(x) \cup N_H(x)$ and $N_{G^x_H}(x) \cap N_H(x) = \emptyset$, y must be in G^x_H, which implies that $\mathcal{S}$ slides a token t_x on x to a vertex $y \in V(G^x_H)$. As in the previous part, one can indeed derive a TS-sequence in G^x_H from $\mathcal{S}$ which slides t_x to y, i.e., it is $(G^x_H, \mathbf{I}_{\ell-1} \cap G^x_H)$-movable. This is a contradiction. ◀

Proof of Proposition 2. Assume that $\mathcal{S} = \langle \mathbf{I}_1, \dots, \mathbf{I}_\ell \rangle$ is a TS-sequence in G that reconfigures $\mathbf{I} = \mathbf{I}_1$ to $\mathbf{J} = \mathbf{I}_\ell$. For any $i \in \{1, 2, \dots, k\}$ and any independent set $\mathbf{I}$ of G, as $\mathbf{I} \cap G_i \subseteq \mathbf{I}$, $\mathbf{I} \cap G_i$ is also independent. Hence, $\mathcal{S}_i = \langle \mathbf{I}_1 \cap G_i, \dots, \mathbf{I}_\ell \cap G_i \rangle$ reconfigures $\mathbf{I} \cap G_i$ to $\mathbf{J} \cap G_i$.

Assume that for each $i \in \{1, 2, \dots, k\}$, there exists a TS-sequence $\mathcal{S}'_i$ in G_i that reconfigures $\mathbf{I} \cap G_i$ to $\mathbf{J} \cap G_i$. For any two TS-sequences $\mathcal{S}'_i$ and $\mathcal{S}'_j$ $(i, j \in \{1, 2, \dots, k\})$, if the length of $\mathcal{S}'_i$ is smaller than the length of $\mathcal{S}'_j$ then we can make them equal by appending $\langle \mathbf{I} \cap G_i, \mathbf{I} \cap G_i, \dots \rangle$ to the end of $\mathcal{S}'_i$. Thus, assume that all $\mathcal{S}'_i$ are of equal length, i.e., any $\mathcal{S}'_i$ can be written in the form $\langle \mathbf{I}^i_1 = \mathbf{I} \cap G_i, \dots, \mathbf{I}^i_l = \mathbf{J} \cap G_i \rangle$. Let $\mathbf{I}^i$ be an independent set of G_i. Since $G_1, G_2, \dots, G_k$ are components of G, $\bigcup_{i=1}^k \mathbf{I}^i$ forms an independent set of G.

Thus, we can extend any sequence $\mathcal{S'}_i$ $(i = 1, 2, \dots, k)$ to a TS-sequence $\mathcal{S}_i$ in G as follows.

$$\mathcal{S}_i = \langle \mathbf{I}_1^i \cup \bigcup_{j=1}^{i-1} \mathbf{I}_l^j \cup \bigcup_{j=i+1}^{k} \mathbf{I}_1^j, \dots, \mathbf{I}_l^i \cup \bigcup_{j=1}^{i-1} \mathbf{I}_l^j \cup \bigcup_{j=i+1}^{k} \mathbf{I}_1^j \rangle.$$

Clearly, the sequence $\mathcal{S}$ constructed by first applying $\mathcal{S}_1$, then $\mathcal{S}_2$, and so on is the one that reconfigures $\mathbf{I}$ to $\mathbf{J}$ in G. ◀

Proof of Proposition 3. Since $u \in \mathbf{I}$ for any $\mathbf{I} \in \mathcal{S}$, the sequence $\mathcal{S}' = \langle \mathbf{I}_1 \setminus \{u\}, \dots, \mathbf{I}_\ell \setminus \{u\} \rangle$ clearly reconfigures $\mathbf{I}_1 \cap G' = \mathbf{I}_1 \setminus \{u\}$ to $\mathbf{I}_\ell \cap G' = \mathbf{I}_\ell \setminus \{u\}$. For any independent set $\mathbf{I}'$ of G', $\mathbf{I}' \cup \{u\}$ clearly forms an independent set of G. Hence, $\mathcal{S} = \langle \mathbf{I}'_1 \cup \{u\}, \dots, \mathbf{I}'_l \cup \{u\} \rangle$ reconfigures $\mathbf{I}'_1 \cup \{u\}$ to $\mathbf{I}'_l \cup \{u\}$. ◀

Proof of Lemma 4. If $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{J}$ then clearly $|\mathbf{I}| = |\mathbf{J}|$. Now, assume that $|\mathbf{I}| = |\mathbf{J}|$. We claim that $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{J}$. Let $C = v_1 v_2 \dots v_k v_1$. Let $\mathbf{I}'$ be an independent set of C such that $|\mathbf{I}'| = |\mathbf{I}| = |\mathbf{J}| \leq \lfloor k/2 \rfloor$ and $v_i \in \mathbf{I}'$ if i is odd. We claim that $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{I}'$. Similarly, one can also show that $\mathbf{J} \overset{C}{\leftrightsquigarrow} \mathbf{I}'$. Consider the following cases:

Case 1: $|\mathbf{I}| = \lfloor k/2 \rfloor$. Since there are no $(C, \mathbf{I})$-rigid tokens and $|\mathbf{I}| = \lfloor k/2 \rfloor$, k must be odd. Let i be the smallest index such that $v_i \in \mathbf{I} \setminus \mathbf{I}'$, $2 \leq i \leq k$. Hence, from the definition of $\mathbf{I}'$, i must be even. Moreover, $v_j \in \mathbf{I}'$ for odd j, $1 \leq j < i - 1$, and $v_j \in \mathbf{I}$ for even j, $i \leq j \leq k - 1$. Hence, one can slide the token on v_i to $v_{i-1} \in \mathbf{I}' \setminus \mathbf{I}$, then slide the token on v_{i+2} to v_{i+1}, and so on. Let $\mathcal{S}$ be the TS-sequence describing the above process, then clearly $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{I}'$, since each sliding step reduces $|\mathbf{I}' \setminus \mathbf{I}|$.

Case 2: $|\mathbf{I}| < \lfloor k/2 \rfloor$. Let i be the smallest index such that $v_i \in \mathbf{I} \setminus \mathbf{I}'$, $2 \leq i \leq k$. If $i = 2$ then since there are no $(C, \mathbf{I})$-rigid tokens, we can assume without loss of generality that $v_k \notin \mathbf{I}$; otherwise there exists a TS-sequence that slides the token in v_k to v_{k-1} and then one can deal with the resulting independent set. Let j be the smallest index such that $v_j \in \mathbf{I}' \setminus \mathbf{I}$, $1 \leq j \leq k$. Since $v_i \notin \mathbf{I}'$, $i > j$. Now, one can slide v_i to v_j and repeat the process. Let $\mathcal{S}$ be the TS-sequence describing the above process, then clearly $\mathbf{I} \overset{C}{\leftrightsquigarrow} \mathbf{I}'$. ◀

B Details of Section 4

Proof of Lemma 6.

($\Leftarrow$). Assume that l is even and the described independent sets $\mathbf{I}'_1, \mathbf{I}'_2$ exist. Since $\mathbf{I} \cap P$ is a maximum independent set of P, it suffices to show that all tokens in $\mathbf{I} \cap P$ are $(G, \mathbf{I}, V(P))$-confined. By Lemma 1, it is equivalent to saying that for every $\mathbf{J}$ satisfying $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, any token placed at $x \in P \cap \mathbf{J}$ is $(G^x_P, \mathbf{J} \cap G^x_P)$-rigid. Let $x \in \mathbf{J} \cap \mathbf{I}'_1 \cap P$ for some $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ and suppose that the token t_x placed at x is $(G^x_P, \mathbf{I}'_1 \cap G^x_P)$-rigid. We claim that it is also $(G^x_P, \mathbf{J} \cap G^x_P)$-rigid. Assume for the contradiction that there exists an independent set $\mathbf{J}'$ of G^x_P such that $\mathbf{J} \cap G^x_P \overset{G^x_P}{\leftrightsquigarrow} \mathbf{J}'$ but $x \notin \mathbf{J}'$. For any independent set $\mathbf{I}$ of G, note that $\mathbf{I} \cap G^x_P$ is also independent. Hence, it follows that $\mathbf{I}'_1 \cap G^x_P \overset{G^x_P}{\leftrightsquigarrow} \mathbf{J} \cap G^x_P \overset{G^X_P}{\leftrightsquigarrow} \mathbf{J}'$, which then implies that t_x is not $(G^x_P, \mathbf{I}'_1 \cap G^x_P)$-rigid. This is a contradiction. Hence, for every independent set $\mathbf{J}$ with $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, any token in $\mathbf{J} \cap \mathbf{I}'_1 \cap P$ is $(G^x_P, \mathbf{J} \cap G^x_P)$-rigid. Similarly, for every independent set $\mathbf{J}$ with $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, any token in $\mathbf{J} \cap \mathbf{I}'_2 \cap P$ is also $(G^x_P, \mathbf{J} \cap G^x_P)$-rigid. Moreover, for every $\mathbf{J}$ with $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, $\mathbf{J} \cap P = \big(\mathbf{J} \cap \mathbf{I}'_1 \cap P\big) \cup \big(\mathbf{J} \cap \mathbf{I}'_2 \cap P\big)$. Hence, every token placed at $x \in \mathbf{J} \cap P$ is $(G^x_P, \mathbf{J} \cap G^x_P)$-rigid.

(⇒). Assume that P is $(G, \mathbf{I})$-confined. Since $\mathbf{I} \cap P$ is a maximum independent set of P and any token placed at $x \in \mathbf{I} \cap P$ is $(G, \mathbf{I})$-movable, it follows that l must be even. We show how to construct $\mathbf{I}'_1$ from $\mathbf{I}$ using TS rule. A similar process can be applied for $\mathbf{I}'_2$. Let i be the smallest index such that $p_i \in \mathbf{I}'_1 \setminus \mathbf{I}$. From the definition of $\mathbf{I}'_1 \cap P$, i must be even. Since $\mathbf{I} \cap P$ is a maximum independent set of P, it follows that $p_j \in \mathbf{I}'_1$ for j odd, $j < i - 1$, and $p_j \in \mathbf{I} \setminus \mathbf{I}'_1$ for j even, $j \geq i$. By Lemma 1, any token placed at $x \in \mathbf{I} \cap P$ must be $(G^x_P, \mathbf{I} \cap G^x_P)$-rigid. Since the token t_{p_i} on p_i is $(G, \mathbf{I})$-movable but $(G^{p_i}_P, \mathbf{I} \cap G^{p_i}_P)$-rigid, it can only be slid to p_{i-1}. In other words, there exists a TS-sequence $\mathcal{S}_{p_i}$ in G which slides t_{p_i} to p_{i-1} Note that $\mathcal{S}_{p_i}$ can be constructed recursively as follows. From Lemma 5, if $\big(N_G(p_{i-1}) \setminus \{p_i\}\big) \cap \mathbf{I} = \emptyset$, $\mathcal{S}_{p_i}$ contains only a single step of sliding t_{p_i} to p_{i-1}. On the other hand, if $\big(N_G(p_{i-1}) \setminus \{p_i\}\big) \cap \mathbf{I} \neq \emptyset$, there must be a TS-sequence $\mathcal{S'}_{p_i}$ in $G' = G - N_G[p_i]$ which slides any token in $\big(N_G(p_{i-1}) \setminus \{p_i\}\big) \cap \mathbf{I}$ to some vertex not in $N_G(p_{i-1}) \setminus \{p_i\}$ without having to move a new token to $N_G(p_{i-1}) \setminus \{p_i\}$ beforehand. From Proposition 3, $\mathcal{S'}_{p_i}$ can be extended to a TS-sequence in G. Hence, $\mathcal{S}_{p_i}$ is constructed by simply performing $\mathcal{S'}_{p_i}$ first, then performing a single sliding step which moves t_{p_i} to p_{i-1}. Repeat the described steps, we finally obtain an independent set $\mathbf{I}'_1$ which satisfies $\mathbf{I} \cap G' \overset{G'}{\rightsquigarrow} \mathbf{I}'_1$ and $\mathbf{I}'_1 \cap P = \{p_1, p_3, \dots\}$. ◀

Proof of Lemma 8.

(⇐). Assume that k is odd and the described independent sets $\mathbf{I}'_1, \mathbf{I}'_2, \mathbf{I}'_3$ exist. As in Lemma 6, it suffices to show that for every $\mathbf{J}$ with $\mathbf{I} \overset{G}{\rightsquigarrow} \mathbf{J}$, every token placed at $x \in \mathbf{J} \cap C$ is $(G^x_C, \mathbf{J} \cap G^x_C)$-rigid. For $i \in \{1, 2, 3\}$, let $x \in \mathbf{J} \cap \mathbf{I}'_i \cap C$ for some $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\rightsquigarrow} \mathbf{J}$ and suppose that the token t_x placed at x is $(G^x_C, \mathbf{I}'_i \cap G^x_C)$-rigid. Using a similar argument as in the proof of Lemma 6, one can show that t_x is also $(G^x_C, \mathbf{J} \cap G^x_C)$-rigid. Moreover, for every $\mathbf{J}$ with $\mathbf{I} \overset{G}{\rightsquigarrow} \mathbf{J}$, $\mathbf{J} \cap C = \bigcup_{i=1}^{3} (\mathbf{J} \cap \mathbf{I}'_i \cap C)$. Hence, every token placed at $x \in \mathbf{J} \cap C$ is $(G^x_C, \mathbf{J} \cap G^x_C)$-rigid, which completes the first part of our proof.

(⇒). Assume that C is $(G, \mathbf{I})$-confined. Since $\mathbf{I} \cap C$ is a maximum independent set of C and any token placed at $x \in \mathbf{I} \cap C$ is $(G, \mathbf{I})$-movable, it follows that k must be odd. The construction of $\mathbf{I}'_1$ and $\mathbf{I}'_2$ can be done similar as in the proof of Lemma 6. For constructing $\mathbf{I}'_3$, instead of starting from $\mathbf{I}$, we start from $\mathbf{I}'_1$ as the only TS-sequence we need is the one that slides the token at c_1 to c_k, which can be obtained from the result of checking if the token placed at c_1 is $(G, \mathbf{I}'_1)$-rigid. ◀

Proof of Lemma 9. Assume that $\mathscr{R}(G, \mathbf{I}) = \emptyset$. We modified the function CheckConfinedPath in Algorithm 1 to check if a length-k-cycle $C = c_1 c_2 \dots c_k c_1$ in G is $(G, \mathbf{I})$-confined as follows (see function CheckConfinedCycle in Algorithm 2). If k is even or $|\mathbf{I} \cap C| < \lfloor k/2 \rfloor$ then clearly C is not $(G, \mathbf{I})$-confined. Otherwise, we first check if the token t_x placed at $x \in \mathbf{I} \cap C$ are $(G^x_C, \mathbf{I} \cap G^x_C)$-rigid or not. If some of them does not satisfy the above condition, then we can conclude that C is not $(G, \mathbf{I})$-confined as some token t_x can be slid to a vertex in G^x_C. Otherwise, we call the CheckRigid function (in Algorithm 1) for each vertex in $\mathbf{I} \cap C$. Note that $\mathscr{R}(G, \mathbf{I}) = \emptyset$, thus it must return no and a TS-sequence which then can be used for constructing the described sets $\mathbf{I}'_1, \mathbf{I}'_2$ and $\mathbf{I}'_3$ in Lemma 8. For constructing $\mathbf{I}'_3$, we start from $\mathbf{I}'_1$ instead of $\mathbf{I}$ and hence need to perform checking if the token placed at c_1 is $(G, \mathbf{I}'_1)$-rigid or not beforehand. Next, after constructing these three independent sets, we check for all $y \in C \cap (\mathbf{I}'_i \setminus \mathbf{I})$ $(i = 1, 2, 3)$ whether the token t_y placed at y is $(G^y_C, \mathbf{I}'_i \cap G^y_C)$-rigid. If all of such t_y are $(G^y_C, \mathbf{I}'_i \cap G^y_C)$-rigid, by Lemma 8, we conclude that C is indeed $(G, \mathbf{I})$-confined.

As in the case of Algorithm 1, in Algorithm 2, for each vertex $u \in V(G)$, the CheckRigid function is called at most 5 times: at most one time during the process of checking if it is

Algorithm 1 Check if a token on $u \in \mathbf{I}$ is $(G, \mathbf{I})$-rigid.

Require: A cactus G, an independent set $\mathbf{I}$ of G, and a vertex $u \in \mathbf{I}$.

Ensure: Return YES if the token on u is $(G, \mathbf{I})$-rigid; otherwise, return NO and a TS-sequence $\mathcal{S}_u$ which slides t to some vertex $v \in N_G(u)$.

```
function CheckRigid($G$, $\mathbf{I}$, $u$)        ▷ Check if a token $t$ on $u$ is $(G, \mathbf{I})$-rigid.
    if $N_G(u) = \emptyset$ then
        return YES
    end if
    for all $v \in N_G(u)$ do
        if $\big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I} = \emptyset$ then
            return NO and a TS-sequence $\mathcal{S}_u$ involving the single step of sliding $t$ from $u$ to $v$.
        end if
        for $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$ do
            Let $G' = G - N_G[u]$.
            Let $H(G', w)$ be the component of $G'$ containing $w$.
            CheckRigid($H(G', w)$, $\mathbf{I} \cap H(G', w)$, $w$)
            CheckRigid($G'$, $\mathbf{I} \cap G'$, $w$) $\leftarrow$ CheckRigid($H(G', w)$, $\mathbf{I} \cap H(G', w)$, $w$)
        end for
        if CheckRigid($G'$, $\mathbf{I} \cap G'$, $w$) $=$ NO for any $w \in \big(N_G(v) \setminus \{u\}\big) \cap \mathbf{I}$ then
            for all components $H(G', w)$ with $|N_G(v) \cap H(G', w)| = 2$ do
                Let $C$ be the (unique) cycle in $G$ containing $v, w$.
                CheckConfinedPath($H(G', w)$, $\mathbf{I} \cap H(G', w)$, $C - v$)
            end for
            if CheckConfinedPath($H(G', w)$, $\mathbf{I} \cap H(G', w)$, $C - v$) $=$ NO for any component $H(G', w)$ with $|N_G(v) \cap H(G', w)| = 2$ then
                return NO and a TS-sequence $\mathcal{S}_u$ which slides $t$ from $u$ to $v$.
            end if
        end if
    end for
    return YES
end function
function CheckConfinedPath($G$, $\mathbf{I}$, $P$)
    Let $k$ be the length of $P$.
    if $k$ is even or $|\mathbf{I} \cap P| < \lfloor k/2 \rfloor$ then return NO
    else
        for all $x \in \mathbf{I} \cap P$ do
            if CheckRigid($G^x_P$, $\mathbf{I} \cap G^x_P$, $x$) $=$ NO then return NO
            end if
            CheckRigid($G$, $\mathbf{I}$, $x$) ▷ Must return NO and a TS-sequence which will be used for the construction of $\mathbf{I}'_1$ and $\mathbf{I}'_2$.
        end for
        Construct $\mathbf{I}'_1$ (as in Lemma 6).
        for all $x \in P \cap (\mathbf{I}'_1 \setminus \mathbf{I})$ do
            if CheckRigid($G^x_P$, $\mathbf{I}'_1 \cap G^x_P$, $x$) $=$ NO then return NO
            end if
        end for
        Construct $\mathbf{I}'_2$ (as in Lemma 6).
        for all $x \in P \cap (\mathbf{I}'_2 \setminus \mathbf{I})$ do
            if CheckRigid($G^x_P$, $\mathbf{I}'_2 \cap G^x_P$, $x$) $=$ NO then return NO
            end if
        end for
        return YES
    end if
end function
```

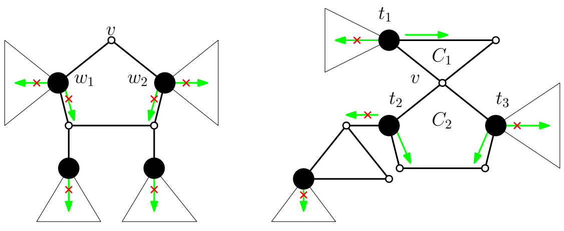
Algorithm 2 Check if a cycle is $(G, \mathbf{I})$-confined.

```
Require: A cactus $G$, an independent set $\mathbf{I}$ of $G$ with $\mathscr{R}(G, \mathbf{I}) = \emptyset$, and a cycle $C$ of $G$.
Ensure: Return YES if $C$ is $(G, \mathbf{I})$-confined; otherwise, return NO.
function CheckConfinedCycle($G$, $\mathbf{I}$, $C$)
    Let $k$ be the length of $C$.
    if $k$ is even or $|\mathbf{I} \cap C| < \lfloor k/2 \rfloor$ then return NO
    else
        for all $x \in \mathbf{I} \cap C$ do
            if CheckRigid($G^x_C$, $\mathbf{I} \cap G^x_C$, $x$) = NO then return NO
            end if
            CheckRigid($G$, $\mathbf{I}$, $x$) ▷ Must return NO (as $\mathscr{R}(G, \mathbf{I}) = \emptyset$) and a TS-sequence
                which will be used for the construction of $\mathbf{I}'_1$, $\mathbf{I}'_2$, and $\mathbf{I}'_3$.
        end for
        Construct $\mathbf{I}'_1$ (as in Lemma 8).
        for all $x \in (\mathbf{I}'_1 \setminus \mathbf{I}) \cap C$ do
            if CheckRigid($G^x_C$, $\mathbf{I}'_1 \cap G^x_C$, $x$) = NO then return NO
            end if
        end for
        Construct $\mathbf{I}'_2$ (as in Lemma 8).
        for all $x \in (\mathbf{I}'_2 \setminus \mathbf{I}) \cap C$ do
            if CheckRigid($G^x_C$, $\mathbf{I}'_2 \cap G^x_C$, $x$) = NO then return NO
            end if
        end for
        CheckRigid($G$, $\mathbf{I}'_1$, $c_1$)
        Construct $\mathbf{I}'_3$ (as in Lemma 8).
        for all $x \in (\mathbf{I}'_3 \setminus \mathbf{I}'_1) \cap C$ do
            if CheckRigid($G^x_C$, $\mathbf{I}'_3 \cap G^x_C$, $x$) = NO then return NO
            end if
        end for
        return YES
    end if
end function
```

$(G, \mathbf{I})$-rigid (and should return NO because of our assumption), at most one time during the process of checking if the token placed at c_1 is $(G, \mathbf{I}'_1)$-rigid and at most three times during the process of checking the conditions described in Lemma 8. Each function CheckRigid takes $O(|G|)$ time for any cactus G (see Lemma 7). Thus, it takes $O(n)$ time to decide if a cycle C is $(G, \mathbf{I})$-confined. Consequently, computing $\mathscr{C}(G, \mathbf{I})$ takes at most $O(n^2)$ time. ◀

C Details of Section 5

Proof of Lemma 11. Assume that there are two vertices w and w' in $N_G(v) \cap \mathbf{I}$ such that the tokens t_w and $t_{w'}$ placed at w and w' are both $(G', \mathbf{I} \cap G')$-rigid, respectively (see Figure 6(a)). From the assumption, t_w and $t_{w'}$ must be $(G, \mathbf{I})$-movable. Therefore, t_w (at least) can be slid to v. But, this can happen only when $t_{w'}$ can be slid to a vertex in $N_{G'}(w')$, i.e., $t_{w'}$ is $(G', \mathbf{I} \cap G')$-movable, which contradicts our assumption. Hence, there is at most one $(G', \mathbf{I} \cap G')$-rigid token in $N_G(v) \cap \mathbf{I}$.



(a) (b)

Figure 6 Illustration for Lemma 11

.

Now, assume that there exists a cycle C containing v such that the path $P = C - v$ is $(G', \mathbf{I} \cap G')$-confined. By Lemma 1, for every independent set $\mathbf{I}'$ with $\mathbf{I} \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{I}'$, $|\mathbf{I} \cap P| = \lfloor k/2 \rfloor$, where k is the length of C. Hence, for every $x \in \mathbf{I} \cap C$, the token on x is at least $(G^x_C, \mathbf{I} \cap G^x_C)$-rigid. Hence, if k is even, it follows that no token can be slid (in G) along edges of C, i.e., all tokens in $\mathbf{I} \cap C$ are $(G, \mathbf{I})$-rigid, which is a contradiction. Therefore, k must be odd. It follows that the tokens in $N_G(v) \cap \mathbf{I} \cap C$ must be $(G', \mathbf{I} \cap G')$-movable. Now, assume for the contradiction that the token $t_{w'}$ at some vertex $w' \in (N_G(v) \cap \mathbf{I}) - C$ which is $(G', \mathbf{I} \cap G')$-rigid. Since $t_{w'}$ is $(G, \mathbf{I})$-movable, it can at least be slid to v. This is a contradiction to Lemma 5(ii). Hence, every tokens in $N_G(v) \cap \mathbf{I}$ must be $(G', \mathbf{I} \cap G')$-movable.

Finally, we claim that if $\mathscr{C}(G, \mathbf{I}) = \emptyset$ then there are at most one cycle C containing v such that the path $P = C - v$ is $(G', \mathbf{I} \cap G')$-confined. Assume for the contradiction that there are two cycles C_1 and C_2 satisfy the above property (see Figure 6(b)). For $i = 1, 2$, since $v \notin \mathbf{I}$ and $\mathbf{I} \cap (C_i - v)$ is a maximum independent set of $C_i - v$, it follows that $\mathbf{I} \cap C_i$ is a maximum independent set of C_i. Additionally, note that $\mathscr{C}(G, \mathbf{I}) = \emptyset$. Thus, there is no $(G, \mathbf{I}, V(C_i))$-confined token $(i = 1, 2)$ placed at any vertex of $\mathbf{I} \cap C_i$. From the assumption, all tokens in $\mathbf{I} \cap (C_i - v) = \mathbf{I} \cap C_i$ are $(G, \mathbf{I}, V(C_i - v))$-confined. On the other hand, since $\mathbf{I} \cap C_1$ is a maximum independent set of C_1, there exists a token t_1 at some vertex $v_1 \in N_{C_1}(v)$. As before, t_1 must be $(G, \mathbf{I}, V(C_1 - v))$-confined and not $(G, \mathbf{I}, V(C_1))$-confined. Therefore, it can be slid to v. Similarly, there exists a token t_2 at some vertex at some vertex $v_2 \in N_{C_2}(v)$ such that t_2 is $(G, \mathbf{I}, V(C_2 - v))$-confined and not $(G, \mathbf{I}, V(C_2))$-confined. Clearly, t_2 must also be slid to v, but this is a contradiction since one need to slide t_1 to a vertex not in $N_G(v)$ first, which can be done (at least) when t_2 has been moved. Note that since $\mathbf{I} \cap C_2$ is a maximum independent set of C_2, there always exists some token in $N_{C_2}(v)$ while no token in $\mathbf{I} \cap C_2$ is moved to a vertex not in $V(C_2)$. Therefore, there are at most one cycle C containing v such that the path $P = C - v$ is $(G', \mathbf{I} \cap G')$-confined. ◀

Proof of Lemma 12. By definition, a token t at $u \in \mathbf{I}$ is $(G, \mathbf{I})$-rigid if for every $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$, $u \in \mathbf{J}$. It follows that t is also $(G, \mathbf{J})$-rigid, since for any independent set $\mathbf{J}'$ such that $\mathbf{J} \overset{G}{\leftrightsquigarrow} \mathbf{J}'$, $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J} \overset{G}{\leftrightsquigarrow} \mathbf{J}'$, which then implies $u \in \mathbf{J}'$. Hence, $\mathscr{R}(G, \mathbf{I}) = \mathscr{R}(G, \mathbf{J})$.

Assume that $\mathscr{R}(G, \mathbf{I}) = \mathscr{R}(G, \mathbf{J}) = \emptyset$. We claim that if $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ then $\mathscr{C}(G, \mathbf{I}) = \mathscr{C}(G, \mathbf{J})$. Suppose that there exists a cycle C of G such that $C \in \mathscr{C}(G, \mathbf{I}) \setminus \mathscr{C}(G, \mathbf{J})$. That is, $\mathbf{I} \cap C$ is a maximum independent set of C, and all tokens in $\mathbf{I} \cap C$ are $(G, \mathbf{I}, V(C))$-confined. By Lemma 1, for every $\mathbf{J}'$ with $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J} \overset{G}{\leftrightsquigarrow} \mathbf{J}'$, $C \cap \mathbf{J}'$ must also be a maximum independent set of C, and the token t_x placed at $x \in \mathbf{J}' \cap C$ is $(G^x_C, \mathbf{J}' \cap G^x_C)$-rigid, i.e., $C \in \mathscr{C}(G, \mathbf{J})$, which is a contradiction. ◀

Proof of Lemma 13. Assume that there exists a TS-sequence $\mathcal{S} = \langle \mathbf{I} = \mathbf{I}_1, \mathbf{I}_2, \dots, \mathbf{J} = \mathbf{J} \rangle$ in G which reconfigures $\mathbf{I}$ to $\mathbf{J}$, and $\mathscr{R}(G, \mathbf{I}) = \mathscr{R}(G, \mathbf{J})$. We show that $\mathbf{I} \cap G' \overset{G'}{\leftrightsquigarrow} \mathbf{J} \cap G'$. Since no tokens can be placed at any neighbor of $\mathscr{R}(G, \mathbf{I}) = \mathscr{R}(G, \mathbf{J}) = \mathscr{R}(G, \mathbf{I}_i)$ $(i = 1, 2, \dots, r)$, for any independent set $\mathbf{I}$ of G, $\mathbf{I} \setminus \mathscr{R}(G, \mathbf{I})$ is indeed an independent set of G'. For any $i \in \{2, \dots, r\}$, let $u \in \mathbf{I}_{i-1} \setminus \mathbf{I}_i$ and $v \in \mathbf{I}_i \setminus \mathbf{I}_{i-1}$. Since $u \notin \mathbf{I}_i$ and $v \notin \mathbf{I}_{i-1}$, both u and v are not in $\mathscr{R}(G, \mathbf{I})$, hence they must be vertices of G'. Therefore, $\mathcal{S}' = \langle \mathbf{I}_1 \setminus \mathscr{R}(G, \mathbf{I}), \mathbf{I}_2 \setminus \mathscr{R}(G, \mathbf{I}), \dots, \mathbf{J} \setminus \mathscr{R}(G, \mathbf{I}) \rangle$ is a TS-sequence in G' which reconfigures $\mathbf{I} \setminus \mathscr{R}(G, \mathbf{I}) = \mathbf{I} \cap G'$ to $\mathbf{J} \setminus \mathscr{R}(G, \mathbf{I}) = \mathbf{J} \cap G'$.

Assume that there exists a TS-sequence $\mathcal{S}' = \langle \mathbf{I}'_1 = \mathbf{I} \cap G', \mathbf{I}'_2, \dots, \mathbf{I}'_s = \mathbf{J} \cap G' \rangle$ in G' which reconfigures $\mathbf{I} \cap G'$ to $\mathbf{J} \cap G'$. By definition of G', it follows that for any independent set $\mathbf{I}'$ of G', $\mathbf{I}' \cup \mathscr{R}(G, \mathbf{I})$ forms an independent set of G. Hence, $\mathcal{S} = \langle \mathbf{I}'_1 \cup \mathscr{R}(G, \mathbf{I}), \mathbf{I}'_2 \cup \mathscr{R}(G, \mathbf{I}), \dots, \mathbf{I}'_s \cup \mathscr{R}(G, \mathbf{I}) \rangle$ is a TS-sequence which reconfigures $\mathbf{I}'_1 \cup \mathscr{R}(G, \mathbf{I}) = \mathbf{I}$ to $\mathbf{I}_s \cup \mathscr{R}(G, \mathbf{I}) = \mathbf{J}$.

We now show that $\mathscr{R}(G', \mathbf{I} \cap G') = \emptyset$. Let $v \in \mathbf{I} \cap G'$. Then, the token t_v placed at v is $(G, \mathbf{I})$-movable, because otherwise $v \in \mathscr{R}(G, \mathbf{I})$. Hence, there exists a TS-sequence $\mathcal{S}$ in G which slides t_v to a vertex $w \in N_G(v)$. Note that $w \in V(G')$. As before, from $\mathcal{S}$, one can construct a TS-sequence $\mathcal{S}'$ in G' which slides t_v to w, hence implies t_v is $(G', \mathbf{I} \cap G')$-movable. Therefore, $\mathscr{R}(G', \mathbf{I} \cap G') = \emptyset$. Similarly, one can also show that $\mathscr{R}(G', \mathbf{J} \cap G') = \emptyset$.

Suppose that $\mathscr{C}(G', \mathbf{I} \cap G') = \mathscr{C}(G', \mathbf{I} \cap G') \neq \emptyset$ and there exists a TS-sequence $\mathcal{S}' = \langle \mathbf{I}'_1 = \mathbf{I} \cap G', \mathbf{I}'_2, \dots, \mathbf{I}'_s = \mathbf{J} \cap G' \rangle$ in G' that reconfigures $\mathbf{I} \cap G'$ to $\mathbf{J} \cap G'$. For $j = 2, \dots, s$, let $u \in \mathbf{I}'_{j-1} \setminus \mathbf{I}'_j$ and $v \in \mathbf{I}'_j \setminus \mathbf{I}'_{j-1}$. Since all tokens in $\mathbf{I} \cap C$ are $(G', \mathbf{I} \cap G', V(C))$-confined, u and v must be either both in G'' or both in some cycle $C \in \mathscr{C}(G', \mathbf{I} \cap G')$ Hence, $\mathcal{S}'' = \langle \mathbf{I}'_1 \cap G'' = \mathbf{I} \cap G'', \mathbf{I}'_2 \cap G'', \dots, \mathbf{I}'_s \cap G'' = \mathbf{J} \cap G'' \rangle$ is a TS-sequence in G'' which reconfigures $\mathbf{I} \cap G''$ to $\mathbf{J} \cap G''$.

Assume that there exists a TS-sequence $\mathcal{S}'' = \langle \mathbf{I}''_1 = \mathbf{I} \cap G'', \mathbf{I}''_2, \dots, \mathbf{I}''_t = \mathbf{J} \cap G'' \rangle$ in G'' which reconfigures $\mathbf{I} \cap G''$ to $\mathbf{J} \cap G''$. We claim that one can construct a TS-sequence $\mathcal{S}'$ in G' which reconfigures $\mathbf{I} \cap G' = (\mathbf{I} \cap G'') \cup (\mathbf{I} \cap \mathscr{C}(G', \mathbf{I} \cap G'))$ to $\mathbf{J} \cap G' = (\mathbf{J} \cap G'') \cup (\mathbf{J} \cap \mathscr{C}(G', \mathbf{I} \cap G'))$. Note that for a given independent set $\mathbf{I}''$ of G'' and a cycle $C \in \mathscr{C}(G', \mathbf{I} \cap G')$, $\mathbf{I}'' \cup \big(\mathbf{I} \cap C\big)$ may not be an independent set of G'. The same observation holds for any independent set that is reconfigurable from $\mathbf{I}$. Let $\mathcal{F}$ be the set of all components of G''. From the previous part, one can construct a TS-sequence $\mathcal{S''}_F = \langle \mathbf{I}''_1 \cap F, \mathbf{I}''_2 \cap F, \dots, \mathbf{I}''_t \cap F \rangle$ for each component $F \in \mathcal{F}$. Let $A = \bigcup_{C \in \mathscr{C}(G', \mathbf{I} \cap G')} \bigcup_{x \in \mathbf{I} \cap C} \big(N_{G'}(x) \setminus V(C)\big)$. For a given component F of G'',

- If $\mathcal{S''}_F$ involves no vertex in A.

  For any independent set $\mathbf{I}_F \in \mathcal{S''}_F$ and any cycle C of G', $\mathbf{I}_F \cup (\mathbf{I} \cap C)$ forms an independent set of G'. It follows that $\mathcal{S''}_F$ can be "extended" to a TS-sequence in G'.
- If $\mathcal{S''}_F$ involves vertices in $A' = A \cap F$ (see Figure 7).

  Let $C \in \mathscr{C}(G', \mathbf{I} \cap G')$. Since G' is a cactus, there is at most one vertex $v \in \mathbf{I} \cap C$ such that $N_{G'}(v) \cap V(F) \neq \emptyset$. Moreover, if there are two vertices $u_1, u_2 \in V(F)$ such that $N_{G'}(u_i) \cap V(C) \neq \emptyset$ $(i = 1, 2)$ then they must both adjacent to v. By definition of $(G', \mathbf{I} \cap G', V(C))$-confined tokens, for each such cycle C described above, there exists a TS-sequence $\mathcal{S}(C, v)$ which slides the token t_v at $v \in \mathbf{I} \cap C$ $(N_{G'}(v) \cap V(F) \neq \emptyset)$ to some vertex w in $N_C(v)$. Now, if there are two of such cycle C, say C_1 and C_2, let v_1 (resp. v_2) be a vertex in $\mathbf{I} \cap C_1$ (resp. $\mathbf{I} \cap C_2$) such that $N_{G'}(v_1) \cap V(F) \neq \emptyset$ (resp. $N_{G'}(v_2) \cap V(F) \neq \emptyset$). Since G is a cactus, $V(G^x_{C_1}) \cap V(G^y_{C_2}) = \emptyset$ for any $x \in V(C_1) \setminus \{v_1\}$ and $y \in V(C_2) \setminus \{v_2\}$. It follows that $\mathcal{S}(C_1, v_1)$ does not involve any vertex that is involved with $\mathcal{S}(C_2, v_2)$ and vice versa.

The TS-sequence $\mathcal{S}'$ thus can be constructed as follows. First of all, we perform any sequence $\mathcal{S''}_F$ that does not involve vertices of A. Next, for a component F such that $\mathcal{S''}_F$ involves some vertex of A, let $C \in \mathscr{C}(G', \mathbf{I} \cap G')$ be such that there exists a vertex $v \in \mathbf{I} \cap C$ satisfying

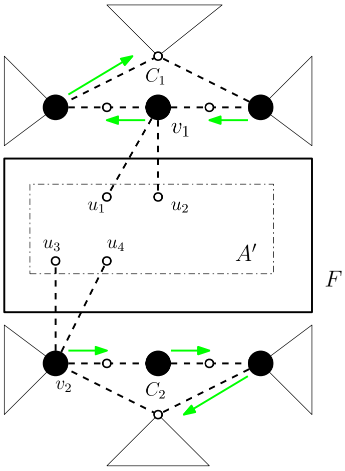


Figure 7 $\mathcal{S}''_F$ involves vertices in $A' \subseteq A$ (Lemma 13).

$N_G(v) \cap V(F) \subseteq A$. As observed before, such a vertex v is uniquely determined. Then, we perform $\mathcal{S}(C, v)$, then perform $\mathcal{S}''_F$, and then perform $\mathcal{S}(C, v)$ in reverse order. If the vertex $w \in N_C(v)$ where the token t_v is slid to after performing $\mathcal{S}(C, v)$ is also in $\mathbf{J}$ then in the step of reversing $\mathcal{S}(C, v)$, we do not reverse the step of sliding t_v to w. At this moment, we have reconfigured $\mathbf{I} \cap G''$ to $\mathbf{J} \cap G''$ in G'. The remaining problem is to reconfigure $\mathbf{I} \cap C$ to $\mathbf{J} \cap C$ in G' for each cycle $C \in \mathscr{C}(G', \mathbf{I} \cap G')$, which can be done using Lemma 4 and the observation that for any vertex $v \in V(C)$, if $v \in \mathbf{J}$ then $N_G(v) \cap \mathbf{J} = \emptyset$.

Using a similar argument as before (based on the fact that if $\mathbf{I}'$ is an independent set of G' then $\mathbf{I}' \cap G''$ is also an independent set of G''), one can show that $\mathscr{R}(G'', \mathbf{I} \cap G'') = \mathscr{R}(G'', \mathbf{J} \cap G'') = \emptyset$, and $\mathscr{C}(G'', \mathbf{I} \cap G'') = \mathscr{C}(G'', \mathbf{J} \cap G'') = \emptyset$. ◀

Proof of Lemma 14. First of all, we claim that if $N_{\mathcal{B}_w}(w) \cap \mathbf{I} = \emptyset$ then one can slide a closest token in G^* to w, where G^* is the graph obtained from G by removing all vertices in $\mathcal{B}_w - w$. In other words, there exists an independent set $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$ and $w \in \mathbf{J}$. If $w \in \mathbf{I}$ then we are done. Thus, assume that $w \notin \mathbf{I}$. Let $w' \in \mathbf{I} \cap G^*$ be such that $\mathsf{dist}_{G^*}(w, w') = \min_{w'' \in \mathbf{I} \cap G^*} \mathsf{dist}_{G^*}(w, w'')$. Let $P = w_1 \dots w_p$ ($p \geq 3$) be a shortest ww'-path with $w_1 = w$ and $w_p = w'$. Let $M = N_{G^*}(w_{p-1}) \cap \mathbf{I}$. Since $N_{\mathcal{B}_w}(w) \cap \mathbf{I} = \emptyset$, it follows that $M = N_{G^*}(w_{p-1}) \cap \mathbf{I} = N_G(w_{p-1}) \cap \mathbf{I}$ for any $p \geq 3$. The definition of w' implies that no tokens are placed at $N_G[w_i]$ for $i = 1, 2, \dots, p-2$. We claim that a token on some vertex of M can be slid to w. If $|M| = 1$, i.e., M contains only w', then one can slide (in G) the token on w' to w directly. If $|M| \geq 2$, then by Lemma 11, there exists at most one vertex z in M such that the token on z is $(G', \mathbf{I} \cap G')$-rigid, where $G' = G - w_{p-1}$ (see Figure 8(a)). On the other hand, if there exists a cycle D containing w_{p-1} such that the path $Q = D - w_{p-1}$ is $(G', \mathbf{I} \cap G')$-confined, then all tokens in M must be $(G', \mathbf{I} \cap G')$-movable (see Figure 8(b)). Note that because $\mathscr{C}(G, \mathbf{I}) = \emptyset$, such a cycle D described above (if exists) must be unique. Also note that by Lemma 5 and the assumption that $\mathscr{R}(G, \mathbf{I}) = \emptyset$, both z and D cannot exist at the same time. If both of them do not exist, we can slide the token $t_{w'}$ placed at w' to w by first sliding all tokens in $M - w'$ (which are clearly $(G', \mathbf{I} \cap G')$-movable) to some vertices in G', and then slide $t_{w'}$ to w. If z exists, we first reduce the number of tokens in M by sliding all tokens in $M - z$ (which are clearly $(G', \mathbf{I} \cap G')$-movable) to some vertices in G', and then slide the token t_z on z to w. On the other hand, if D exists (uniquely), then one can slide a token $t_{z'}$ on $z' \in M \cap D$ to w by first sliding all tokens in $M - C$ (which

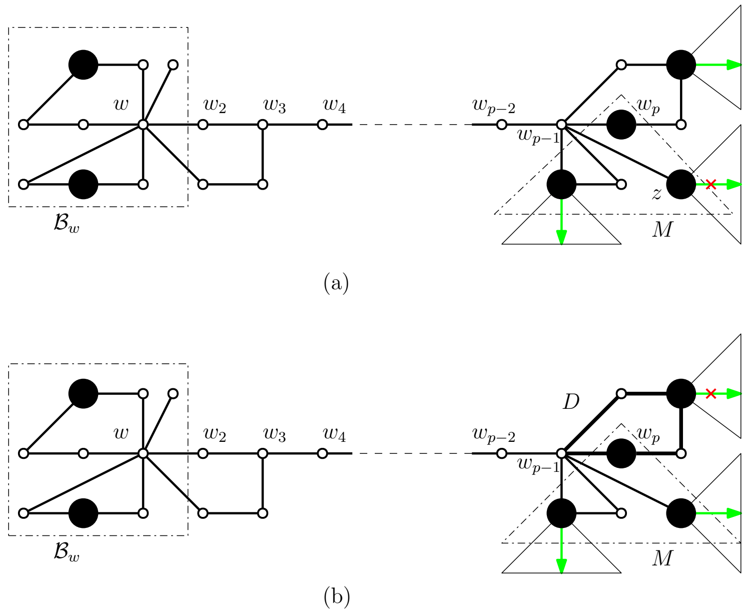


Figure 8 (a) The token t_z at z is $(G', \mathbf{I} \cap G')$-rigid; (b) The cycle D containing w_{p-1} such that the path $Q = D - w_{p-1}$ is $(G', \mathbf{I} \cap G')$-confined.

are clearly $(G', \mathbf{I} \cap G')$-confined) to some vertices in G' then sliding $t_{z'}$ to w_{p-1} (which, by Lemma 11, is the only way of moving $t_{z'}$ "out of" D), and finally to w.

Next, we estimate the maximum number of tokens that can be placed at vertices of $\mathcal{B}_w$. Observe that for any block $B \in \mathcal{B}_w$, since B is either K_2 or a cycle, $B - w$ is indeed a path. Moreover, the path $P = B - w$ satisfies that any token t_x placed at $x \in \mathbf{I} \cap P$ is $(G_P^x, \mathbf{I} \cap G_P^x, V(P))$-confined, simply because in this case G_P^x is the graph contains a single vertex x. By Lemma 11, there is at most one block $B \in \mathcal{B}_w$ that contains $\lfloor |B|/2 \rfloor$ token(s), while all other blocks $B' \neq B$ must contain at most $\lfloor |B'|/2 \rfloor - 1$ token(s). Thus, $|\mathbf{I} \cap \mathcal{B}_w| \leq \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1 \big) + 1$.

Finally, we claim that if $|\mathbf{I} \cap \mathcal{B}_w| \leq \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1 \big)$, then one can "arrange" the tokens in $\mathbf{I} \cap \mathcal{B}_w$ such that there are no tokens placed at vertices of $N_{\mathcal{B}_w}[w]$. More formally, there exists an independent set $\mathbf{J}$ such that $\mathbf{I} \overset{G}{\rightsquigarrow} \mathbf{J}$ and $N_{\mathcal{B}_w}[w] \cap \mathbf{J} = \emptyset$. If there exists a block $B \in \mathcal{B}_w$ such that $|\mathbf{I} \cap B| = \lfloor |B|/2 \rfloor$ then since $|\mathbf{I} \cap \mathcal{B}_w| \leq \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1 \big)$, there must be another block $B' \in \mathcal{B}_w$ where $|B' \cap \mathbf{I}| < \lfloor |B'|/2 \rfloor - 1$. Since $\mathscr{R}(G, \mathbf{I}) = \emptyset$ and $\mathscr{C}(G, \mathbf{I}) = \emptyset$, one can slide a token from B to w (if there is no token at w) and then slide it to a vertex in B'. If there is a token at w, we slide it to a vertex in B' directly. Since at most one such block B exists, we can now assume that $|\mathbf{I} \cap B| \leq \lfloor |B|/2 \rfloor - 1$ for every block $B \in \mathcal{B}_w$. Clearly, a block $B \in \mathcal{B}_w$ contains a token only when $|B| \geq 4$, i.e., it is a cycle of length at least 4. Using Lemma 4 and note that all blocks $B \in \mathcal{B}_w$ are safe, one can easily obtain the described set $\mathbf{J}$.

Using the above claims, we now prove Lemma 14.

(i) Assume that $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1 \big) = 0$. Since $|B| \geq 2$ for any block B of G, it follows that for all $B \in \mathcal{B}_w$, $2 \leq |B| \leq 3$, i.e., B is either K_2 or a cycle of length 3. Clearly, $N_{\mathcal{B}_w}(w) = V(\mathcal{B}_w) \setminus \{w\}$.
Now, for a safe vertex $v \in V(\mathcal{B}_w)$, one must have that $v \in N_{\mathcal{B}_w}(w) \subseteq N_G(w)$. If $v \in \mathbf{I}$ then we are done. Therefore, assume that $v \notin \mathbf{I}$. Note that in this case $|\mathbf{I} \cap \mathcal{B}_w| \leq 1$. If $|\mathbf{I} \cap \mathcal{B}_w| = 0$ then by the first claim above, one can slide a token to w, and then to v.

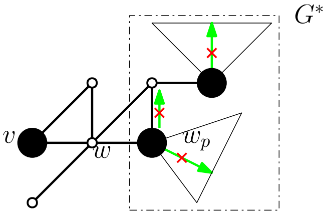


Figure 9 Illustration of **Case (i)-1** of Lemma 15(i).

Otherwise, if $w \in \mathbf{I}$, then clearly the token placed at w can be slid to v. On the other hand, if there is a vertex $v' \notin \{v, w\}$ where $v' \in \mathbf{I} \cap \mathcal{B}_w$ then since $\mathscr{R}(G, \mathbf{I}) = \emptyset$ and $\mathscr{C}(G, \mathbf{I}) = \emptyset$, it follows that the token placed at v' can be slid to a vertex outside the block containing v' and w, therefore must be slid to w (which is the unique cut vertex of G in $\mathcal{B}_w$), and then can be slid to v from w.

(ii) Assume that $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) \geq 1$. If $|\mathbf{I} \cap \mathcal{B}_w| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$ then we can just simply use the third claim to "arrange" the tokens in $\mathbf{I} \cap \mathcal{B}_w$.
If $|\mathbf{I} \cap \mathcal{B}_w| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) + 1$ then there must exist a unique token t in $N_{\mathcal{B}_w}[w]$ which cannot be "arranged" using the third claim. Note that in this case $|\mathbf{I} \cap (\mathcal{B}_w - w)| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$. If t is placed at w then $N_{\mathcal{B}_w}(w) \cap \mathbf{I} = \emptyset$ and we are done. If t is placed at some vertex in $N_{\mathcal{B}_w}(w)$ then it can be slid to w because $\mathscr{R}(G, \mathbf{I}) = \emptyset$ and $\mathscr{C}(G, \mathbf{I}) = \emptyset$. By sliding t to w, there is now no token placed at any vertex in $N_{\mathcal{B}_w}(w)$, and the resulting independent set is the set $\mathbf{I}'$ we need.
Hence, we can assume that $|I \cap \mathcal{B}_w| < \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$. We claim that one can construct an independent set $\mathbf{I}'$ such that $\mathbf{I} \overset{G}{\rightsquigarrow} \mathbf{I}'$, $N_{\mathcal{B}_w}(w) \cap \mathbf{I}' = \emptyset$, and $|\mathbf{I}' \cap (\mathcal{B}_w - w)| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$. Using the third claim, we can assume without loss of generality that $N_{\mathcal{B}_w}[w] \cap \mathbf{I} = \emptyset$. We construct the set $\mathbf{I}'$ using TS rule as follows. While the number of tokens in $\mathcal{B}_w - w$ is smaller than $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$, we use the first claim to move some token t not in $\mathcal{B}_w - w$ to w, then move t to some block $B \in \mathcal{B}_w$ which contains less than $\lfloor |B|/2 \rfloor - 1$ token(s), then using the third claim to "arrange" the set of tokens in $\mathcal{B}_w$ so that $N_{\mathcal{B}_w}[w]$ contains no token. Repeat the above steps until the number of tokens in $\mathcal{B}_w$ is equal to $\sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big)$, we finally obtain $\mathbf{I}'$. ◀

Proof of Lemma 15.

(i) First of all, we claim that $\mathscr{R}(G^*, \mathbf{I}^*) = \emptyset$. Assume for the contradiction that $\mathscr{R}(G^*, \mathbf{I}^*) \neq \emptyset$. Let $w' \in \mathbf{I}^*$ be a vertex where a $(G^*, \mathbf{I}^*)$-rigid token is placed. Let $P = w_1 w_2 \dots w_p$ be a vw'-path with $w_1 = v$, $w_2 = w$ and $w_p = w'$.

Case (i)-1: $w_{p-1} = w$. (See Figure 9)
In this case, it is clear that $\mathsf{dist}_G(w, w_p) = 1$. From Lemma 14, any block $B \in \mathcal{B}_w$ is either K_2 or a cycle of length 3. Let B be the safe block containing v. If B is K_2 then clearly the token t_v placed at v is $(G - w, \mathbf{I} \cap (G - w))$-rigid. On the other hand, if B is a cycle of length 3 then the path $B - w$ is clearly $(G - w, \mathbf{I} \cap (G - w))$-confined. By Lemma 11, in any of these two cases, the token t_{w_p} placed at $w_p = w_3 \in N_G(w)$ must be $(G - w, \mathbf{I} \cap (G - w))$-movable. By definition, G^* is indeed a connected component of $G - w$ and $\mathbf{I}^* = \mathbf{I} \cap G^* = (\mathbf{I} - v) \cap (G - w)$. Hence, t_{w_p} must be $(G^*, \mathbf{I} \cap G^*)$-movable, which is a contradiction.

Case (i)-2: $w_{p-2} = w$. (See Figure 10.) In this case, we can assume that any $(G^*, \mathbf{I}^*)$-rigid token is of distance (in G) at least 2 from w (which then implies $\mathsf{dist}_G(w, w_p) = 2$

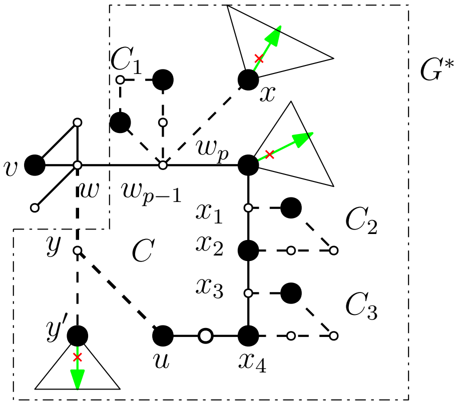


Figure 10 Illustration of **Case (i)-2** of Lemma 15(i).

in this case) since if otherwise then we back to **Case i-(1)** and claim that there must be some contradiction.

Suppose that there exists a cycle C_1 in G^* such that $w_{p-1} \in V(C_1)$, $w_p \notin V(C_1)$, and the path $P_1 = C_1 - w_{p-1}$ is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-confined. Let $H(G^* - N_{G^*}[w_p], P_1)$ be the component of $G^* - N_{G^*}[w_p]$ containing P_1. Since G is a cactus, it follows that $N_G(w) \cap H(G^* - N_{G^*}[w_p], P_1) = \emptyset$. Hence, $H(G^* - N_{G^*}[w_p], P_1)$ must also be a component of $G - N_G[w_p]$. Therefore, C_1 satisfies that $w_{p-1} \in V(C_1)$, $w_p \notin V(C_1)$, and the path $P_1 = C_1 - w_{p-1}$ is $(G - N_G[w_p], \mathbf{I} \cap (G - N_G[w_p]))$-confined. It follows that the token t_{w_p} placed at w_p cannot be slid in G to w_{p-1}. Note that Lemma 11 implies that C_1 is uniquely determined. Since t_{w_p} is $(G, \mathbf{I})$-movable, it follows that there exists a vertex $x_1 \in N_G(w_p) \setminus \{w_{p-1}\}$ such that t_{w_p} can be slid in G to x_1. Since t_{w_p} is $(G^*, \mathbf{I}^*)$-rigid, it follows that $\big(N_{G^*}(x_1) \setminus \{w_p\}\big) \cap \mathbf{I}^* = \big(N_G(x_1) \setminus \{w_p\}\big) \cap \mathbf{I} \neq \emptyset$.

Let $x_2 \in N_{G^*}(x_1) \setminus \{w_p\}\big) \cap \mathbf{I}^*$. Now, if there exists a cycle C_2 in G^* such that $\{x_1, x_2\} \subseteq V(C_2)$, $w_p \notin V(C_2)$, and the path $P_2 = C_2 - x_1$ is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-confined, then using the same argument as with P_1, it follows that t_{w_p} cannot be slid in G to x_1, which contradicts our assumption. Therefore, for any $x_2 \in N_{G^*}(x) \setminus \{w_p\}\big) \cap \mathbf{I}^*$, such a cycle C_2 does not exist.

Hence, there must be some $x_2 \in N_{G^*}(x) \setminus \{w_p\}\big) \cap \mathbf{I}^*$ such that the token t_{x_2} placed at x_2 must be $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-rigid, and hence also $(G^*, \mathbf{I}^*)$-rigid since t_{w_p} is also $(G^*, \mathbf{I}^*)$-rigid. On the other hand, since t_{x_2} is $(G, \mathbf{I})$-movable, it follows that the component $H(G^* - N_{G^*}[w_p], x_2)$ of $G^* - N_{G^*}[w_p]$ containing x_2 must not be a component of $G - N_G[w_p]$, which then implies that $w \in V(H(G - N_G[w_p], x_2))$, where $H(G - N_G[w_p], x_2)$ is the component of $G - N_G[w_p]$ containing x_2. Hence, there exists a cycle C in G containing w, w_{p-1}, w_p, x_1 and x_2. As G is a cactus, the cycle C is unique.

Let $x_3 \neq x_1$ be another neighbor of x_2 in C. Using a similar argument as with C_1, one can show that there does not exist any cycle C_3 in G^* such that $x_3 \in V(C_3)$, $x_2 \notin V(C_3)$, and the path $P_3 = C_3 - x_3$ is $(G^* - N_{G^*}[y], \mathbf{I}^* \cap (G^* - N_{G^*}[x_2]))$-confined. Note that in such cycle C_3 described above, $V(C_3) \cap V(C) = \{x_3\}$. Hence, there must be some $x_4 \in \big(N_{G^*}(x_3) \setminus \{x_2\}\big) \cap \mathbf{I}^*$ such that the token t_{x_4} placed at x_4 is $(G^* - N_{G^*}[x_2], \mathbf{I}^* \cap (G^* - N_{G^*}[x_2]))$-rigid, and hence $(G^*, \mathbf{I}^*)$-rigid as t_{x_2} is also $(G^*, \mathbf{I}^*)$-rigid. On the other hand, since t_{x_4} is $(G, \mathbf{I})$-movable, it follows that the component $H(G^* - N_{G^*}[x_2], x_4)$ of $G^* - N_{G^*}[x_2]$ containing x_4 must not be a component of $G - N_G[x_2]$, which then implies that $w \in V(H(G - N_G[x_2], x_4))$, where $H(G - N_G[x_2], x_4)$ is the component of $G - N_G[x_2]$ containing x_4. Since G is a cactus, it must happen that $x_4 \in V(C)$. Repeat the arguments with vertices of C, we finally obtain that there must be some $(G^*, \mathbf{I}^*)$-rigid token placed at a vertex $u \in V(C)$ of distance 1 or 2 from w (in G). Since $\mathsf{dist}_G(w, w_p) = 2$ and t_{w_p} is a closest $(G^*, \mathbf{I}^*)$-rigid token to w, no $(G^*, \mathbf{I}^*)$-rigid token can be placed at some vertex of distance 1 from w. Thus, $\mathsf{dist}_G(w, u) = 2$.

Hence, without loss of generality, we now can assume that there does not exist any cycle C_1 in G^* such that $w_{p-1} \in V(C_1)$, $w_p \notin V(C_1)$, and the path $P_1 = C_1 - w_{p-1}$ is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-confined (if such cycle C_1 exists, then find such vertex u described above and regard it as w_p). Since t_{w_p} is $(G^*, \mathbf{I}^*)$-rigid, there must be some vertex $x \in \big(N_{G^*}(w_{p-1}) \setminus \{w_p\}\big) \cap \mathbf{I}^*$ such that the token t_x placed at x is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-rigid, and hence also $(G^*, \mathbf{I}^*)$-rigid as t_{w_p} is $(G^*, \mathbf{I}^*)$-rigid. Thus, both t_{w_p} and t_x are $(G^* - w_{p-1}, \mathbf{I}^* \cap (G^* - w_{p-1}))$-rigid. Since all tokens in $\mathbf{I}$ are $(G, \mathbf{I})$-movable and $w_{p-1} \notin \mathbf{I}$, by Lemma 11, it follows that at most one of the two tokens t_{w_p} and t_x is $(G - w_{p-1}, \mathbf{I} \cap (G - w_{p-1}))$-rigid. Without loss of generality, assume t_{w_p} is not $(G - w_{p-1}, \mathbf{I} \cap (G - w_{p-1}))$-rigid. Hence, it must happen that $w \in V(H(G - w_{p-1}, w_p))$, where $H(G - w_{p-1}, w_p)$ is the component of $G - w_{p-1}$ containing w_p. Thus, there exists a (unique) cycle C in G containing w and w_p. Now, let $H(G^* - w_{p-1}, x)$ and $H(G^* - w_{p-1}, w_p)$ be the components of $G^* - w_{p-1}$ containing x and w_{p-1}, respectively. As $H(G^* - w_{p-1}, w_p)$ is not a component of $G - w_{p-1}$, it follows that $H(G^* - w_{p-1}, x)$ is a component of $G - w_{p-1}$, that is, $H(G^* - w_{p-1}, x) = H(G - w_{p-1}, x)$ because if otherwise, $w \in V(H(G - w_{p-1}, x))$, which contradicts to the fact that G is a cactus. Hence, t_x is indeed $(G - w_{p-1}, \mathbf{I} \cap (G - w_{p-1}))$-rigid, which means that t_{w_p} cannot be slid in G to w_{p-1}.

Let $x_1 \in N_G(w_p) \setminus \{w_{p-1}\}$ be a neighbor of w_p such that t_{w_p} can be slid in G to x_1. If $x_1 \notin V(C)$ then since t_{w_p} is $(G^*, \mathbf{I}^*)$-rigid and $(G, \mathbf{I})$-movable, it must happen that $w \in H(G - w_p, x_1)$, which is a contradiction as G is a cactus. Hence, $x_1 \in V(C)$. As before, one can show that there exists a vertex $x_2 \in \big(N_{G^*}(x_1) \setminus \{w_p\}\big) \cap \mathbf{I}^*$ which is $(G^*, \mathbf{I}^*)$-rigid and $(G, \mathbf{I})$-movable, and hence must be in $V(C)$. Repeat the arguments, we finally obtain that there must be some $(G^*, \mathbf{I}^*)$-rigid token placed at some vertex in $V(C)$ of distance 2 (in G) from w, say u, which is different from w_p and x. Now, let y be the common neighbor of w and u. As the token t_u placed at u is $(G^*, \mathbf{I}^*)$-rigid, there exists some vertex $y' \in \big(N_{G^*}(y) \setminus \{u\}\big) \cap \mathbf{I}^*$ such that the token $t_{y'}$ placed at y' is $(G^* - N_{G^*}[u], \mathbf{I}^* \cap (G^* - N_{G^*}[u]))$-rigid, and hence $(G^*, \mathbf{I}^*)$-rigid as t_u is $(G^*, \mathbf{I}^*)$-rigid. Let $H(G^* - N_{G^*}[u], y')$ be the component of $G^* - N_{G^*}[u]$ containing y'. Since $t_{y'}$ is $(G, \mathbf{I})$-movable, $H(G^* - N_{G^*}[u], y')$ is not a component of $G - N_G[u]$, which means that $w \in H(G - N_G[u], y')$. But this is a contradiction as G is a cactus.

Case (i)-3: $w_{p-1} \neq w$ and $w_{p-2} \neq w$. (See Figure 11.)

As before, one can assume that any $(G^*, \mathbf{I}^*)$-rigid token is of distance (in G) at least 3 from w. Assume that there exists a cycle C_1 such that $w_{p-1} \in V(C_1)$, $w_p \notin V(C_1)$, $w_{p-2} \notin V(C_1)$, and the path $P_1 = C_1 - w_{p-1}$ is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-confined. As in **Case (i)-2**, one can show that there must be a $(G^*, \mathbf{I}^*)$-rigid token placed at some vertex of distance 1 or 2 (in G) from w, which then leads to a contradiction. Hence, such a cycle C_1 does not exist.

Now, consider a (unique) cycle C_2 such that $\{w_{p-1}, w_{p-2}\} \subseteq V(C_2)$, $w_p \notin V(C_2)$, and the path $P_2 = C_2 - w_{p-1}$ is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-confined.

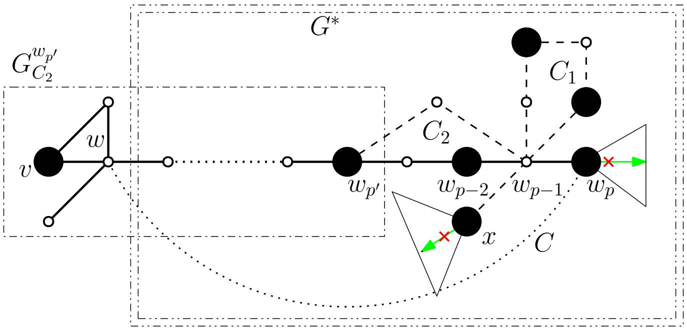


Figure 11 Illustration of **Case (i)-3** of Lemma 15(i).

Firs, assume that it does not exist. Since t_{w_p} is $(G^*, \mathbf{I}^*)$-rigid, there must be some vertex $x \in \big(N_{G^*}(w_{p-1}) \setminus \{w_p\}\big) \cap \mathbf{I}^*$ such that the token t_x placed at x is $(G^* - N_{G^*}[w_p], \mathbf{I}^* \cap (G^* - N_{G^*}[w_p]))$-rigid, and hence also $(G^*, \mathbf{I}^*)$-rigid as t_{w_p} is $(G^*, \mathbf{I}^*)$-rigid. As before, at most one of the two tokens t_{w_p} and t_x is $(G - w_{p-1}, \mathbf{I} \cap (G - w_{p-1}))$-rigid. Without loss of generality, assume that t_{w_p} is not $(G - w_{p-1}, \mathbf{I} \cap (G - w_{p-1}))$-rigid. Hence, it must happen that $w \in V(H(G - w_{p-1}, w_p))$, where $H(G - w_{p-1}, w_p)$ is the component of $G - w_{p-1}$ containing w_p. Thus, there exists a (unique) cycle C in G containing w and w_p. Using a similar argument as in the previous part, one can show that this will lead to a contradiction.

Therefore, such a cycle C_2 described above must exist. Let p' be the smallest index ($1 \leq p' \leq p - 1$) such that $w_{p'} \in V(C_2) \cap V(P)$. Using Lemma 8 and the fact that for any $x \in V(C_2) \setminus \{w_{p'}\}$, ${G^*}^x_{C_2} = G^x_{C_2}$ (i.e., $w \in G^{w_{p'}}_{C_2}$), we can thus assume that $w_{p'} \in \mathbf{I}$ and the token $t_{w_{p'}}$ placed at $w_{p'}$ is $({G^*}^{w_{p'}}_{C_2}, \mathbf{I}^* \cap {G^*}^{w_{p'}}_{C_2})$-rigid and $(G^{w_{p'}}_{C_2}, \mathbf{I} \cap G^{w_{p'}}_{C_2})$-movable. Replace G by $G^{w_{p'}}_{C_2}$, the independent set $\mathbf{I}$ by $\mathbf{I} \cap G^{w_{p'}}_{C_2}$, and w_p by $w_{p'}$ in the previous arguments, one can then either obtain a contradiction (when $\mathsf{dist}_G(w, w_{p'}) \leq 2$) or repeat the arguments once more time (when $\mathsf{dist}_G(w, w_{p'}) \geq 3$). Hence, we can now conclude that $\mathscr{R}(G^*, \mathbf{I}^*) = \emptyset$.

Next, we claim that $\mathscr{C}(G^*, \mathbf{I}^*) = \emptyset$. Assume that it is not empty, i.e., there exists a cycle $C^* \in \mathscr{C}(G^*, \mathbf{I}^*)$. Note that C^* is also a cycle of G, and $\mathbf{I} \cap C^* = \mathbf{I}^* \cap C^*$, which means that $\mathbf{I} \cap C^*$ is also a maximum independent set of C^*. Without loss of generality, using Lemma 8, we can assume that there is some token t_x placed at a vertex $x \in \mathbf{I} \cap C^*$ such that t_x is $(G^x_{C^*}, \mathbf{I} \cap G^x_{C^*})$-movable but $({G^*}^x_{C^*}, \mathbf{I}^* \cap {G^*}^x_{C^*})$-rigid. It follows that $w \in V(G^x_{C^*})$. Since any TS-sequence in $G^x_{C^*}$ can indeed be extended to a TS-sequence in G (see the proof of Lemma 1), it follows that $\mathscr{R}(G^x_{C^*}, \mathbf{I} \cap G^x_{C^*}) = \emptyset$. Additionally, using the previous part, one can show that the removal of vertices in $\mathcal{B}_w$ from $G^x_{C^*}$ does not result any new rigid token in the obtained graph ${G^*}^x_{C^*}$, which clearly contradicts the assumption that t_x is $({G^*}^x_{C^*}, \mathbf{I}^* \cap {G^*}^x_{C^*})$-rigid.

(ii) We first claim that $\mathscr{R}(G^*, \mathbf{I}^*) = \emptyset$. Assume for the contradiction that $\mathscr{R}(G^*, \mathbf{I}^*) \neq \emptyset$. Let $w' \in \mathbf{I}^*$ be a vertex where a $(G^*, \mathbf{I}^*)$-rigid token is placed. Let $Q = w_1 w_2 \dots w_q$ be a ww'-path with $w_1 = w$ and $w_q = w'$ ($q \geq 1$).

Case (ii)-1: $w_q = w$. First, assume that $N_{\mathcal{B}_w}(w) \subseteq N_G[\mathbf{I} \cap (\mathcal{B}_w - w)]$. Also note that in this case $|\mathbf{I} \cap \mathcal{B}_w| = \sum_{B \in \mathcal{B}_w} \big(\lfloor |B|/2 \rfloor - 1\big) + 1$. It follows that the token t_w placed at w cannot be slid (in G) to any vertex in $N_{\mathcal{B}_w}(w)$. Let $\mathcal{S} = \langle \mathbf{I}_1 = \mathbf{I}, \mathbf{I}_2, \dots, \mathbf{I}_\ell \rangle$ be a TS-sequence which slides t_w to some vertex in $N_{G^*}(w)$. Since w is the unique cut vertex in $\mathcal{B}_w$ and $|\mathbf{I} \cap \mathcal{B}_w|$ is maximum, $\mathcal{S}$ does not involve any vertex in $\mathbf{I} \cap (\mathcal{B}_w - w)$, i.e., for any

$\mathbf{J} \in \mathcal{S}$, $(\mathbf{I} \cap (\mathcal{B}_w - w)) \subseteq \mathbf{J}$. (Roughly speaking, no token in $\mathcal{B}_w$ can "move out" while t_w "stay" in w). Hence, $\mathcal{S}' = \langle \mathbf{I}_1 \setminus (\mathbf{I} \cap (\mathcal{B}_w - w)), \mathbf{I}_2 \setminus (\mathbf{I} \cap (\mathcal{B}_w - w)), \dots, \mathbf{I}_\ell \setminus (\mathbf{I} \cap (\mathcal{B}_w - w)) \rangle$ is a TS-sequence in G^* which slides t_w to a vertex in $N_{G^*}(w)$, which is clearly a contradiction. Hence, $N_{\mathcal{B}_w}(w) \subsetneq N_G[\mathbf{I} \cap (\mathcal{B}_w - w)]$. It follows that there exists some vertex $x \in N_{\mathcal{B}_w}(w) \cap V(G^*)$. From the definition of G^* and $\mathbf{I} \cap N_{\mathcal{B}_w}(w) = \emptyset$, we must have $N_{G^*}(x) \cap \mathbf{I} = \{w\}$, i.e., t_w can be directly slid to x in G^*, which is a contradiction.

Case (ii)-2: $w_{q-1} = w$. Without loss of generality, we assume that no $(G^*, \mathbf{I}^*)$-rigid token is placed at w. Assume that there exists a cycle C_1 in G^* such that $w_q \notin V(C_1)$, $w_{q-1} \in V(C_1)$, and the path $P_1 = C_1 - w_{q-1}$ is $(G^* - N_{G^*}[w_q], \mathbf{I} \cap (G^* - N_{G^*}[w_q]))$-confined. Let $H(G^* - N_{G^*}[w_q], P_1)$ be the component of $G^* - N_{G^*}[w_q]$ containing P_1. Since all vertices in $N_G[\mathbf{I} \cap (\mathcal{B}_w - w)]$ are non-cut, $H(G^* - N_{G^*}[w_q], P_1)$ is also a component of $G - N_G[w_q]$, i.e., the token t_{w_q} placed at w_q cannot be slid to w in G. Using a similar argument as in case **i-(2)**, one can indeed assume that such cycle C_1 does not exist and then derive some contradiction.

Case (ii)-3: $w_{q-2} = w$. Similar as in **Case (i)-3**, one can argue that there does not exist any cycle C_1 such that $w_{q-1} \in V(C_1)$, $w_q \notin V(C_1)$, $w_{q-2} \notin V(C_1)$, and the path $P_1 = C_1 - w_{q-1}$ is $(G^* - N_{G^*}[w_q], \mathbf{I} \cap (G^* - N_{G^*}[w_q]))$-confined. On the other hand, there must be some C_2 with $\{w_{q-1}, w_{q-2}\} \subseteq V(C_2)$, $w_q \notin V(C_2)$ and the path $P_2 = C_2 - w_{q-1}$ is $(G^* - N_{G^*}[w_q], \mathbf{I} \cap (G^* - N_{G^*}[w_q]))$-confined. As in **Case i-(3)**, we assume that $\mathscr{R}(G^w_{C_2}, \mathbf{I} \cap G^w_{C_2}) = \emptyset$ and argue with the triple $(G^w_{C_2}, \mathbf{I} \cap G^w_{C_2}, w)$ instead of $(G, \mathbf{I}, w_q)$ and immediately derive the contradiction because of **Case ii-(1)**.

Case (ii)-4: $w_{q-1} \neq w$ and $w_{q-2} \neq w$. One can use a similar argument as in **Case (i)-3** to claim that some contradiction must happen.

Using a similar argument as in part (i), one can also show that $\mathscr{C}(G^*, \mathbf{I}^*) = \emptyset$. ◀

Proof of Lemma 16. The only-if-part is trivial. We claim the if-part, i.e., if $|\mathbf{I}| = |\mathbf{J}|$ then $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{J}$. In order to show this, we claim that there is some independent set $\mathbf{I}^*$ such that $\mathbf{I} \overset{G}{\leftrightsquigarrow} \mathbf{I}^*$ and $\mathbf{J} \overset{G}{\leftrightsquigarrow} \mathbf{I}^*$. The following algorithm constructs such $\mathbf{I}^*$. Initially, $\mathbf{I}^* = \emptyset$.

- Pick a cut vertex w with $\mathcal{B}_w \neq \emptyset$.
- If $\sum_{B \in \mathcal{B}_w} \left(\lfloor |B|/2 \rfloor - 1\right) = 0$, pick a safe vertex $v \in V(\mathcal{B}_w)$, using Lemma 14(i), slide a token in $\mathbf{I}$ and a token in $\mathbf{J}$ to v. Let $\mathbf{I}' = \mathbf{I} \setminus \{v\}$ and $\mathbf{J}' = \mathbf{J} \setminus \{v\}$. Add v to $\mathbf{I}^*$. Remove all vertices in $\mathcal{B}_w$ and let G' be the resulting graph.
- If $\sum_{B \in \mathcal{B}_w} \left(\lfloor |B|/2 \rfloor - 1\right) \geq 1$, using Lemma 14($ii$), slide tokens of $\mathbf{I}$ and tokens of $\mathbf{J}$ to the vertices in $\mathcal{B}_w$. Using Lemma 4, for each block $B \in \mathcal{B}_w$, exhaustively place the tokens at the vertices $v_2[B], v_4[B], \dots$. Let $\mathbf{I}' = \mathbf{I} \setminus (\mathcal{B}_w - w)$ and $\mathbf{J}' = \mathbf{J} \setminus (\mathcal{B}_w - w)$. Add the vertices in $\mathcal{B}_w$ where tokens are placed to $\mathbf{I}^*$. Remove all vertices in $N_G[\mathbf{I}^* \cap (\mathcal{B}_w - w)]$. Let G' be the resulting graph.
- Repeat the above steps with the new triple $(G', \mathbf{I}', \mathbf{J}')$. The algorithm stops when there are no tokens to move.

The correctness of this algorithm is followed from Lemma 14 and Lemma 15. ◀

Complexity of Distributions and Average-Case Hardness[*]

Dmitry Itsykson[1], Alexander Knop[2], and Dmitry Sokolov[3]

1 **St. Petersburg Department of V. A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**
dmitrits@pdmi.ras.ru

2 **St. Petersburg Department of V. A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**
aaknop@gmail.com

3 **St. Petersburg Department of V. A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**
sokolov.dmt@gmail.com

Abstract

We address the following question in the average-case complexity: does there exists a language L such that for all easy distributions D the distributional problem (L, D) is easy on the average while there exists some more hard distribution D' such that (L, D') is hard on the average? We consider two complexity measures of distributions: the complexity of sampling and the complexity of computing the distribution function.

For the complexity of sampling of distribution, we establish a connection between the above question and the hierarchy theorem for sampling distribution recently studied by Thomas Watson. Using this connection we prove that for every $0 < a < b$ there exist a language L, an ensemble of distributions D samplable in $n^{\log^b n}$ steps and a linear-time algorithm A such that for every ensemble of distribution F that samplable in $n^{\log^a n}$ steps, A correctly decides L on all inputs from $\{0,1\}^n$ except for a set that has infinitely small F-measure, and for every algorithm B there are infinitely many n such that the set of all elements of $\{0,1\}^n$ for which B correctly decides L has infinitely small D-measure.

In case of complexity of computing the distribution function we prove the following tight result: for every $a > 0$ there exist a language L, an ensemble of polynomial-time computable distributions D, and a linear-time algorithm A such that for every computable in n^a steps ensemble of distributions F, A correctly decides L on all inputs from $\{0,1\}^n$ except for a set that has F-measure at most $2^{-n/2}$, and for every algorithm B there are infinitely many n such that the set of all elements of $\{0,1\}^n$ for which B correctly decides L has D-measure at most 2^{-n+1}.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases average-case complexity, hierarchy theorem, sampling distributions, diagonalization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.38

1 Introduction

This paper is devoted to average-case complexity. In the average case settings, every computational problem is supplied with an ensemble of distributions on inputs. The problem

[*] The research presented in Section 3 was supported by Russian Science Foundation (project 16-11-10123). The third author was partially supported by "Dynasty" foundation.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 38; pp. 38:1–38:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is easy on the average if it can be solved efficiently on all but a small fraction (according to the distribution) of the inputs.

The paper [6] gave an example of a noncomputable ensemble of distributions such that every language with that ensemble of distributions is easy on the average iff it is easy in the worst case. This explains why the average-case complexity studies not all but only feasible ensembles of distributions. The most natural class of ensembles of distributions is the class of polynomial-time samplable distributions. Such distributions are distributions of outputs of polynomial-time randomized algorithms. The second important class of ensembles of distributions is the class of polynomial-time computable ensembles of distributions. An ensemble of distributions is computable in polynomial time if it's cumulative distribution function is computable in polynomial time. It is known that every polynomial-time computable ensemble of distribution is polynomial-time samplable but the opposite is not true if one-way functions exist [2].

It is well known that several hard problems can be efficiently solved on almost all inputs for some natural distributions. For example the **NP**-complete problem Hamiltonian Path is decidable in a linear time on almost all inputs according to the uniform distributions on the graphs [3]. Another interesting example is the Graph Isomorphism problem that is solvable in linear time on almost all inputs in the case of the uniform distribution on the inputs [1], while there exists much more tricky distribution (see for example [8]) such that there are no known polynomial-time algorithms that solve the graph isomorphism problem with high probability.

Statement of the problem. The standard time hierarchy theorem in the average-case settings states that for all $a > 0$ there exists a distributional problem (L, D) such that every $g(n)$-time algorithm errs on almost all (or on a significant fraction of) inputs according to D but there exists an algorithm with slightly bigger running time $f(n)$ that correctly solves the problem on almost all inputs according to D. For deterministic algorithms, it is easy to show by the straightforward diagonalization that there exists a language L that is decidable in n^a steps but every algorithm with running time $O(n^{a-\epsilon})$ gives incorrect answer on all sufficiently large inputs for $\epsilon > 0$. For randomized algorithms with bounded error Pervyshev [7] showed that for all $b > 0$ there exists a language L with the uniform distribution that is decidable in randomized polynomial time with a bounded error on all but ϵ fraction of inputs but every randomized algorithm with running time n^b gives incorrect answer with high probability on at least $\frac{1}{2} - \epsilon$ fraction of inputs for all $\epsilon > 0$. The paper [5] showed that the fraction of hard instances in the Pervyshev's result may be improved to $1 - \frac{1}{k} - \epsilon$ if we switch from languages to k-valued functions.

In this paper we study the dual question: is it possible that a language suddenly transits from very average-case easy to very average-case hard if we slightly increase the complexity of the distribution? Namely, we study the following question: does there exists a language L such that for all distributions E of complexity $g(n)$ the distributional problem (L, E) is easy on the average, but there exist an ensemble of distributions D of complexity $f(n)$ such that the distributional problem (L, D) is hard on the average?

- We consider two complexity measures of distributions:
  1. time complexity for sampling;
  2. time complexity of computing the distribution function.
- We say that a distributional problem (L, D) is easy on the average if there is a linear-time algorithm that for all n gives correct answer on $1 - \alpha(n)$ fraction (according to D) of inputs of length n, where $\alpha(n) = o(1)$. (It seems that we may claim the existence of a polynomial-time algorithm instead of linear-time, but it turns out that if there is

an example with a polynomial-time algorithm, then there is also an example with a linear-time algorithm).

- We consider two variants of the notion that (L, D) is hard on the average:
  1. strong hardness: every algorithm for infinitely many n give a correct answer on at most $\beta(n)$ fraction (according to D) of inputs of length n, where $\beta(n) = o(1)$; (It seems that we may claim this condition only for polynomial-time algorithms but it turns out that if there is an example that is hard for polynomial-time algorithms, then there is also an example that is hard for all algorithms.)
  2. weak hardness: every algorithm for infinitely many n give a correct answer on at most $1 - \beta(n)$ fraction (according to D) of inputs of length n. In this case it is reasonable to assume that $\alpha(n) = o(\beta(n))$.
- It is desirable for $f(n)$ to be not much larger than $g(n)$. In tight results $f(n)$ would be at most polynomial in $g(n)$; in other results $f(n)$ is bounded by a quasipolynomial in $g(n)$.

1.1 Our results

Samplable distributions. The most interesting complexity measure of distributions is the complexity of sampling. In Section 3.1 we consider the statement with the strong notion of hardness. We show that in this case the affirmative answer to our question is *equivalent* to the following *hierarchy for sampling distributions*: there exists a distribution D that is samplable in $f(n)$ steps such that for every distribution F that is samplable in $g(n)$ steps, the statistical distance between D and F is at least $1 - o(1)$.

Watson [9] recently proved the similar (but weaker for our goals) theorem:

▶ **Theorem** ([9]). *For all $a > 0$, $\epsilon > 0$ and $k \in \mathbb{N}$ there exists an ensemble $D \in$ **PSamp** such that:*

- *for all n the distribution D_n is concentrated on $\{1, 2, \ldots, k\}$;*
- *for every ensemble of distributions $F \in \mathbf{Samp}[n^a]$ there exist infinitely many n such that statistical distance between D_n and F_n is at least $1 - \frac{1}{k} - \epsilon$.*

Watson's theorem is not sufficient for our goals since we need the statistical distance $1 - o(1)$ while the theorem gives the statistical distance $1 - \delta$ for all constants $\delta > 0$. We stress that from the equivalence result the tending of the statistical distance to 1 is necessary in order to get an example of a language that is easy according to easy distributions and hard for some more complicated distribution. The proof of Watson's theorem is based on the tree-like diagonalization; we explain (see details in the end of Section 3.1) why the tree-like diagonalization can not be used to get a statistical distance $1 - o(1)$ for polynomial f and g. The statistical distance in Watson's theorem is optimal for distributions concentrated on k values. Thus to get a statistical distance $1 - o(1)$ the function k should be increasing and thus the diagonalization tree should have outgoing degree at least $k(n)$ and this condition makes the diagonalization tree too large and it is impossible to layout it. We show that it is possible to layout the tree in the case when f and g differ quasipolynomially. We prove the hierarchy for sampling distributions for $f(n) = n^{\log^b n}$ and $g(n) = n^{\log^a n}$ for all $0 < a < b$. Our proof uses the proof strategy that is similar to Watson's theorem but our proof is significantly simpler and in particular, we do not use list-decodable error-correcting codes for the transmitting information. From the hierarchy for sampling distributions and the equivalence result we get the following theorem.

▶ **Theorem 1.** *For all $\epsilon > 0$ and $c > 0$ there exist a language L and a linear-time algorithm A such that for every polynomial-time samplable ensemble of distributions F and all n,*

$\Pr_{x \leftarrow F_n}[A(x) = L(x)] \geq 1 - \frac{1}{2^{(\log\log\log n)^c}}$ *and there exists* $D \in \mathbf{Samp}[n^{\log^{\epsilon} n}]$ *such that for every algorithm* B *for infinitely many* n*,* $\Pr_{x \leftarrow D_n}[B(x) = L(x)] \leq \frac{1}{2^{(\log\log\log n)^c}}$*.*

Note that although Theorem 1 argues only about deterministic algorithms B, it implies that for any probabilistic algorithm B' with running time bounded by a computable function, $\Pr_{x \leftarrow D_n}[\Pr_{B'}[B'(x) = L(x)] > 1/2] \leq \frac{1}{2^{(\log\log\log n)^c}}$, where the inner probability is taken over the random bits of algorithm B'. It is interesting to compare Theorem 1 with the result of Gutfreund, Shaltiel and Ta-Shma [4]; they proved that for every $\alpha(n) = o(1)$ there is a distribution D that is samplable in quasipolynomial-time such that for every **NP**-complete language L every polynomial-time randomized algorithm fails to compute L with probability at least $\alpha(n)$ for infinitely many n unless $\mathbf{NP} \subseteq \mathbf{BPP}$. In contrast to [4] Theorem 1 is unconditional, uses the strong notion of hardness and additionally states that L is easy for all polynomial-time samplable distributions, on the other hand the distribution from [4] is the same for all **NP**-complete languages and **NP**-complete languages are important while a language from Theorem 1 is an artificial language based on the tricky diagonalization.

In Section 3.2 we consider the weak notion of hardness and $f(n) = poly(g(n))$. Analogously to the strong hardness we show that in this case the affirmative answer to our question is equivalent to the following conjecture:

▶ **Conjecture 2.** *There exist infinitely small functions* $\beta(n)$ *and* $\alpha(n) = o(\beta(n))$ *such that for all integer* $a > 0$ *and* $b > 0$ *there exist an ensemble of distributions* $D \in \mathbf{PSamp}$*, an increasing sequence of integers* l_n *and a sequence of sets* $S_n \subseteq \{0,1\}^{l_n}$ *such that the following holds:* $D(S_n) > \beta(l_n)$ *for all* n*; for all* $F \in \mathbf{Samp}[n^a]$*,* $F(S_n) \leq \alpha(l_n)$ *for infinitely many* n*.*

Nontrivial condition on this condition is that $\alpha(n)$ is infinitely small. For constants α and β ($\alpha < \beta$) the statement follows from Watson's theorem. For infinitely small α the statement is nontrivial even in the case $\alpha(n) = \beta(n)$. We prove the following theorem.

▶ **Theorem 3.** *For all integer* $a > 0$ *and* $b > 0$ *there exist an ensemble of distributions* $D \in \mathbf{PSamp}$*, a sequence of integers* l_n *and a sequence of sets* $S_n \subseteq \{0,1\}^{l_n}$ *such that the following holds:*

- $D(S_n) > \frac{1}{l_n^b}$ *for all* n*;*
- *For all* $F \in \mathbf{Samp}[n^a]$*,* $F(S_n) \leq \frac{1}{l_n^b}$ *for infinitely many* n*.*

Computable distributions. In Section 4 we consider a complexity of a distribution as the complexity of computing the distribution function. In case of computable distributions in contrast to samplable ones it is possible to find an element with small probability using binary search in polynomial time. However there is the following difficulty: it is not clear how to verify efficiently whether an algorithm computes a distributional function or not. This difficulty prevents to construct the universal computable distribution that dominates all other computable distributions while it is possible to do in the samplable case. We overcome this difficulty and prove the following result for the strong hardness and $f(n) = poly(g(n))$:

▶ **Theorem 4.** *For every* $a > 0$ *there exists a language* L *and an ensemble of polynomial-time computable distributions* D *such that:*

- *there exists a linear-time algorithm* A *such that* $\Pr_{x \leftarrow E_n}[A(x) \neq L(x)] = O(2^{-n})$ *for all* E *that are computable in* $O(n^a)$ *steps;*
- *for every algorithm* A *and for all* n*,* $\Pr_{x \leftarrow D_n}[A(x) \neq L(x)] > 1 - \frac{1}{2^{n-1}}$*.*

2 Preliminaries

An ensemble of distributions is a sequence $\{D_n\}_{n=1}^{\infty}$, where D_n is a probability distribution on $\{0,1\}^n$. Sometimes it is convenient to assume that D_n is concentrated on $\{0,1,\dots,2^n-1\}$.

For two distributions A and B on $\{0,1\}^n$ the statistical distance between them is $\Delta(A,B) = \max_{S\subseteq\{0,1\}^n} |\Pr_{x\leftarrow A}[x\in S] - \Pr_{x\leftarrow B}[x\in S]|$.

A distributional problem is a pair (L,D) that consists of the language L and the ensemble of distributions D. Let $\delta : \mathbb{N}\to[0,1]$ be a function. We say that a distributional problem (L,D) is heuristically decidable it time $t(n)$ with error $\delta(n)$ if there exists an algorithm A such that A runs in $O(t(n))$ steps on the inputs of length n and the following holds: $\Pr_{x\leftarrow D_n}[A(x)\neq L(x)] \le \delta(n)$ for all n. We denote it as $(L,D)\in \mathrm{Heur}_{\delta(n)}\mathbf{DTime}[t(n)]$. We also define a class of distributional problems $\mathrm{Heur}_{\delta(n)}\mathbf{P} = \bigcup_{c>0}\mathrm{Heur}_{\delta(n)}\mathbf{DTime}[n^c]$.

We also define a class $\mathrm{Heur}_{\delta(n)}\mathbf{R}$ that consists of all distributional problems (L,D) such that there exists an algorithm A such that $\Pr_{x\leftarrow D_n}[A(x)\neq L(x)]\le\delta(n)$ for all n.

We say that an ensemble of distributions D is samplable in time $t(n)$ if there exists a randomized algorithm S that on the input 1^n runs in at most $O(t(n))$ steps and $S(1^n)$ is distributed accordingly D_n. The set of all ensembles that are samplable in time $t(n)$ we denote as $\mathbf{Samp}[t(n)]$. We consider the set $\mathbf{PSamp} = \bigcup_{c>0}\mathbf{Samp}[n^c]$ of all polynomial-time samplable ensembles.

3 Samplable distributions

3.1 Strong hardness

▶ **Definition 5.** We say that time constructible functions f and g satisfy the *hierarchy property of sampling distributions* with parameter $\lambda(n)$ if there exists an ensemble of distributions $D\in\mathbf{Samp}[f(n)]$ such that for every ensemble of distributions $F\in\mathbf{Samp}[g(n)]$, there exist infinitely many numbers n such that the statistical distance between D_n and F_n is at least $1-\lambda(n)$.

▶ **Definition 6.** We say that time constructible functions f and g satisfy the *hierarchy property on complexity of distributional problems* with parameters $\alpha(n)>0$ and $\beta(n)>0$ if there exist a language L and an ensemble of distributions $D\in\mathbf{Samp}[f(n)]$ steps such that:

- $(L,F)\in\mathrm{Heur}_{\alpha(n)}\mathbf{P}$ for all $F\in\mathbf{Samp}[g(n)]$;
- $(L,D)\notin\mathrm{Heur}_{1-\beta(n)}\mathbf{P}$.

We say that f and g satisfy *strong* hierarchy property on complexity of distributional problems if the conditions are formulated as:

- There is a linear-time algorithm A such that for all $F\in\mathbf{Samp}[g(n)]$ $\Pr_{x\leftarrow F_n}[A(x) = L(x)]\ge 1-\alpha(n)$ for all n large enough;
- $(L,D)\notin\mathrm{Heur}_{1-\beta(n)}\mathbf{R}$.

▶ **Lemma 7.** *For every time constructible functions $f(n)$, $h(n)$ and $g(n)\ge n$ if f and h satisfy the hierarchy property on sampling distributions with parameter $\lambda(n)$ and $g(n)\log g(n) = o(h(n))$ then f and g satisfy the strong hierarchy property on complexity of distributional problems with parameters $\alpha(n)$ and $\lambda(n)$ for $\alpha(n)=\omega(\lambda(n))$.*

Proof. Let A_i be an enumeration of all randomized algorithms supplied with an alarm clock that interrupt their executions after $O(g(n))$ steps. We will think about A_i as algorithms that sample distributions; that is the output of $A_i(1^n)$ we interpret as a string from $\{0,1\}^n$ by some fixed way. Let B be an algorithm that samples a distribution as follows: on input

1^n with probability $\frac{1}{2}$ it executes $A_1(1^n)$ (and returns its result), with probability $\frac{1}{2^2}$ it executes $A_2(1^n)$, …, with probability $\frac{1}{2^{n-1}}$ it executes $A_{n-1}(1^n)$ and with probability $\frac{1}{2^{n-1}}$ executes $A_n(1^n)$. Let B define an ensemble of distributions E. It is straightforward that $E \in \mathbf{Samp}[h(n)]$.

Since f and h satisfy the hierarchy property of sampling distributions, there exists an ensemble $D \in \mathbf{Samp}[f(n)]$ such that $\Delta(D_n, E_n) \geq 1 - \lambda(n)$ for infinitely many numbers n. We denote the set of all such n as $I = \{n_1, n_2, \dots\}$. For $n \in I$ there exists a set $S_n \subseteq \{0,1\}^n$ such that $D_n(S_n) - E_n(S_n) \geq 1 - \lambda(n)$, hence $E_n(S_n) \leq \lambda(n)$.

We will define a language L such that $L \subseteq \bigcup_{n \in I} S_n$. Let T_i be an enumeration of all algorithms. We define L such that for every $x \in S_{n_k}$, $x \in L$ if and only if T_k does not stop on the input x or rejects it. By the construction $(L, D) \notin \mathrm{Heur}_{1-\lambda(n)}\mathbf{R}$.

We consider an algorithm that returns 0 on every input. If $R \in \mathbf{Samp}[g(n)]$, then there exists i such that A_i samples R. For $n \geq i$ for every set $S \subseteq \{0,1\}^n$ the following inequality holds: $E(S) \geq 2^{-i}R(S)$. Hence for every ensemble R from $\mathbf{Samp}[g(n)]$ this algorithm has error at most $c\lambda(n)$, where c is a constant that depends only on the ensemble R; $c\lambda(n) < \alpha(n)$ for n large enough. ◀

We also prove the opposite implication.

▶ **Lemma 8.** *If f and g satisfy the hierarchy property of complexity of distributional problems with parameters $\alpha(n)$ and $\beta(n)$ then f and g satisfy the sampling hierarchy property with parameter $\alpha + \beta$.*

Proof. For all $F \in \mathbf{Samp}[g(n)]$ there exists a polynomial time algorithm A that solves (L, F) in $\mathrm{Heur}_{\alpha(n)}\mathbf{P}$ and also $(L, D) \notin \mathrm{Heur}_{1-\beta(n)}\mathbf{P}$. Let S_n be set of all $x \in \{0,1\}^n$ such that $A(x) = L(x)$. We know that $F_n(S_n) \geq 1 - \alpha(n)$ for all n and $D_n(S_n) \leq \beta(n)$ for for infinitely many n. Hence $\Delta(D_n, F_n) \geq F_n(S_n) - D_n(S_n) \geq 1 - \alpha(n) - \beta(n)$ for infinitely many n. ◀

Lemma 7 and Lemma 8 implies that if f and g satisfy the hierarchy property of the complexity of distributional problems with two infinitely small parameters then f and $g/\log^2 g$ satisfy the strong hierarchy property on the complexity of distributional problems with two infinitely small parameters. As we mentioned Watson [9] proved that for every $a > 0$, $\epsilon > 0$ and every constant k there exists $b > 0$ such that n^a and n^b satisfy the hierarchy property on sampling distributions with parameter $\frac{1}{k} + \epsilon$. In fact Watson proved the stronger statement since ensemble D is concentrated on k inputs. Watson conjectured that for every $a > 0$ there exists infinitely small function $\alpha(n)$ there exists $b > 0$ such that n^a and n^b satisfy the hierarchy property on sampling distributions with parameter $\alpha(n)$. This statement is still an open question. We prove the following theorem:

▶ **Theorem 9.** *For every a, b, c such that $0 < a < b$ and $c > 0$ functions $f(n) = n^{\log^b n}$ and $g(n) = n^{\log^a n}$ satisfies the sampling hierarchy property with the parameter $\lambda(n) = \frac{1}{2^{(\log\log\log n)^c}}$.*

▶ **Corollary 10.** *For every a, b, c such that $0 < a < b$ and $c > 0$ functions $f(n) = n^{\log^b n}$ and $g(n) = n^{\log^a n}$ satisfies the strong hierarchy property on complexity of distributional problems with parameters $\alpha(n) = \beta(n) = \frac{1}{2^{(\log\log\log n)^c}}$.*

Note that Theorem 1 stated in the introduction follows from Corollary 10.

Before giving a formal proof of Theorem 9 we present an idea of the proof.

In the following, we assume that random variables and elements of ensembles of distributions take values from the set $\{0, 1, \dots, 2^n - 1\}$ instead of $\{0,1\}^n$.

Our proof like a proof of the Watson's theorem is based on the tree-like diagonalization. We construct a distribution D and diagonalize over all distributions samplable in $O(g(n))$

steps by the enumeration of their generators A_i. For the i-th distribution we will prove that the statistical distance between D and $A_i(1^n)$ is large for some n from $[n_i, n_i^*]$, where n_i^* is significantly more than n_i. For every i we construct a tree T_i with vertices uniquely marked with numbers from $[n_i, n_i^*]$. The root of T_i is marked by n_i^* and leaves of T_i are marked with numbers that are about n_i. The number of a parent is greater than the number of a child also the number of a parent is bounded by a quasipolynomial in numbers of its child. Let t be an element from $\{0, 1, \dots, 2^{n_i} - 1\}$ such that in all leaves A_i-probability of t is less then $\lambda(m_i)$, where m_i is the maximum leaf. Such t exists since there are not too many leaves, the possible values of distributions is at least 2^{n_i} and for every distribution the number of elements with probability at least $\lambda(n)$ is at most $\frac{1}{\lambda(n)}$. The distribution $D_{n_i^*}$ is concentrated on t. We assume that for all $n \in [n_i; n_i^*]$ the statistical distance between $A_i(1^n)$ and D_n is less than $1 - \lambda(n)$. Our goal is to define D in such a way that in at least one leaf D is concentrated on t. This will contradict our assumption and the definition of t.

We will transmit information about t from a parent to at least one of its children. The distribution D on the children of p has the following property: if D_p is concentrated (with probability $1 - \epsilon$) on some element, then D_n is concentrated on the same element for at least one child n of p. From the assumption about statistical distances we have that $\Pr[A_i(1^p) = t] \geq \lambda(p) - \epsilon$, hence there are at most $\frac{2}{\lambda}$ candidates on the role of t if we have an access to $A_i(1^p)$. We generate a list of all elements with $A_i(1^p)$-probability at least $\lambda(p) - \epsilon$. In the first child of p we make D concentrated on the first element of the list, on the second child on the second element and so on. There is a problem that there are possibly different lists will be generated in different children; we solve this problem by using several thresholds for frequencies. Formally we do it in the following lemma:

▶ **Lemma 11.** *There is an algorithm $C^{\bullet}(n, i, \delta, \lambda)$ that has an oracle access to some random variable γ taking values in $\{0, 1, \dots, 2^n - 1\}$ such that for all positive integer n and $\delta, \lambda \in (0, 1]$ if $\Pr[\gamma = t] \geq \lambda$ for some t, then there is some integer $0 \leq i \leq \lceil 1 + \frac{1}{\lambda} \rceil^2$ such that $\Pr[C^{\gamma}(n, i, \delta, \lambda) = t] \geq 1 - \delta$ and $C^{\bullet}$ runs at most $\mathrm{poly}(n, \log \frac{1}{\delta}, \frac{1}{\lambda})$ steps.*

Proof. Consider the following algorithm $C^{\gamma}(n, i, \delta, \lambda)$:

1. Let $k = \lceil \frac{1}{\lambda} + 1 \rceil$ and $\epsilon = \frac{\lambda^3}{10k}$;
2. We interpret i as a pair (a, b), where $a, b \in [k]$;
3. Request the oracle for $N = \lceil \frac{2(n+1+\log \frac{1}{\delta})}{\epsilon^2} \rceil$ samples of γ;
4. Consider the list $y_1, \dots, y_s$ of all elements with frequency at least $\lambda - \epsilon a$;
5. Return y_b if $b \leq s$ or 0 otherwise.

Note that for $\lambda \in (0, 1]$

$$k(\lambda - \epsilon(2k)) \geq (\frac{1}{\lambda} + 1)(\lambda - \lambda^3/5) = 1 + \lambda - \lambda^2/5 - \lambda^3/5 > 1. \tag{1}$$

Hence the number of elements x such that $\Pr[\gamma = x] > \lambda - \epsilon k$ is less than k; by the similar reasons $s < k$, where s the size of the list in the 4-th step of the algorithm C.

Consider intervals $I_j = [\lambda - \epsilon j - \epsilon/2; \lambda - \epsilon j + \epsilon/2]$. There is $a \in [k]$ such that $\Pr[\gamma = x] \notin I_a$ for all x since otherwise $1 = \sum_x \Pr[\gamma = x] \geq k(\lambda - \epsilon k - \epsilon/2)$ that contradicts inequality (1). Hence there is $a \in [k]$ such that $|\Pr[\gamma = x] - \lambda - \epsilon a| > \epsilon/2$ for all x.

Let $x_1, \dots, x_l$ be the list of all elements x such that $\Pr[\gamma = x] > \lambda - \epsilon a$. We know that if $\Pr[\gamma = x] > \lambda - \epsilon a$, then $\Pr[\gamma = x] > \lambda - \epsilon a + \epsilon/2$ and also if $\Pr[\gamma = x] \leq \lambda - \epsilon a$, then $\Pr[\gamma = x] < \lambda - \epsilon a - \epsilon/2$. For given a for every $j \in [l]$, x_j appears in the list from 4th step of algorithm C with probability at least $1 - 2e^{-\epsilon^2 N/2}$. If $\Pr[\gamma = x] \leq \lambda - \epsilon a$ then by Chernoff bound x does not appear in the list from the 4th step of the algorithm C with probability at least $1-2e^{-\epsilon^2 N/2}$. Since γ is concentrated on the set of size 2^n with probability at least $1-2^{n+1}e^{-\epsilon^2 N/2} \geq 1-\delta$ the list generated on 4th step of algorithm C is precisely the list $x_1, \dots, x_l$. Since $\Pr[D=t] > \lambda$, there is b such that $x_b = t$. Hence if $i=(a,b)$ then $\Pr[C^\gamma(n,i,\delta,\lambda)=t] \geq 1-\delta$. ◀

Proof of Theorem 9. Our proof is based on the tree-like delayed diagonalization. We diagonalize against all randomized algorithms supplied with a $O(g(n))$-alarm clock, we interpret them as samplers of distributions. Let $A_1, A_2, \dots$ be an enumeration of all randomized algorithms supplied with a $O(g(n))$-alarm clock.

Let us consider an $\epsilon > 0$ such that $(1+a)(1+\epsilon) < (1+b)$ and fix some c. We define integer sequences n_i and n_i^* such that $n_1 = 1$, $n_i^* = 2^{(\log n_i)^{(1+\epsilon)^{d_i}}}$, where $d_i = \lceil \log_{1+\epsilon} 2 \rceil \lceil (\log\log n_i)^2 \rceil$ and $n_{i+1} = n_i^* + 1$. For every i we define an ensemble of distributions D_n for $n \in \{n_i, n_i+1, \dots, n_i^*\}$ such that there exists $k \in \{n_i, n_i+1, \dots, n_i^*\}$ such that $\Delta(D_k, A_i(1^k)) \geq 1-\lambda(k)$.

▶ **Lemma 12.** *For every $\epsilon > 0$ there exists a family of trees T_i such that:*

1. *The set of vertices of T_i is a subset of $\{n_i, n_i+1, \dots, n_i^*\}$;*
2. *n_i^* is the root of T_i;*
3. *All leaves of T_i have numbers at most $m_i = 2n_i$;*
4. *The depth of T_i is $d_i = \lceil \log_{1+\epsilon} 2 \rceil \lceil (\log\log n_i)^2 \rceil$;*
5. *If p is a parent of n then $p \leq 2^{\log^{1+\epsilon} n}$;*
6. *There is an algorithm that for any vertex n of T_i outputs the parent p of n and the number of children of p that are less than n in poly(n) steps;*
7. *For every inner vertex v of T_i, v has $k = \lceil \frac{1}{\lambda(n_i^*)} + 1 \rceil^2$ children.*

Proof. Let us denote $\delta = \lceil \log_{1+\epsilon} 2 \rceil$. We define a tree T_i as a complete balanced tree with depth d_i. The number of leaves in the tree can be estimated as follows: $k^{d_i} \leq (2^{(\log\log\log n_i^*)^{3c}})^{\delta(\log\log n_i)^2} \leq (2^{(\log\log n_i)^{12c}})^{\delta(\log\log n_i)^2} = 2^{\delta(\log\log n_i)^{24c}} \leq n_i$.

The root n_i^* is the only vertex on the zero level. There are exactly k^s vertices on s-th level. Let $a_{i,j} = 2^{(\log n_i)^{(1+\epsilon)^j}}$, where $j \in \{0,1,2,\dots\}$. Vertices of T_i on level $(d_i - s)$ are $[a_{i,s}; a_{i,s} + k^{d_i - s} - 1]$.

Note that $a_{i,s+1} - a_{i,s} \geq a_{i,1} - a_{i,0} = 2^{(\log n_i)^{(1+\epsilon)}} - n_i \geq 2^{(\log n_i)^{(1+\epsilon)}-1} > n_i \geq k^{d_i} \geq k^{d_i - s}$. Hence on all levels there is enough place for vertices.

The parent of j-th vertex on s-th level has number $\lfloor \frac{j}{k} \rfloor$. Let $h(n) = n^{\log^\epsilon n}$. Since $h(n+k) \geq h(n) + k$ we have $h(2^{\log^{(1+\epsilon)^s} n_i} + j) \geq h(2^{\log^{(1+\epsilon)^s} n_i}) + j \geq 2^{\log^{(1+\epsilon)^{s+1}} n_i} + j/k$, therefore the property 5 is satisfied. The verification of other properties is straightforward. ◀

Now we describe an algorithm that samples D_n for $n \in \{n_i, \dots, n_i^*\}$ in $O(f(n))$ steps.

1. If $n = n_i^*$ then output the minimal $t_i \in \{0,1,\dots,2^{n_i}-1\}$ such that for all $l \in [n_i; m_i]$ we have that $\Pr[A_i(1^l) = t_i] < \lambda(n_i)/2$. Such t_i indeed exists since for every l there are at most $\frac{2}{\lambda(n_i)}$ elements with $A_i(1^l)$-probability at least $\lambda(n_i)/2$ and $\frac{2}{\lambda(n_i)} m_i \leq 2^{n_i}$. Such t_i can be found in at most $m_i c_i g(m_i) 2^{c_i g(m_i)}$ steps by brute force search over all possible random bits, where c_i is a constant that depends on i.

$$m_i c_i g(m_i) 2^{c_i g(m_i)} \leq 2^{m_i g(m_i)} \leq 2^{2n_i g(2n_i)} \leq 2^{2n_i (2n_i)^{2\log^a n_i}} < \\ 2^{2^{4\log^{(1+a)} n_i}} \leq 2^{2^{2^{4(1+a)\log\log n_i}}} \leq 2^{2^{2^{(\log\log n_i)^2}}} < n_i^* = o(f(n_i^*))$$

2. If n is not a vertex of T_i then return 0.

3. Otherwise, let p be the parent of n and j is a number of n in the list of all children of p. By the property of T_i, $p \leq 2^{\log^{1+\epsilon} n}$ and such p can be found in $poly(n)$ steps. We return $C^{A_i(1^p)}(p, j, \lambda(n)/2, \lambda(p)/2)$, where C is the algorithm from Lemma 11. By Lemma 11 C runs at most $\mathrm{poly}(p)$ steps and on every step the simulation of $A_i(1^p)$ occupies at most $c_i g(p)$ steps. Note that $c_i g(p)\mathrm{poly}(p) < 2^{2\log^{a+1} p} < 2^{2\log^{1+a}(2^{\log^{1+\epsilon} n})} = 2^{2\log^{(1+a)(1+\epsilon)} n} < 2^{\log^{(1+b)} n} = f(n)$.

For the sake of contradiction we assume that for all $n \in \{n_i, \ldots, n_i^*\}$, $\Delta(D_n, A_i(1^n)) < 1 - \lambda(n)$. By induction on the level s of T_i we prove that there is a vertex v of level s in T_i such that $D_v(t_i) \geq 1 - \lambda(v)/2$. If $D_v(t_i) \geq 1 - \lambda(v)$ for some leaf v then $\Pr[A_i(1^v) = t_i] \geq (1 - \lambda(v)/2) - (1 - \lambda(v)) = \lambda(v)/2$ but we define t_i such that $\Pr[A_i(1^v) = t_i] < \lambda(v)/2$. Hence we will get a contradiction in leaves.

The base of induction follows from the construction of $D_{n_i^*}$. Let us prove the inductive step from s to $s+1$. Let v be a vertex of level s such that $D_v(t_i) \geq 1 - \lambda(v)/2$. If v is a leaf then we are done. Otherwise $\Pr[A_i(1^v) = t_i] > \lambda(v)/2$ since $\Delta(D_v, A_i(1^v)) < 1 - \lambda(v)$. Hence by Lemma 11 there is a child u with number j among the all children of v such that $\Pr[C^{A_i(1^v)}(v, j, \lambda(u)/2, \lambda(v)/2) = t_i] > 1 - \lambda(u)/2$. ◀

Our proof in contrast to Watson's proof does not use error correcting codes with list decoding. This is because we find one element that has a small probability for all leaves of the tree. This trick was impossible in Watson's case since all distributions were concentrated on a constant number of points. In Watson's proof, there were a lot of information transmitted from the root to leaves, and parts of this information were stored in different vertices. Watson used list error decoding codes in order to prevent information distortion.

Now we show why this approach cannot be adapted to the case of $g(n) = n^a$ and polynomial $f(n)$. The problem is the following: for nonconstant $\lambda(n)$ the tree T_i should have nonconstant degree: every inner vertex has at least k_i children, where k_i goes to infinity. In the root of the tree, we have to make exponential in any leaf number of steps; and the parent of every node n should be at most polynomial of every children. Thus for every leaf l the distance between root and l is at least $\Omega(\log \ell)$. Let m_i be the leaf with the maximal number; then the distance from the root to m_i is at least $L = \Omega(\log m_i)$. Let S be the set of vertices such that their numbers are less then m_i but the numbers of their parents are more then m_i. Note that all vertices on the distance L from the root must either be in S or have a descendent in S. Therefore the size of S should be at least k_i^L that is greater then m_i for large i, since k_i goes to infinity. But this is a contradiction since S is set of vertices with numbers less then m_i.

3.2 Weak hardness

In this section we consider statement of the problem with the weak notion of hardness and tight hierarchy ($f(n) = poly(g(n))$). We start from equivalent formulations:

▶ **Proposition 13.** *The following statements are equivalent:*

1. *There exists infinitely small functions $\beta(n)$ and $\alpha(n) = o(\beta(n))$ such that for all $a > 0$ there exists an ensemble of distributions $D \in$ **PSamp** and a language L such that the following holds:*
   - $(L, F) \in \mathrm{Heur}_{\alpha(n)}\mathbf{P}$ *for all* $F \in \mathbf{Samp}[n^a]$*;*
   - $(L, D) \notin \mathrm{Heur}_{\beta(n)}\mathbf{P}$*.*

2. *There exists infinitely small functions $\beta(n)$ and $\alpha(n) = o(\beta(n))$ such that for all $a > 0$ there exist an ensemble of distributions $D \in$ **PSamp**, an increasing sequence of integers l_n and a sequence of sets $S_n \subseteq \{0,1\}^{l_n}$ such that the following holds:*
   - *$D(S_n) > \beta(l_n)$ for all n;*
   - *For all $F \in \mathbf{Samp}[n^a]$, $F(S_n) \leq \alpha(n)$ for infinitely many n.*
3. *There exists infinitely small functions $\beta(n)$ and $\alpha(n) = o(\beta(n))$ such that for all $a > 0$ there exists an ensemble of distributions $D \in$ **PSamp** and a language L such that the following holds:*
   - *there exists linear-time algorithm A such that for all $F \in \mathbf{Samp}[n^a]$, $\Pr_{x \leftarrow F_n}[L(x) = A(x)] \geq 1 - \alpha(n)$ for all n large enough;*
   - *$(L, D) \notin \mathrm{Heur}_{\beta(n)}\mathbf{R}$.*

We prove the statement that is weaker than statement 2 from Proposition 13. Namely we prove it in the case $\alpha(n) = \beta(n) = \frac{1}{n^b}$. By the similar way it is possible to prove it for other infinitely small functions: $\frac{1}{2^n}$, $\frac{1}{\log n}$ etc.

Now we are ready for proving Theorem 3. We start from the intuition of the proof. For simplicity we start from the proof of the other statement with threshold $\frac{1}{2}$ instead of $\frac{1}{l_n^b}$. We use the delayed diagonalization; we consider integer sequences n_i and n_i^* such that $n_1 = 1$, $n_i = n_i^* + 1$, $n_i^* = 2^{n_i^a}$. Let F_i be enumeration of all randomized algorithms with alarm clock n^{a+1} such that every algorithm appears infinitely many times in this enumeration; we consider F_i as samplers of distributions.

We denote by $T_{0,n}$ and $T_{1,n}$ the set of binary strings of length n starting with 0 and 1 respectively. Consider the following sampler of the distribution D_n: if $n = n_i^*$, we find $t_i \in \{0,1\}$ such that $\Pr[F_i(1^{n_i}) \in T_{t_i,n_i}] \leq \frac{1}{2}$ and return the random element from T_{t_i,n_i^*}; if $n_i \leq n < n_i^*$ we execute $F_i(1^{n+1})$ if it returns a string starting with $s \in \{0,1\}$ we return a random element from $T_{s,n}$.

Assume that there exists $E \in \mathbf{Samp}[n^a]$ such that for all $n > n_0$ if for some $S \subseteq \{0,1\}^n$, $\Pr[D_n \in S] > \frac{1}{2}$, then $\Pr[F_i(1^n) \in S] > \frac{1}{2}$. Let F_i is a sampler for E and $i > n_0$. We know that $\Pr[D_{n_i^*} \in T_{t_i,n_i^*}] = 1$, then $\Pr[F_i(1^{n_i^*}) \in T_{t_i,n_i^*}] > \frac{1}{2}$, thus $\Pr[D_{n_i^*-1} \in T_{t_i,n_i^*-1}] > \frac{1}{2}$ and so on. Finally we get $\Pr[F_i(1^{n_i}) \in T_{t_i,n_i}] > \frac{1}{2}$ and this contradicts the definition of t_i.

For threshold $\frac{1}{k}$ the proof will be the same but we split $\{0,1\}^n$ into $\log k$ parts. In case of threshold $\frac{1}{n^b}$ we will have a different number of parts for different n, and we will use trees of intervals instead of chains.

Proof of Theorem 3. Consider an enumeration of all randomized algorithms F_i with alarm clock n^{a+1} such that every algorithm appears infinitely many times in this enumeration; we consider F_i as samplers of distributions. We define integer sequences n_i and n_i^* such that $n_1 = 1$, $n_i^* = 2^{n_i^a}$, and $n_{i+1} = 2n_i^*$.

Split all strings of length n on n^b nonempty sets; we call them intervals and denote by $T_{j,n}$ for $j \in \{1, 2, \dots, n^b\}$. For $n \in [n_i; n_i^*]$ we define a graph (it will be a forest) as follows:

- The set of vertexes of the graph is the set of all intervals $T_{j,n}$ for $n = 2^k$ and $n_i \leq n \leq n_i^*$;
- All elements of T_{j,n_i} are roots of trees of the forest;
- For $n \in \{n_i, 2n_i, 4n_i, \dots, n_i^*/2\}$, $T_{j,n}$ has 2^b children: $\{T_{j',2n} \mid 2^b(j-1) \leq j' \leq 2^b j - 1\}$.
- All elements of T_{j,n_i^*} are leaves of trees of the forest;

We define a sampler for D as follows. It gets on the input 1^n:

- If $n = n_i^*$ for some i, then find an interval T_{j,n_i} with the smallest probability according to $F_i(1^{n_i})$. If there are several such T_{j,n_i}, we take one with the minimal j. (Note that this can be done in $poly(n_i^*)$ time by brute-force). Then chose random descendent of T_{j,n_i} on length n_i^* and return some string form this descendent. Note that $\Pr[F_i(1^{n_i}) \in T_{j,n_i}] \leq \frac{1}{n_i^b}$;

- If $n_i \le n < n_i^*$ for some i, then run $F_i(1^{2n})$ and if the result belongs to a descendent of $T_{j,n}$ for some j, then return random string from $T_{j,n}$.

Let us prove that for all i there exists j and $n \in [n_i; n_i^*]$ such that $\Pr[D_n \in T_{j,n}] > \frac{1}{n^b}$ and $\Pr[F_i(1^n) \in T_{j,n}] \le \frac{1}{n^b}$. (This will conclude the proof of the theorem if we choose $S_i = T_{j,n}$.) Assume the opposite; that is for all j and $n \le n_i^*$ if $\Pr[F_i(1^n) \in T_{j,n}] \le \frac{1}{n^b}$, then $\Pr[D_n \in T_{j,n}] < \frac{1}{n^b}$. Let T_{j,n_i} be an interval with the smallest probability according to $F_i(1^{n_i})$, hence $\Pr[F_i(1^{n_i}) \in T_{j,n_i}] \le \frac{1}{n_i^b}$. By induction on l we prove that for all $n = 2^l n_i$ (and $n \le n_i^*$) there exists k such that $T_{k,n}$ is a descendant of T_{j,n_i} and $\Pr[D_n \in T_{k,n}] \le \frac{1}{n^b}$. The base case $l = 0$ is already proved. Let us prove the inductive step from l to $l+1$. Let $n = 2^l n_i$. Assume that $\Pr[D_n \in T_{k,n}] \le \frac{1}{n^b}$ then by the pigeonhole principle and construction of D there is one of 2^b children of $T_{k',2n}$ such that $\Pr[F_i(1^{2n}) \in T_{k',2n}] \le \frac{1}{(2n)^b}$ and hence by assumption $\Pr[D_{2n} \in T_{k',2n}] \le \frac{1}{(2n)^b}$. Therefore there exists k such that $\Pr[D_{n_i^*} \in T_{k,n_i^*}] \le \frac{1}{(n_i^*)^b}$ and T_{k,n_i^*} is a descendant of T_{j,n_i}, but the construction of D implies that the D-probability of every descendant of T_{j,n_i} on length n_i^* is equal to $\frac{n_i^b}{(n_i^*)^b} > \frac{1}{(n_i^*)^b}$. ◀

▶ **Corollary 14.** *For all $a > 0$ and $b > 0$ there exists a ensemble of distributions $D \in$ **PSamp**, a language L and a linear-time algorithm A such that the following holds: $\Pr_{x \leftarrow F_n}[A(x) \ne L(x)] = O(\frac{1}{n^b})$ for all $F \in$ **Samp**$[n^a]$; $(L, D) \notin \mathrm{Heur}_{\frac{1}{n^b}}$**R**.*

4 Computable distributions

Ensemble of distributions D_n is computable in time $t(n)$ if for all n probabilities of all elements according to D_n are binary rational numbers and there exists an algorithm $A(x)$ that runs in $O(t(|x|))$ steps and computes the cumulative distribution function of D_n (i.e. $\sum_{y \le x} D_n(x)$, where $\le$ is lexicographical order). The set of all ensembles that are computable in time $t(n)$ we denote as **Comp**$[t(n)]$. The set **PComp** $= \bigcup_{c>0}$ **Comp**$[n^c]$ is the set of all ensembles computable in polynomial time.

▶ **Lemma 15.** *If an ensemble $D \in$ **PSamp** and for all n the distribution D_n is concentrated on one element, then $D \in$ **PComp**.*

It is possible to prove the statement that is analogous to hierarchy property of n^a and n^b of sampling distributions but for computable distributions.

▶ **Proposition 16.** *For all $a > 0$ there exists an ensemble $D \in$ **PComp** such that for all ensembles $F \in$ **Comp**$[n^a]$ there are infinitely many numbers n such that $\Delta(D_n, F_n) \ge 1 - 2^{-n}$.*

Now we prove Theorem 4 that is similar to hierarchy property of n^a and n^b on the complexity of distributional problems but for computable distributions.

Proof. We cannot literally repeat the proof of Lemma 7 regardless of we have even already proved Proposition 16. The reason is the following: not every algorithm computes the distributional function, it is not necessary that it computes even monotonic function. And it is not easy to verify that algorithms compute a distribution function.

Let A_i be an enumeration of all algorithms supplied with alarm-clock Cn^a, where C is some constant. We interpret them as algorithms that compute distribution functions. However, we remember that it is not necessary that all of them computes a correct distribution function. We interpret the result of $A_i(x)$ as a binary real number between 0 and 1.

For every n we will show that it is possible in $\mathrm{poly}(n)$ time to find $x_n \in \{0,1\}^n$ such that if $i \in \{1, 2, \ldots, n\}$ and A_i is distributional function, then the A_i-probability of x_n is at most

2^{i-n}. The distribution D_n would be concentrated on x_n; the resulting ensemble is computable in polynomial time by Lemma 15. If for all n we find such x_n, then we may define L similarly to the proof of Lemma 7. Namely we will choose L such that $L \subseteq \bigcup_n \{x_n\}$ and $x_n \in L$ if and only if n-th algorithm in the enumeration of all algorithms rejects x_n. For all $F \in \mathbf{Comp}[n^a]$ the algorithm that returns 0 on all inputs decides (L, F) in $\mathrm{Heur}_{2^{i-n}}\mathbf{DTime}[n]$, if F is computable by A_i in our enumeration. By the construction $(L, D) \notin \mathrm{Heur}_{1-\frac{1}{2^{n-1}}}\mathbf{P}$.

Now we describe the procedure of finding strings x_n. Initially $I = \{1, 2, \dots, n\}$, we will delete element i from I if we discover that A_i is not a distribution function on $\{0,1\}^n$. On each iteration we define $F(x) = \sum_{i \in I} \frac{1}{2^i} A_i(x)$. By binary search we try to find such element $x \in \{0,1\}^n$ that $F(x) - F(x') \leq 2^{-n}$, where x' is lexicographical predecessor of x and $F(x') = 0$ if $x = 0^n$. If binary search succeeds, then $x_n := x$. If binary search fails then it means that we discover nonmonotonicity of $F(x)$, using this we may find $i \in I$ such that A_i is nonmonotonic and exclude all such i from I and start new iteration. If $I = \emptyset$ then choose $x_n = 0^n$, in other cases for all $i \in I$ if A_i computes a correct distribution function then x_n has probability at most 2^{i-n}. ◀

Acknowledgements. The authors are grateful to anonymous reviewers for useful comments.

References

1 László Babai, Paul Erdős, and Stanley Selkow. Random graph isomorphism. *SIAM J. Comput.*, 9(3):628–635, 1980.

2 Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992. doi:10.1016/0022-0000(92)90019-F.

3 Yuri Gurevich and Saharon Shelah. Expected computation time for hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

4 Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. If NP languages are hard on the worst-case, then it is easy to find their hard instances. *Computational Complexity*, 16(4):412–441, 2007. doi:10.1007/s00037-007-0235-8.

5 Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Heuristic time hierarchies via hierarchies for sampling distributions. In *Algorithms and Computation – 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 201–211, 2015.

6 Ming Li and Paul M.B. Vitanyi. Average case complexity under the universal distribution equals worst case complexity. *Information Processing Letters*, 42:145–149, 1992.

7 Konstantin Pervyshev. On heuristic time hierarchies. In *IEEE Conference on Computational Complexity*, pages 347–358, 2007. doi:10.1109/CCC.2007.20.

8 E. R. van Dama and M. Muzychuk. Some implications on amorphic association schemes. *Journal of Combinatorial Theory, Series A*, 117:111–127, 2010.

9 Thomas Watson. Time hierarchies for sampling distributions. *SIAM J. Comput.*, 43(5):1709–1727, 2014. doi:10.1137/120898553.

Computing the Pattern Waiting Time: A Revisit of the Intuitive Approach[*]

Kai Jin

Department of Computer Science, University of Hong Kong, Hong Kong
`cscjjk@gmail.com`

Abstract

We revisit the waiting time of patterns in repeated independent experiments. We show that the most intuitive approach for computing the waiting time, which reduces it to computing the stopping time of a Markov chain, is optimum from the perspective of computational complexity. For the single pattern case, this approach requires us to solve a system of m linear equations, where m denotes the length of the pattern. We show that this system can be solved in $O(m+n)$ time, where n denotes the number of possible outcomes of each single experiment. The main procedure only costs $O(m)$ time, while a preprocessing procedure costs $O(m+n)$ time. For the multiple pattern case, our approach is as efficient as the one given by Li [14].

Our method has several advantages over other methods. First, it extends to compute the variance or even higher moment of the waiting time for the single pattern case. Second, it is more intuitive and does not entail tedious mathematics and heavy probability theory. Our main result (Theorem 2) might be of independent interest to the theory of linear equations.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Pattern matching, G.3 [Probability and Statistics] Markov processes, G.1.3 [Numerical Linear Algebra] Linear systems

Keywords and phrases Pattern Occurrence, Waiting Time, Penney's Game, Markov Chain

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.39

1 Introduction

In this paper, we revisit the waiting time of patterns in repeated independent experiments. Consider an experiment with countably many *outcomes*, denoted by $1, \ldots, n$, where the probabilities of the outcomes might be nonidentical. Let the experiment be performed repeatedly. Given a collection of t finite sequences of possible outcomes, called *patterns*, we compute the expected waiting time till one of the pattern is observed in a run of experiments. We also compute the probability for each pattern to be the first to appear.

These quantities have been extensively studied in history and different approaches were invented for computing them. The existing approaches apply complicated mathematical tools, e.g. the generating functions applied in [12] and [4], the martingale theory applied in [14], and the renewal theory applied in [4]. In this paper we revisit a much more intuitive approach for computing them and we show that this approach is computationally optimal.

For the single pattern case, we use the following approach to compute the expected waiting time. We reduce it to compute the expected stopping time of a Markov chain (see its construction in Subsection 1.2), which consists of m nonterminal states, where m is

[*] This work is done at Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University. Supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, and the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 39; pp. 39:1–39:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the length of the given pattern. The expected stopping time can easily be computed in polynomial time by standard methods, since it reduces to solve a system of linear equations with m equations and m variables. We propose a novel method to solve this system and thus compute the expected stopping time starting from any state of the Markov chain, which only costs $O(m+n)$ time and thus is computationally optimal. This time complexity cannot be guaranteed by standard linear system solving; for achieving it we exploit structural properties of the Markov chain and some insights about pattern matching.

Our method easily extends to compute the variance or even higher moments of the waiting time for a single pattern, starting from any state of the Markov chain. The running time is still $O(m+n)$ for the variances, and is $O(m \cdot k^2 + n)$ for the k-th moments.

For the multiple pattern case, computing the expected waiting time and the probability for each pattern to be first to appear reduces to solving a system of linear equations with $t+1$ variables and $t+1$ equations, in which the coefficients can be computed by our algorithm for the single pattern case. Our algorithm has the same time complexity as the one given in [14].

1.1 Related work and applications

The waiting time of patterns is a classic problem in probability theory which arises in 1960s ([17, 23]). Since then it has drawn a lot of attentions and has been studied extensively. It has many applications in different fields, including computer science, telecommunication, molecular biology, statics and applied probability; see [22, 11, 16, 15] and the references within.

Due to the practical importance of the waiting time, several approaches were invented for studying it. Guibas and Odlyzko [12] studied it via a combinatorial approach. They computed the generating function of the number of strings with any fixed length which contain none of the given patterns. Breen, Waterman and Zhang [4] obtained similar results via a probabilistic approach, in which the renewal theory of Feller [7] is applied. Li [14] studied it via martingale theory and general Markov chain theory (See also [19]). Most approaches entail tedious mathematics and heavy probability theory, even for the single pattern case. Useful reviews of different approaches can be found in two recent books: [3, 8].

To compute the waiting time, there is an intuitive approach as aforementioned. It appears in several literatures and is usually called Markov Embedding approach. However, to the best of our knowledge, researches have not recognized that this approach is optimum from the perspective of computational complexity. (Usually, the stopping time of a Markov chain cannot be computed in linear time).

Many variants of the original problem have been studied. For example, Fudos, Pitoura and Szpankowski [9] studied the probability of exactly r occurrences of a pattern in a fixed number of Bernoulli experiments (independent identical experiments). In addition, people also considered the waiting time in Markovian experiments (Markovian dependent experiments) instead of Bernoulli experiments; see [5, 18, 2] and the references within.

A closely related topic to the waiting time is the frequency of patterns. Régnier and coauthors made important contributions to this topic. Let O_k denote the number of occurrence of a given pattern in k Markovian experiments when overlapping is counted separately. [21] computed the mean and variance of O_k. This result was extended to the higher order Markovian experiments, for non-overlap or overlap counting, in [20].

The literature of the problem concerning the probability for each pattern to be the first to appear starts from [17]. This problem is referred to as Penney's Game ([1]). Conway gave a beautiful formula for the 2-players Penney's Game (See [10]). The best strategy is known for choosing the pattern so that the winning probability is optimal; see [10, 12, 6].

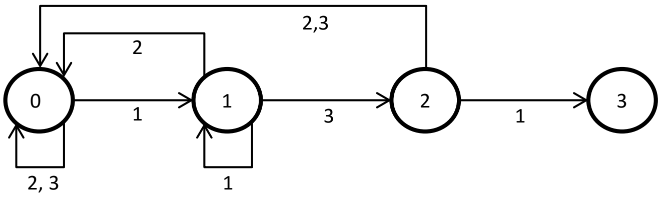


Figure 1 The Markov chain corresponding to the above example.

1.2 Technique overview

Now, we define the aforementioned Markov chain corresponding to a single pattern, state our main result and present the overview of our techniques.

The Markov chain corresponding to pattern W

Denote $m = |W|$ and assume $W = W_1W_2..W_m$. Denote $W^{(i)} = W[1..i]$, which is the prefix of W with length i. Recall that there are n possible outcomes and assume that the probability of outcome i is $Pr(i)$. Note that $Pr(1), \dots, Pr(n)$ are fixed parameters.

The Markov chain consists of $m + 1$ states, state 0 to state m, each of which corresponds to a prefix of W. State m is the *terminal state*. For each state $i(0 \leq i \leq m - 1)$, there are n outgoing edges whose ending points are denoted by $\tau(i, 1), \dots, \tau(i, n)$, respectively. Specifically, if it's now at state i and the next outcome is j, the next state should be

$$\tau(i, j) = \max\{h \mid W^{(h)} \text{ is a suffix of } (W^{(i)} + \text{`}j\text{'})\}. \tag{1}$$

The j-th outgoing edge of each state has an associated probability $Pr(j)$.

Note: The symbol '+' in Equation 1 indicates concatenation of strings.

▶ **Example 1.** Let $W =$ "131". $n = 3$. The corresponding Markov chain is drawn in Figure 1.

Clearly, computing the expected waiting time of pattern W reduces to computing the expected stopping time of the preceding Markov chain.

Our Main Results

Suppose $(c_0, \dots, c_{m-1})$ are m constants. We consider the following system of linear equations.

$$x_i = \begin{cases} 0, & i = m; \\ c_i + \sum_{j=1}^{n} Pr(j) \cdot x_{\tau(i,j)}, & i = 0..m-1. \end{cases} \tag{2}$$

Note that there are $m + 1$ variables in this system, which are $x_0, \dots, x_m$, and there are also $m + 1$ equations. Our main result is the following.

▶ **Theorem 2.** *The solution of system (2) can be computed in $O(m + n)$ time.*

Denote e_i as the expected "stopping time" starting from state i; and denote f_i as the expected "square of stopping time" starting from state i. Formally,

$$e_i \quad := \quad \sum_{t \geq 0} t \cdot Pr(\text{It takes } t \text{ steps to get the terminal state from state } i); \tag{3}$$

$$f_i \quad = \quad \sum_{t \geq 0} t^2 \cdot Pr(\text{It takes } t \text{ steps to get the terminal state from state } i). \tag{4}$$

The following equations are easy to prove.

$$e_i = \begin{cases} 0, & i = m; \\ \left(\sum_{j=1}^{n} Pr(j) \cdot e_{\tau(i,j)}\right) + 1, & i = 0..m-1. \end{cases} \tag{5}$$

$$f_i = \begin{cases} 0, & i = m; \\ \left(\sum_{j=1}^{n} Pr(j) \cdot f_{\tau(i,j)}\right) + (2e_i - 1), & i = 0..m-1. \end{cases} \tag{6}$$

Proof of (5). Assume $i < m$; otherwise it is trivial. By First-Step-Analysis,

$$\begin{aligned} e_i &= \sum\nolimits_{j=1}^{n} Pr(j) \sum\nolimits_{t \geq 0} (t+1) \cdot Pr(\text{It takes } t \text{ steps to get state } m \text{ from state } \tau(i,j)) \\ &= \sum\nolimits_{j=1}^{n} Pr(j) \cdot \left(e_{\tau(i,j)} + 1\right) \\ &= \left(\sum\nolimits_{j=1}^{n} Pr(j) \cdot e_{\tau(i,j)}\right) + 1. \end{aligned}$$

◀

Proof of (6). Assume $i < m$; otherwise it is trivial. By First-Step-Analysis,

$$\begin{aligned} f_i &= \sum\nolimits_{j=1}^{n} Pr(j) \sum\nolimits_{t \geq 0} (t+1)^2 \cdot Pr(\text{It takes } t \text{ steps to get state } m \text{ from state } \tau(i,j)) \\ &= \sum\nolimits_{j=1}^{n} Pr(j) \cdot \left(f_{\tau(i,j)} + 2e_{\tau(i,j)} + 1\right) \\ &= \left(\sum\nolimits_{j=1}^{n} Pr(j) \cdot f_{\tau(i,j)}\right) + 2(e_i - 1) + 1. \end{aligned}$$

◀

According to Theorem 2, we can compute array $e = (e_0, \ldots, e_m)$ in $O(m+n)$ time. Moreover, after array e has been computed, we can further compute array $f = (f_0, \ldots, f_m)$ in $O(m+n)$ time. The expected stopping time is e_0, and its variance is $f_0 - e_0^2$.

▶ **Corollary 3.** *Given a pattern W with length m, the expected waiting time of W and the variance of the waiting time of W can be computed in $O(m+n)$ time.*

The above method for computing the expectation and variance can be easily generalized to compute the higher moments. We show this generalization in Section 5.

Technique overview – solve system (2)

System (2) has m unknowns $x_0, \ldots, x_{m-1}$ to compute. Rather than compute them directly, we define $y_i = x_0 - x_i$ and compute the unknowns $y_1, \ldots, y_m$. The difficulty on computing y_i lies in computing the term $\sum_{j=1..n, j \neq W_{i+1}} Pr(j) \cdot y_{\tau(i,j)}$, which is denoted by z_i.

Let $\pi : \{0, .., m\} \to \{-1, .., m-1\}$ be the well-known prefix function of W. Formally,

$$\pi_i = \begin{cases} -1, & i = 0; \\ \max\left\{h \mid h < i \text{ and } W^{(h)} \text{ is a suffix of } W^{(i)}\right\}, & i > 0. \end{cases} \tag{7}$$

Our key observation is that z_i can be computed from z_{π_i} in $O(1)$ time. More specifically,

$$z_i = z_{\pi_i} + [y_{\pi_i+1} \cdot Pr(W_{\pi_i+1})] - [y_{\sigma_i} \cdot Pr(W_{i+1})] \quad (\text{for } 1 \leq i < m),$$

where $\sigma_i = \tau(\pi_i, W_{i+1})$.

Based on this observation, we compute y and z in $O(m)$ time. Besides, we spend $O(n+m)$ time to compute the auxiliary array σ. Our method is computationally optimum, since inputting the task instance would already cost $O(n+m)$ time.

Technique overview – the multiple pattern case

For the multiple pattern case, we only consider the expectation of waiting time and the probability for each pattern to be first to appear; we do not consider variance of even higher moments of the waiting time. Our method is similar to the one of Li [14]. Specifically, we shall solve the same system of linear equations (which consists of $t+1$ variables) proposed by Li. However, we use a different approach to compute the coefficients of this system.

Outline. We prove Theorem 2 in Section 2. We present an alternative linear algorithm for computing $e_0,\ldots,e_{m-1}$ in Section 3. We consider the multiple pattern case in Section 4. We compute the higher moments of the waiting time of a single pattern in Section 5.

2 Solving system (2) in linear time

▶ **Definition 4.** For $0 \le i \le m$, we define a *periods set*

$$\mathcal{P}(i) = \begin{cases} \{0\}, & i = 0; \\ \mathcal{P}(\pi_i) \cup \{i\}, & i > 0. \end{cases}$$

The following lemmas can be found in any textbook on finite automata and is the basis of the famous KMP (Knuth-Morris-Pratt) algorithm ([13]). We omit their trivial proofs.

▶ **Lemma 5.** *Suppose $0 \le i \le m$ and $0 \le h \le i$. Then,*

$h \in \mathcal{P}(i)$ if and only if $W^{(h)}$ is a suffix of $W^{(i)}$.

▶ **Lemma 6.** *For $1 \le i < m$ and $1 \le j \le n$, we have*

$$\tau(i,j) = \begin{cases} i+1, & j = W_{i+1}; \\ \tau(\pi_i, j), & j \ne W_{i+1}. \end{cases} \tag{8}$$

In this section, we prove Theorem 2, which states that the system of linear equations stated in (2) can be solved in linear time.

We do not compute x directly; instead, we compute another array $y = (y_0,\ldots,y_m)$, which are defined as follows. To compute y efficiently, we introduce two more arrays $z = (z_0,\ldots,z_{m-1})$ and $\sigma = (\sigma_1,\ldots,\sigma_{m-1})$ as follows.

$$y_i = x_0 - x_i \quad (0 \le i \le m) \tag{9}$$

$$z_i = \sum_{j=1..n, j \ne W_{i+1}} Pr(j) \cdot y_{\tau(i,j)} \quad (0 \le i < m) \tag{10}$$

$$\sigma_i = \tau(\pi_i, W_{i+1}) \quad (\forall 1 \le i < m) \tag{11}$$

We state two formulas. (Their proofs are deferred for a moment.)

$$y_i = (c_{i-1} + y_{i-1} - z_{i-1})/Pr(W_i) \quad (\text{for } 1 \le i \le m)\,. \tag{12}$$

$$z_i = z_{\pi_i} + [y_{\pi_i+1} \cdot Pr(W_{\pi_i+1})] - [y_{\sigma_i} \cdot Pr(W_{i+1})] \quad (\text{for } 1 \le i < m) \tag{13}$$

We give the algorithm in Algorithm 1.

Note that when we compute z_i in Line 5, we have $\pi_i < i, \pi + 1 \le i$ and $\sigma_i \le i$. These inequalities respectively imply that the quantities z_{π_i}, y_{π_i+1}, and y_{σ_i} have been computed before we compute z_i. Therefore, Algorithm 1 is correct according to (12) and (13).

Algorithm 1 Algorithm for computing x

```
1: Compute $\pi$ and $\sigma$;                       ▷ See the detailed algorithm in the next subsection.
2: $y_0, z_0 \leftarrow 0$;
3: **for** $i = 1..m-1$ **do**
4:     $y_i \leftarrow (c_{i-1} + y_{i-1} - z_{i-1})/Pr(W_i)$.                          ▷ Applying (12).
5:     $z_i \leftarrow z_{\pi_i} + y_{\pi_i+1} \cdot Pr(W_{\pi_i+1}) - y_{\sigma_i} \cdot Pr(W_{i+1})$        ▷ Applying (13).
6: **end for**
7: $y_m \leftarrow (c_{m-1} + y_{m-1} - z_{m-1})/Pr(W_m)$;                             ▷ Applying (12).
8: $x_0 \leftarrow y_m$.                          ▷ Because $y_m = x_0 - x_m = x_0 - 0 = x_0$.
9: Compute $x_1, \dots, x_{m-1}$ from $y$ by formula $x_i = x_0 - y_i$.
```

Proof of (12). Assume $1 \le i \le m$. Then,

$$\begin{aligned}
y_{i-1} + c_{i-1} &= x_0 - x_{i-1} + c_{i-1} && \text{(due to (9))}\\
&= x_0 - \sum_{j=1..n} Pr(j) \cdot x_{\tau(i-1,j)} && \text{(due to (2))}\\
&= (\sum_{j=1..n} Pr(j)) \cdot x_0 - \sum_{j=1..n} Pr(j) \cdot x_{\tau(i-1,j)} && (\text{since } \sum_{j=1}^{n} Pr(j) = 1)\\
&= \sum_{j=1..n} Pr(j) \cdot (x_0 - x_{\tau(i-1,j)})\\
&= \sum_{j=1..n} Pr(j) \cdot y_{\tau(i-1,j)} && \text{(due to (9))}\\
&= \sum_{j=1..n, j \neq W_i} Pr(j) \cdot y_{\tau(i-1,j)} + Pr(W_i) \cdot y_{\tau(i-1,W_i)}\\
&= z_{i-1} + Pr(W_i) \cdot y_{\tau(i-1,W_i)} && \text{(due to (10))}\\
&= z_{i-1} + Pr(W_i) \cdot y_i. && (\text{since } \tau(i-1, W_i) = i)
\end{aligned}$$

This further implies (12). ◀

Proof of (13). Assume $1 \le i < m$. We have

$$z_i = \sum_{1 \le j \le n, j \neq W_{i+1}} Pr(j) \cdot y_{\tau(i,j)} = \sum_{1 \le j \le n, j \neq W_{i+1}} Pr(j) \cdot y_{\tau(\pi_i,j)}$$

The first equation follows from the definition (10), whereas the second follows from (8).

On the other side, by the definition (10) of z_{π_i}, we have

$$z_{\pi_i} = \sum_{1 \le j \le n, j \neq W_{\pi_i+1}} Pr(j) \cdot y_{\tau(\pi_i,j)} \,.$$

Therefore,

$$z_i - z_{\pi_i} = Pr(W_{\pi_i+1}) \cdot y_{\tau(\pi_i, W_{\pi_i+1}))} - Pr(W_{i+1}) \cdot y_{\tau(\pi_i, W_{i+1})}.$$

Substituting $\tau(\pi_i, W_{i+1})$ and $\tau(\pi_i, W_{\pi_i+1}))$ by σ_i and $\pi_i + 1$ respectively, we obtain (13). ◀

▶ Remark. Proving Formula 13 is the key step in designing our linear time algorithm. This formula shows that we can rapidly compute z_i from z_{π_i}. The proof of this formula mainly applies the properties of τ stated in Lemma 6.

In a more straight-forward way, we may apply Formula 10 instead of Formula 13 to compute z_i at Line 5. This would produce an alternative algorithm that runs in $O(m^2)$ time.

2.1 Preprocessing procedure – computing π and σ in linear time

The prefix function π can be computed in $O(m)$ time by using the famous KMP (Knuth-Morris-Pratt) algorithm, see [13]. Next, we assume that π is computed and we present the algorithm for computing $\sigma = (\sigma_1, \ldots, \sigma_{m-1})$. Recall that $\sigma_i = \tau(\pi_i, W_{i+1})$.

We describe the algorithm in Algorithm 2. It uses a Depth-First-Search (DFS).

Algorithm 2 Compute σ by a DFS

```
Tr[1..$n$] is a temporary array.
procedure ComputeSigma(i)                ▷ When entering this procedure, Tr[$*$] = $\tau(i,*)$
    for $j : \pi(j) = i$ do
        $\sigma_j \leftarrow$ Tr[$W_{j+1}$];
        Tr[$W_{j+1}$] $\leftarrow j + 1$;                        ▷ Here, Tr[$*$] becomes $\tau(j,*)$
        ComputeSigma(j);
        Tr[$W_{j+1}$] $\leftarrow \sigma_j$;                      ▷ Let Tr[$*$] return to $\tau(i,*)$
    end for
end procedure
Tr[1], ..., $Tr[n] \leftarrow 0$;
Tr[$W_1$] $\leftarrow 1$;                      ▷ The last two lines make Tr[$*$] = $\tau(0,*)$
ComputeSigma(0)
```

The correctness of Algorithm 2 is obvious. When entering the procedure ComputeSigma with parameter i, the temporary array Tr$[1, \ldots, n]$ stores $\tau(i,1), \ldots, \tau(i,n)$.

This preprocessing procedure runs in $O(n+m)$ time.

▶ Remark. There is a chance that this preprocessing procedure may be improved to $O(m)$ time. However, for two reasons the $O(n+m)$ time solution is just fine. First, inputting the task instance would cost $O(n+m)$ time. Second, usually we can assume that $n \le m$. (If $n > m$, we may modify the input parameters and merge the redundant outcomes at first.)

3 An alternative method followed by Li's approach

In this section, we restate some beautiful results proved by Shuo-yen Robert Li in [14]. Then, we show that based on these results, we can design an alternative algorithm which computes $e_0, \ldots, e_{m-1}$ in $O(m)$ time. We then compare this algorithm to Algorithm 1.

▶ **Definition 7.** Let $A = A_1A_2 \ldots A_s$ and $W = W_1W_2 \ldots W_m$ be two strings over $\{1..n\}$. For every pair (i,j) of integers, write

$$\delta_{ij} = \begin{cases} Pr(W_j)^{-1}, & \text{if } 1 \le i \le s, 1 \le j \le m \text{ and } A_i = W_j; \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Then define

$$A * W = \delta_{11}\delta_{22} \ldots \delta_{ss} + \delta_{21}\delta_{32} \ldots \delta_{s,s-1} + \ldots + \delta_{s1} \tag{15}$$

▶ **Example 8.** Let $A =$ "2113", $W =$ "131". Assume that $Pr(1) = \frac{1}{2}, Pr(2) = \frac{1}{3}, Pr(3) = \frac{1}{6}$. Then $W * W = 2 \cdot 6 \cdot 2 + 0 \cdot 0 + 2 = 26$, and $A * W = 0 \cdot 0 \cdot 2 \cdot 0 + 2 \cdot 0 \cdot 0 + 2 \cdot 6 + 0 = 12$.

▶ **Lemma 9** ([14]). *Given a starting string A, the expected waiting time for a pattern W is*

$$W * W - A * W,$$

provided that W is not a substring of A. In particular, the waiting time of pattern W (without a starting sequence) is $W * W$.

By Lemma 9, we have the following equation for $e_0, \ldots, e_{m-1}$.

$$e_i = W * W - W^{(i)} * W. \tag{16}$$

In the following, we prove the following result.

▶ **Theorem 10.** *We can compute the values of* $W^{(0)} * W = 0, W^{(1)} * W, \ldots, W^{(m)} * W = W * W$ *altogether in* $O(m)$ *time, and thus compute* $e_0, \ldots, e_{m-1}$ *in* $O(m)$ *time.*

To prove this result, we first state the following equation of $W^{(i)} * W$.

Recall the periods set $\mathcal{P}(i)$ defined in Definition 4. Denote $\mathsf{prod}_h = \prod_{j=1}^{h} Pr(W_j)^{-1}$. Then,

$$W^{(i)} * W = \sum_{h \in \mathcal{P}(i)} \mathsf{prod}_h. \tag{17}$$

The proof of Equation 17 is deferred for a moment.

Furthermore, recall that

$$\mathcal{P}(i) = \begin{cases} \{0\}, & i = 0; \\ \mathcal{P}(\pi_i) \cup \{i\}, & i > 0. \end{cases}$$

We obtain the following equation based on (17).

$$W^{(i)} * W = \begin{cases} 0, & i = 0; \\ W^{(\pi_i)} * W + \mathsf{prod}_i, & i > 0. \end{cases} \tag{18}$$

According to this recursive equation, we obtain Theorem 10 immediately. The detailed algorithm is omitted. In the following we prove Equation 17.

Proof of (17). Set $A = W^{(i)}$. According to the definition of $A * W$ in (15),

$$\begin{aligned} W^{(i)} * W &= A * W \\ &= \delta_{11}\delta_{22} \ldots \delta_{ii} + \delta_{21}\delta_{32} \ldots \delta_{i,i-1} + \ldots + \delta_{i1} \\ &= [W^{(i)} \text{ is a suffix of } W^{(i)}] \cdot Pr(W_1)^{-1} \ldots Pr(W_i)^{-1} + \\ &\quad [W^{(i-1)} \text{ is a suffix of } W^{(i)}] \cdot Pr(W_1)^{-1} \ldots Pr(W_{i-1})^{-1} + \ldots \\ &\quad + [W^{(1)} \text{ is a suffix of } W^{(i)}] \cdot Pr(W_1)^{-1} \\ &= \sum_{1 \le h \le i} [W^{(h)} \text{ is a suffix of } W^{(i)}] \cdot \mathsf{prod}_i \\ &= \sum_{h \in \mathcal{P}(i)} \mathsf{prod}_i \end{aligned}$$

The last step is due to Lemma 5. ◀

▶ Remark. In designing the above linear algorithm for computing $W^{(0)} * W, \ldots, W^{(m)} * W$, the key is to apply the recursive formula (18). This formula tells us that we can compute $W^{(i)} * W$ rapidly from $W^{(\pi(i))} * W$. We note that the same technique is applied in the previous algorithm for computing $z_1, \ldots, z_{m-1}$ shown in Section 2.

Lemma 9 provides a concise and beautiful formula, which reveals many insights of the problem. However, its proof requires some advanced mathematic tools; for example, the Doob's fundamental theorem on stopping times of martingales. In addition, it only computes the expected waiting time, while our approach easily extends to compute the variance.

4 The multiple pattern case

In this section, we consider the multiple pattern case. Given a set of t patterns $W_1, \ldots, W_t$. Suppose that they are reduced, which means that no pattern is a substring of another. We compute the expected waiting time till one of the pattern is observed in a run of experiments. We also compute the probability for each pattern to be first to appear.

Our method is similar to the one given by Li [14].

▶ **Lemma 11.** *For $1 \le i \le t$, let N_i be the random variable which indicate the waiting time till W_i is observed, and let p_i be the probability that W_i precedes the remaining $t-1$ patterns and be the first pattern to appear. Let $N = min(N_1, \ldots, N_t)$. Let $e_{i,k}$ denote the expected stopping time of the Markov chain corresponding to W_i, starting from its state k. For every i, j such that $1 \le i, j \le t$, we denote by $k(i,j)$ the largest k such that $W_i^{(k)}$ is a suffix of W_j. Then*

$$e_{i,0} = EN + \sum_{j=1}^{t} p_j e_{i,k(i,j)} \quad (1 \le i \le t) \tag{19}$$

Proof.

$$\begin{aligned} e_{i,0} &= E(N_i) \\ &= EN + E(N_i - N) \\ &= EN + \sum_{j=1}^{t} E(N_i - N \mid N = N_j)) \cdot Pr(N = N_j) \\ &= EN + \sum_{j=1}^{t} p_j e_{i,k(i,j)} \end{aligned}$$ ◀

Note that the value of e can be computed by our algorithm for the simple pattern case. Also note that we have another simple equation as follows.

$$\sum_{j=1}^{t} p_j = 1. \tag{20}$$

In the matrix form, the t equations in (19) and the equation in (20) become:

$$\begin{pmatrix} 0 & 1 & \ldots & 1 \\ 1 & e_{1,k(1,1)} & \ldots & e_{1,k(1,t)} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & e_{t,k(t,1)} & \ldots & e_{t,k(t,t)} \end{pmatrix} \begin{pmatrix} EN \\ p_1 \\ \ldots \\ p_t \end{pmatrix} = \begin{pmatrix} 1 \\ e_{1,0} \\ \ldots \\ e_{t,0} \end{pmatrix}. \tag{21}$$

Let M denote the coefficient matrix of this system. Li [14] proved that M is nonsingular. So, we can compute $EN, p_1, \ldots, p_t$ by solving this system of linear equations.

5 Higher moments of the waiting time of a single pattern

For any integer $k \ge 0$ and for any $0 \le i \le m$, denote

$$g_{k,i} := \sum_{t \ge 0} t^k \cdot Pr(\text{It takes } t \text{ steps to get the terminal state from state } i). \tag{22}$$

In Subsection 1.2, we defined array $e = g_1$ and array $f = g_2$ and we show that both of them can be computed efficiently by applying Theorem 2. In the following we show that arrays $g_1, \ldots, g_k$ can be computed altogether in $O(m \cdot k^2)$ time (after the preprocessing procedure for computing σ). As a corollary, we obtain:

▶ **Corollary 12.** *Given a pattern W with length m, the first k moments of the waiting time of W can be computed in $O(n + m \cdot k^2)$ time.*

We state an equation of g_k at first. For $k \geq 1$,

$$g_{k,i} = \begin{cases} 0, & i = m; \\ \left(\sum_{j=1}^{n} Pr(j) \cdot g_{k,\tau(i,j)}\right) + \sum_{h=0}^{k-1} \binom{k}{h}(-1)^{k-1-h} g_{h,i}, & i = 0..m-1. \end{cases} \tag{23}$$

▶ **Example 13.** For $i < m$, we have

$$\begin{aligned}
g_{1,i} &= \left(\sum\nolimits_{j=1}^{n} Pr(j) \cdot g_{1,\tau(i,j)}\right) + g_{0,i}; \\
g_{2,i} &= \left(\sum\nolimits_{j=1}^{n} Pr(j) \cdot g_{2,\tau(i,j)}\right) + 2g_{1,i} - g_{0,i}; \\
g_{3,i} &= \left(\sum\nolimits_{j=1}^{n} Pr(j) \cdot g_{3,\tau(i,j)}\right) + 3g_{2,i} - 3g_{1,i} + g_{0,i}; \\
g_{4,i} &= \left(\sum\nolimits_{j=1}^{n} Pr(j) \cdot g_{4,\tau(i,j)}\right) + 4g_{3,i} - 6g_{2,i} + 4g_{1,i} - g_{0,i}; \\
&\cdots
\end{aligned}$$

According to (23), we can apply Theorem 2 to compute g_k after we have computed $g_0, \ldots, g_{k-1}$. Therefore, we obtain the aforementioned results and Corollary 12.

We apply the following identity to prove (23).

$$\sum_{h=l}^{k} \binom{k}{h}\binom{h}{l}(-1)^h = 0 \qquad \text{(when } k > l \geq 0\text{)} \tag{24}$$

Proof of (24).

$$\sum_{h=l}^{k} \binom{k}{h}\binom{h}{l}(-1)^h = \sum_{h=l}^{k} \binom{k}{l}\binom{k-l}{h-l}(-1)^h = \binom{k}{l}\sum_{h=l}^{k}\binom{k-l}{h-l}(-1)^h = 0$$ ◀

Proof of (23). Assume $i < m$; otherwise it is trivial. By First-Step-Analysis,

$$\begin{aligned}
g_{k,i} &= \sum\nolimits_{j=1}^{n} Pr(j)\sum\nolimits_{t\geq 0}(t+1)^k \cdot Pr(\text{It takes } t \text{ steps to get state } m \text{ from state } \tau(i,j)) \\
&= \sum\nolimits_{j=1}^{n} Pr(j)\sum\nolimits_{t\geq 0} \cdot \left(t^k + \sum\nolimits_{h=0}^{k-1}\binom{k}{h} t^h\right) \cdot Pr(\text{It takes } t \text{ steps ... from state } \tau(i,j)) \\
&= \left(\sum\nolimits_{j=1}^{n} Pr(j)\cdot g_{k,\tau(i,j)}\right) + \sum\nolimits_{h=0}^{k-1}\binom{k}{h}\sum\nolimits_{j=1}^{n} Pr(j) g_{h,\tau(i,j)} \\
&= \left(\sum\nolimits_{j=1}^{n} Pr(j)\cdot g_{k,\tau(i,j)}\right) + \sum\nolimits_{h=0}^{k-1}\binom{k}{h}\left(g_{h,i} - \sum\nolimits_{l=0}^{h-1}\binom{h}{l}(-1)^{h-1-l} g_{l,i}\right) \\
&= \left(\sum\nolimits_{j=1}^{n} Pr(j)\cdot g_{k,\tau(i,j)}\right) - \sum\nolimits_{h=0}^{k-1}\binom{k}{h}\sum\nolimits_{l=0}^{h}\binom{h}{l}(-1)^{h-1-l} g_{l,i} \\
&= \left(\sum\nolimits_{j=1}^{n} Pr(j)\cdot g_{k,\tau(i,j)}\right) - \sum\nolimits_{l=0}^{k-1} g_{l,i}\cdot \sum\nolimits_{h=l}^{k-1}\left(\binom{k}{h}\binom{h}{l}(-1)^{h-1-l}\right) \\
&= \left(\sum\nolimits_{j=1}^{n} Pr(j)\cdot g_{k,\tau(i,j)}\right) - \sum\nolimits_{l=0}^{k-1} g_{l,i}\cdot \left(\binom{k}{l}(-1)^{k-l}\right) \\
&= \left(\sum\nolimits_{j=1}^{n} Pr(j)\cdot g_{k,\tau(i,j)}\right) + \sum\nolimits_{h=0}^{k-1} g_{h,i}\cdot \left(\binom{k}{h}(-1)^{k-1-h}\right)
\end{aligned}$$

Note that the second last step applies Identity 24. ◀

Acknowledgements. The author would like to thank Zhiyi Huang for taking part in fruitful discussions, thank Jian Li and Andrew Yao and for much help, and thank several anonymous reviewers from MFCS16 and ISAAC16 for much great suggestions.

References

1 Anonymous. Penney's game. Technical report, Wikipedia, 2016. URL: `https://en.wikipedia.org/wiki/Penney%27s_game`.

2 J. A. D. Aston and D. E. K. Martin. Waiting time distributions of competing patterns in higher-order markovian sequences. *Journal of Applied Probability*, 42(4):977–988, 2005.

3 N. Balakrishnan and M. V. Koutras. *Runs and Scans with Applications*. Wiley, 2002.

4 S. Breen, M. S. Waterman, and N. Zhang. Renewal theory for several patterns. *Journal of Applied Probability*, 22(1):228–234, 1985.

5 O. Chrysaphinou and S. Papastavridis. The occurrence of sequence patterns in repeated dependent experiments. *Theory of Probability & Its Applications*, 35(1):145–152, 1991. `doi:10.1137/1135015`.

6 J. A. Csirik. Optimal strategy for the first player in the penney ante game. *Combinatorics, Probability and Computing*, 1:311–321, 1992. `doi:10.1017/S0963548300000365`.

7 W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. Wiley, 3 edition, 1968.

8 J. C. Fu and W. Y. W. Lou. *Distribution Theory of Runs and Patterns and Its Applications*. World Scientific Publishing, 2003.

9 I. Fudos, E. Pitoura, and W. Szpankowski. On pattern occurrences in a random text. *Information Processing Letters*, 57(6):307–312, 1996.

10 M. Gardner. On the paradoxical situations that arise from nontransitive relations. *Scientific American*, 231(4), 1974.

11 R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

12 L. J. Guibas and A. M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *Journal of Combinatorial Theory, Series A*, 30(2):183–208, 1981. doi:10.1016/0097-3165(81)90005-4.

13 D. E. Knuth, Jr. J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.

14 S. R. Li. A martingale approach to the study of occurrence of sequence patterns in repeated experiments. *The Annals of Probability*, 8(6):1171–1176, 1980.

15 M. E. Lladser, M. D. Betterton, and R. Knight. Multiple pattern matching: a markov chain approach. *Journal of Mathematical Biology*, 56(1):51–92, 2007. doi:10.1007/s00285-007-0109-3.

16 M. Lothaire. *Applied Combinatorics on Words.* Cambridge University Press, 2005.

17 W. Penney. Problem 95: Penney-ante. *Journal of Recreational Mathematics*, page 241, 1969.

18 V. Pozdnyakov. On occurrence of patterns in markov chains: Method of gambling teams. *Statistics & Probability Letters*, 78(16):2762–2767, 2008. doi:10.1016/j.spl.2008.03.023.

19 V. Pozdnyakov and M. Kulldorff. Waiting times for patterns and a method of gambling teams. *The American Mathematical Monthly*, 113(2):134–143, 2006.

20 M. Régnier. A unified approach to word occurrence probabilities. *Discrete Applied Mathematics*, 104:259–280, 2000.

21 M. Régnier and W. Szpankowski. On pattern frequency occurrences in a markovian sequence. *Algorithmica*, 22(4):631–649, 1998. doi:10.1007/PL00009244.

22 G. Reinert, S. Schbath, and M. S. Waterman. Probabilistic and statistical properties of words: An overview. *Journal of Computational Biology*, 7(2):1–46, 2000.

23 A. D. Solov'ev. A combinatorial identity and its application to the problem concerning the first occurrence of a rare event. *Theory of Probability & Its Applications*, 11(2):276–282, 1966. doi:10.1137/1111022.

$O(f)$ Bi-Approximation for Capacitated Covering with Hard Capacities[*]

Mong-Jen Kao[1], Hai-Lun Tu[2], and D. T. Lee[3]

1 Institute of Information Science, Academia Sinica, Taipei, Taiwan
2 Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
3 Institute of Information Science, Academia Sinica, Taipei, Taiwan; and Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Abstract

We consider capacitated vertex cover with hard capacity constraints (VC-HC) on hypergraphs. In this problem we are given a hypergraph $G = (V, E)$ with a maximum edge size f. Each edge is associated with a demand and each vertex is associated with a weight (cost), a capacity, and an available multiplicity. The objective is to find a minimum-weight vertex multiset such that the demands of the edges can be covered by the capacities of the vertices and the multiplicity of each vertex does not exceed its available multiplicity.

In this paper we present an $O(f)$ bi-approximation for VC-HC that gives a trade-off on the number of augmented multiplicity and the cost of the resulting cover. In particular, we show that, by augmenting the available multiplicity by a factor of $k \geq 2$, a cover with a cost ratio of $\left(1 + \frac{1}{k-1}\right)(f - 1)$ to the optimal cover for the original instance can be obtained. This improves over a previous result, which has a cost ratio of f^2 via augmenting the available multiplicity by a factor of f.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Capacitated Covering, Hard Capacities, Bi-criteria Approximation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.40

1 Introduction

The capacitated vertex cover problem with hard capacities (VC-HC) models a demand-to-service assignment scenario generalized from the classical vertex cover problem. In this problem, we are given a hypergraph $G = (V, E \subseteq 2^V)$ with maximum edge size f, where each $e \in E$ satisfies $|e| \leq f$ and is associated with a demand $d_e \in \mathbb{R}^{\geq 0}$, and each $v \in V$ is associated with a weight (or cost) $w_v \in \mathbb{R}^{\geq 0}$, a capacity $c_v \in \mathbb{R}^{\geq 0}$, and an available multiplicity $m_v \in \mathbb{Z}^{\geq 0}$. The objective is to find a vertex multiset, or, cover, represented by a demand assignment function $h \colon E \times V \to \mathbb{R}^{\geq 0}$, such that the following two constraints are met:

1. $\sum_{v \in e} h_{e,v} \geq d_e$ for all $e \in E$,
2. $x_v^{(h)} \leq m(v)$ for all $v \in V$, where $x_v^{(h)} := \left\lceil \sum_{e \colon e \in E,\, v \in e} h_{e,v}/c_v \right\rceil$,

and $\sum_{v \in V} w(v) \cdot x_h(v)$ is minimized.

[*] This work is supported in part by Ministry of Science and Technology (MOST), Taiwan, under Grant MOST104-2218-E-001-032.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 40; pp. 40:1–40:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider bicriteria approximation for VC-HC with augmented multiplicity constraints. In particular, we say that a demand assignment h forms an augmented (β, γ)-cover if it is feasible for the augmented multiplicity function $m'_v := \beta \cdot m_v$ for all $v \in V$ and the cost ratio is at most γ compared to the optimal assignment for the original instance. In other words, we are allowed to use additional multiplicities of the vertices up to a factor of β.

Background and Prior Work

The capacitated vertex cover generalizes vertex cover in that a demand-to-service assignment model is evolved from the original 0/1 covering model. This transition was exhibited via several work.

For classical vertex cover, it is known that a f-approximation can be obtained by LP rounding and duality [1, 8]. Khot and Regev [13] showed that, assuming the unique game conjecture, approximating this problem to a ratio better than $f - \epsilon$ is NP-hard for any $\epsilon > 0$ and $f \geq 2$.

Chuzhoy and Naor [4] considered VC-HC on simple graphs with unit edge demands, i.e., $|e| = 2$ and $d_e = 1$ for all $e \in E$. They presented a 3-approximation for the unweighted version of this problem, i.e., $w_v = 1$ for all $v \in V$. On the contrary, they showed that the weighted version is at least as hard as set cover, which renders $O(f)$-approximations unlikely to exist even for this simple setting. Due to this reason, subsequent work on VC-HC has focused primarily on the unweighted version.

Gandhi et al. [5] gave a 2-approximation for unweighted VC-HC with unit edge demand by presenting a refined rounding approach to [4]. Saha and Khuller [14] considered general edge demands and presented an $O(f)$-approximation for f-hypergraphs. Cheung et al. [3] presented an improved approach for this problem. They presented a $\left(1 + 2/\sqrt{3}\right)$-approximation for simple graphs and a $2f$-approximation for f-hypergraphs. The gap of approximation for this problem was recently closed by Kao [10], who presented an f-approximation for any $f \geq 2$.

Grandoni et al. [6] considered weighted VC-HC with unit vertex multiplicity, i.e., $m_v = 1$ for all $v \in V$, and augmented multiplicity constraints. They presented a primal-dual approach that yields an augmented $(2, 4)$-cover for simple graphs[1], which further extends to augmented (f, f^2)-cover for f-hypergraphs. This approach does not generalize, however, to arbitrary vertex multiplicities and does not entail further parametric trade-off either.

Further Related Work

The capacitated covering problem has been studied in various forms and variations. When the number of available multiplicities is unlimited, this problem is referred to soft capacitated vertex cover (CVC). This problem was first considered by Guha et al. [7], who gave a 2-approximation based on primal-dual. Kao et al. [11, 12, 9] studied capacitated dominating set problem and presented a series of results for the complexity and approximability of this problem. Bar-Yehuda et al. [2] considered partial CVC and presented a 3-approximation for simple graphs based on local ratio techniques.

Wolsey [15] considered submodular set cover, which includes classical set cover as a special case and which relates to capacitated covering in a simplified form, and presented a $(\ln \max_S f(S) + 1)$-approximation. This approach was generalized by Chuzhoy and Naor [4]

[1] The bicriteria approximation ratio of [6] is updated in the context due to the different considered models. In [6] each vertex is counted at most once in the cost of the cover, disregarding the number of multiplicities it needs. In our model, however, the cost is weighted over the multiplicities of each vertex.

to capacitated set cover with hard capacities and unit demands, for which a $(\ln \delta + 1)$-approximation was presented, where δ is the maximum size of the sets.

Our Result and Approach

We consider VC-HC with general parameters and present bicriteria approximations that yields a trade-off between the number of augmented multiplicities and the resulting cost. Our main result is the following bicriteria approximation algorithm:

▶ **Theorem 1.** *For any integer $k \geq 2$, we can compute an augmented $\Big(k, (1 + \frac{1}{k-1})(f-1)\Big)$-cover for weighted VC-HC in polynomial time.*

This improves over the previous ratio of (f, f^2) in [6] and provides a parameter trade-off on the augmented multiplicity and the quality of the solution. In particular, the cost ratio we obtained for this bi-approximation is bounded within $\frac{3}{2}(f-1)$ for all $k \geq 2$ and converges asymptotically to $f - 1$ as k tends to infinity.

Our algorithm builds on primal-dual charging techniques combined with a flow-based procedure that exploits the duality of the LP relaxation. The primal-dual scheme we present extends the basic framework from [12, 7], which were designed for the soft capacity model where $m_v = \infty$ for all v. In contrast to the previous result in [6], we employ a different way of handling the dual variables as well as the primal demand assignments that follow. The seemingly subtle difference entails dissimilar analysis and gives a guarantee that is unavailable via their approach.

In particular, for the primal demand assignments, we use flow-based arguments to deal with pending decisions. This ensures that the vertices whose multiplicity limits are attained receive sufficient amount of demands to pay for their costs. The crucial observation in establishing the bicriteria approximation factor is that the feasible regions of the dual LP remains unchanged when the multiplicity constraint is augmented. Therefore the cost of the solution obtained via the primal-dual approach can be bounded by the optimal cost of the original instance. Together this gives our bi-approximation result.

The rest of this paper is organized as follows. In §2 we formally define VC-HC and introduce the natural LP relaxation and its dual LP for which we will be working with. For a better flow to present our bicriteria approximation, we first introduce our primal-dual algorithm and the corresponding analysis in §3. In §4 we establish the bi-approximation approximation ratio and prove Theorem 1. Finally we conclude in §5 with some future directions for related problems.

2 Problem Statement and LP Relaxation

Let $G = (V, E)$ denote a hypergraph with vertex set V and edge set $E \subseteq 2^V$ and $f := \max_{e \in E} |e|$ denote the size of the largest hyperedge in G. For any $v \in V$, we use $E[v]$ to denote the set of edges that are incident to the vertex v. Formally, $E[v] := \{e \colon e \in E \text{ such that } v \in e\}$. This definition extends to set of vertices, i.e., for any $A \subseteq V$, i.e., $E[A] := \bigcup_{v \in A} E[v]$.

2.1 Capacitated Vertex Cover with Hard Capacities (VC-HC)

In this problem we are given a hypergraph $G = (V, E \subseteq 2^V)$, where each $e \in E$ is associated with a demand $d_e \in \mathbb{R}^{\geq 0}$ and each $v \in V$ is associated with a weight (or cost) $w_v \in \mathbb{R}^{\geq 0}$, a capacity $c_v \in \mathbb{R}^{\geq 0}$, and its available multiplicities $m_v \in \mathbb{Z}^{\geq 0}$.

By a demand assignment we mean a function $h\colon E \times V \to \mathbb{Z}^{\geq 0}$, where $h_{e,v}$ denotes the amount of demand that is assigned from edge e to vertex v. For any $v \in V$, we use $D_h(v)$ to denote the total amount of demand vertex v has received in h, i.e., $D_h(v) = \sum_{e \in E[v]} h_{e,v}$.

The corresponding multiplicity function, denoted $x^{(h)}$, is defined to be $x_v^{(h)} = \lceil D_h(v)/c_v \rceil$. A demand assignment h is *feasible* if $\sum_{v \in e} h_{e,v} \geq d_e$ for all $e \in E$ and $x_v^{(h)} \leq m_v$ for all $v \in V$. In other words, the demand of each edge is fully-assigned to (fully-served by) its incident vertices and the multiplicity of each vertex does not exceed its available multiplicities. The *weight (cost)* of h, denoted $w(h)$, is defined to be $\sum_{v \in V} w_v \cdot x_v^{(h)}$.

Given an instance $\Pi = (V, E, d_e, w_v, c_v, m_v)$ as described above, the problem of VC-HC is to compute a feasible demand assignment h such that $w(h)$ is minimized. Without loss of generality, we assume that the input graph G admits a feasible demand assignment.[2]

Augmented Cover.

Let $\Pi = (V, E, d_e, w_v, c_v, m_v)$ be an instance for VC-HC. For any integral $\beta \geq 1$, we say that a demand assignment h forms an augmented (β, γ)-cover if

1. $\sum_{v \in e} h_{e,v} \geq d_e$ for all $e \in E$.
2. $x_v^{(h)} \leq \beta \cdot m_v$ for all $v \in V$.
3. $w(h) \leq \gamma \cdot \min_{h' \in \mathcal{F}} w(h')$, where $\mathcal{F}$ is the set of feasible demand assignments for Π.

2.2 LP Relaxation and the Dual LP

Let $\Pi = (V, E, d_e, w_v, c_v, m_v)$ be the input instance of VC-HC. The natural LP relaxation of VC-HC for the instance Π is given below in LP(1). The first three inequalities model the feasibility constraints of a demand assignment and its corresponding multiplicity function. The fourth inequality states that the multiplicity of a vertex cannot be zero if any demand gets assigned to it. This seemingly unnecessary constraint is required in giving a bounded integrality gap for this LP relaxation.

$$
\begin{array}{lll}
\text{Minimize} \quad \sum_{v \in V} w_v \cdot x_v & & (1) \\
\sum_{v \in e} h_{e,v} \geq d_e, & \forall e \in E & \\
c_v \cdot x_v - \sum_{e \in E[v]} h_{e,v} \geq 0, & \forall v \in V & \\
x_v \leq m_v, & \forall v \in V & \\
d_e \cdot x_v - h_{e,v} \geq 0, & \forall e \in E, \ v \in e & \\
x_v, \, h_{e,v} \geq 0, & \forall e \in E, \ v \in e &
\end{array}
$$

The dual LP for the instance Π is given below in LP(2). A solution $\Psi = (y_e, z_v, g_{e,v}, \eta_v)$ to this LP can be interpreted as an extended packing LP as follows: We want to raise the values of y_e for all $e \in E$. However, the value of each y_e is constrained by z_v and $g_{e,v}$ that are further constrained by w_v for each $v \in e$. The variable η_v provides an additional degree

[2] By selecting all of the available multiplicities, the feasibility of G can be checked via a max-flow computation.

of freedom in this packing program in that it allows higher values to be packed into y_e in the cost of a reduction in the objective value. Note that, this exchange does not always yield a better lower-bound for the optimal solution. In this paper we present an extended primal-dual scheme to handle this flexibility.

$$\text{Maximize} \quad \sum_{e \in E} d_e \cdot y_e - \sum_{v \in V} m_v \cdot \eta_v \tag{2}$$

$$\begin{aligned}
& c_v \cdot z_v + \sum_{e \in E[v]} d_e \cdot g_{e,v} - \eta_v \le w_v, && \forall v \in V \\
& y_e \le z_v + g_{e,v}, && \forall v \in V,\; e \in E[v] \\
& y_e,\; z_v\; g_{e,v}\; \eta_v \ge 0, && \forall v \in V,\; e \in E[v]
\end{aligned}$$

For the rest of this paper, we will use OPT(Π) to denote the cost of optimal solution for the instance Π. Since the optimal value of the above LPs gives a lower-bound on OPT(Π) which we will be working with, we also use OPT(Π) to denote their optimal value in the context.

3 A Primal-Dual Schema for VC-HC

In this section we present our extended primal-dual algorithm for VC-HC. The algorithm we present extends the framework developed for the soft capacity model [12, 7]. In the prior framework, the demand is assigned immediately when a vertex from its vicinity gets saturated. In our algorithm, we keep some of decisions pending until we have sufficient capacity for the demands. In contrast to the primal-dual scheme used in [6], which always stores dual values in $g_{e,v}$, we store the dual values in both $g_{e,v}$ and z_v, depending on the amount of unassigned demand v possesses in its vicinity. This ensures that, the cost of each multiplicity is charged only to the demands it serves.

To obtain a solid bound for this approach, however, we need to guarantee that the vertices whose multiplicity limits are attained receive sufficient amount of demands to charge to. This motivates our flow-based procedure Self-Containment for dealing with the pending decisions. During this procedure, a natural demand assignment is also formed.

3.1 The Algorithm

In this section we present our extended primal-dual algorithm Dual-VCHC. This algorithm takes as input an instance $\Pi = (V, E, d, w, c, m)$ of VC-HC and outputs a feasible primal demand assignment h together with a feasible dual solution $\Psi = (y_v, z_v, g_{e,v}, \eta_v)$ for Π.

The algorithm starts with an initial zero dual solution and eventually reaches a locally optimal solution. During the process, the values of the dual variables in Ψ are raised gradually and some inequalities will meet with equality. We say that a vertex v is *saturated* if the inequality $c_v \cdot z_v + \sum_{e \in E[v]} d_e \cdot g_{e,v} - \eta_v \le w_v$ is met with equality.

Let $E^\phi := \{e : e \in E, d_e > 0\}$ be the set of edges with non-zero demand and $V^\phi := \{v : v \in V, m_v \cdot c_v > 0\}$ be the set of vertices with non-zero capacity. For each $v \in V$, we use $d^\phi(v) = \sum_{e \in E[v] \cap E^\phi} d_e$ to denote the total amount of demand in $E[v] \cap E^\phi$. For intuition, E^ϕ contains the set of edges whose demands are not yet processed nor assigned, and V^ϕ corresponds to the set of vertices that have not yet saturated.

In addition, we maintain a set S, initialized to be empty, to denote the set of vertices that have saturated and that have at least one incident edge in E^{ϕ}. Intuitively, S corresponds to vertices with pending assignments.

The algorithm works as follows. Initially all dual variables in Ψ and the demand assignment h are set to be zero. We raise the value of the dual variable y_e for each $e \in E^{\phi}$ simultaneously at the same rate. To maintain the dual feasibility, as we increase y_e, either z_v or $g_{e,v}$ has to be raised for each $v \in e$. If $d^{\phi}(v) \leq c_v$, then we raise $g_{e,v}$. Otherwise, we raise z_v. In addition, for all $v \in e \cap S$, we *raise* η_v to the extent that keeps v saturated.

When a vertex $u \in V^{\phi}$ becomes saturated, it is removed from V^{ϕ}. Then we invoke a recursive procedure Self-Containment($S \cup \{u\}, u$), which we describe in the next paragraph, to compute a pair (S', h'), where

- S' is a maximal subset of $S \cup \{u\}$ whose capacity, if chosen, can fully-serve the demands in $E[S'] \cap E^{\phi}$, and
- h' is the corresponding demand assignment function (from $E[S'] \cap E^{\phi}$ to S').

If $S' = \emptyset$, then we leave the assignment decision pending and add u to S. Otherwise, S' is removed from S and $E[S']$ is removed from E^{ϕ}. In addition, we add the assignment h' to final assignment h to be output. This process repeats until $E^{\phi} = \emptyset$. Then the algorithm outputs h and Ψ and terminates.

We also note that, the particular vertex to saturate in each iteration is the one with the smallest value of $w^{\phi}(v)/\min\{c_v, d^{\phi}(v)\}$, where $w^{\phi}(v) := w_v - \left(c_v \cdot z_v + \sum_{e \in E[v]} d_e \cdot g_{e,v} - \eta_v\right)$ denotes the current slack of the inequality associated with $v \in V^{\phi}$.

The Procedure Self-Containment(A, u)

In the following we describe the recursive procedure Self-Containment(A, u). It takes as input a vertex subset $A \subseteq V$ and a vertex $u \in V$ and outputs a pair $(S', \tilde{h}')$, where S' is a maximal subset of A whose capacity is sufficient to serve the unassigned demands in its vicinity, and h' is the corresponding demand assignment.

First we define a directed flow-graph $\mathcal{G}(A)$ with a source s^+ and a sink s^- for the vertex set A as follows. Excluding the source s^+ and the sink s^-, $\mathcal{G}(A)$ is a bipartite graph induced by $E[A] \cap E^{\phi}$ and A. For each $e \in E[A] \cap E^{\phi}$, we have a vertex $\tilde{e}$ and an edge $(s^+, \tilde{e})$ in $\mathcal{G}$. Similarly, for each $v \in A$ we have a vertex $\tilde{v}$ and an edge $(\tilde{v}, s^-)$. For each $v \in A$ and each $e \in E[v] \cap E^{\phi}$, we have an edge $(\tilde{e}, \tilde{v})$ in $\mathcal{G}$.

The capacity of each edge is defined as follows. For each $e \in E[A] \cap E^{\phi}$, the capacity of $(s^+, \tilde{e})$ is set to be d_e. For each $v \in A$, the capacity of $(\tilde{v}, s^-)$ is set to be $m_v \cdot c_v$. The capacities of the remaining edges are unlimited.

The procedure Self-Containment works as follows. If $u \in A$, then it computes the max-flow $\tilde{h}$ for $\mathcal{G}(A)$ with the additional constraint that $\tilde{h}(\tilde{u}, s^-)$ is minimized among all max-flows for $\mathcal{G}(A)$.[3] If $u \notin A$, then it simply computes any max-flow $\tilde{h}$ for $\mathcal{G}(A)$. Let

$$S' = \left\{v \colon v \in A \text{ such that } \tilde{h}(s^+, \tilde{e}) = d_e \text{ for all } e \in E[v] \cap E^{\phi}\right\}$$

be the subset of A that is able to serve the demand in $E[S'] \cap E^{\phi}$. If $S' = A$ or $S' = \emptyset$, then it returns $(S', \tilde{h}')$, where $\tilde{h}'$ is the demand assignment induced by $\tilde{h}$. Otherwise it returns Self-Containment(S', u).

[3] This criterion can be achieved by imposing an additional constraint when computing the augmenting paths.

3.2 Properties of Dual-VCHC

Below we derive basic properties of our algorithm. Since the algorithm keeps the constraints feasible when increasing the dual variables, we know that Ψ is feasible for the dual LP for Π. In the following, we first show that h is a feasible demand assignment for Π as well. Then we derive properties we will be using when establishing the bi-approximation factor next section.

Feasibility of the demand assignment h

We begin with procedure Self-Containment. Let $(S', \tilde{h}')$ be the pair returned by procedure Self-Containment$(S \cup \{u\}, u)$. The following lemma shows that S' is indeed maximal.

▶ **Lemma 2.** *If there exists a $B \subseteq S \cup \{u\}$ such that B can fully-serve the demand in $E[B] \cap E^{\phi}$, then $B \subseteq S'$.*

Proof. Let $S_1, S_2, \ldots, S_k$, where $S_1 = S \cup \{u\} \supset S_2 \supset \ldots \supset S_k = S'$, denote the input of the procedure Self-Containment$(S \cup \{u\}, u)$ in each recursion.

Below we argue that $B \subseteq S_i$ implies that $B \subseteq S_{i+1}$ for all $1 \le i < k$. Let $\tilde{h}_B$ denote a maximum flow for the flow graph $\mathcal{G}(B)$. Since B can fully-serve the demand in $E[B] \cap E^{\phi}$, we know that $\tilde{h}_B(s^+, \tilde{e}) = d_e$ for all $e \in E[B] \cap E^{\phi}$.

Consider the flow function computed by Maxflow$(\mathcal{G}(S_i), u)$ and denote it by $\tilde{h}_i$. If $\tilde{h}_i(s^+, \tilde{e}) < d_e$ for some $e \in E[B] \cap E^{\phi}$, then we embed $\tilde{h}_B$ into $\tilde{h}_i$, i.e., cancel the flow from $E[B] \cap E^{\phi}$ to B in $\tilde{h}_i$ and replace it by $\tilde{h}_B$. We see that the resulting flow strictly increases and remains valid for $\mathcal{G}(S_i)$, which is a contradiction to the fact that $\tilde{h}_i$ is a maximum flow for $\mathcal{G}(S_i)$. Therefore, we know that $\tilde{h}_i(s^+, \tilde{e}) = d_e$ for all $e \in E[B] \cap E^{\phi}$ and the vertices of B must be included in S_{i+1}. This show that $B \subseteq S_i$ for all $1 \le i \le k$. ◀

The following lemma states the feasibility of this primal-dual process.

▶ **Lemma 3.** *E^{ϕ} becomes empty in polynomial time. Furthermore, the assignments computed by* Self-Containment *during the process form a feasible demand assignment.*

The cost incurred by h

Below we consider the cost incurred by the partial assignments computed by Self-Containment. Let V_S denote the set of vertices that have been included in the set S. For any vertex v that has saturated, we use (S'_v, h'_v) to denote the particular pair returned by Self-Containment such that $v \in S'_v$. Note that, this pair (S'_v, h'_v) is uniquely defined for each v that has saturated. Therefore, we know that $h_{e,v} = (h'_v)_{e,v}$ holds for any $e \in E[v]$.

In the rest of this section, we will simply use $h_{e,v}$ when it refers to $(h'_v)_{e,v}$ for simplicity of notations. Recall that $D_{h'_v}(v)$ denotes the amount of demand v receives in h'_v. We have the following proposition for the dual solution $\Psi = (y_e, z_v, g_{e,v}, \eta_v)$, which follows directly from the way the dual variables are raised.

▶ **Proposition 4.** *For any $v \in V$ such that $d^{\phi}(v) > c_v$ when saturated, the following holds:*
- *$z_v = y_e$ for all $e \in E[v]$ with $h_{e,v} > 0$.*
- *$\eta_v > 0$ only when $v \in V_S$.*

The following lemma gives the properties for vertices in V_S.

▶ **Lemma 5.** *For any $v \in V_S$, we have*
1. *$D_{h'_v}(v) = m_v \cdot c_v$.*
2. *$w_v \cdot m_v = D_{h'_v}(v) \cdot y_e - m_v \cdot \eta_v$ for all $e \in E[v]$ such that $h_{e,v} > 0$.*

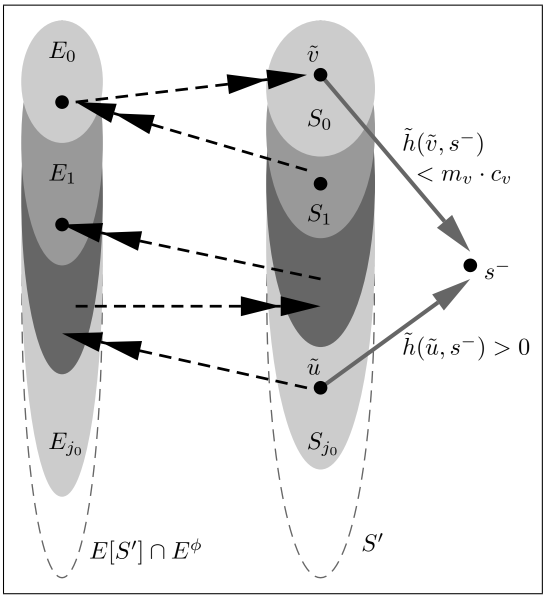


Figure 1 Alternating paths in the flow-graph $\mathcal{G}(S')$.

Proof. First we prove that $D_{h'_v}(v) < m_v \cdot c_v$. Without loss of generality, we assume that $m_v \geq 1$ and $D_{h'_v}(v) < m_v \cdot c_v$ for a contradiction.

Consider the iteration for which the vertex v was removed from S and let u be the vertex that becomes saturated in that iteration. By Lemma 2, we know that in the beginning of that iteration, $\nexists B \subseteq S$ such that B can fully-serve $E[B] \cap E^\phi$. Therefore it follows that $u \in S'_v$, for otherwise S'_v would have been removed from S in the previous iteration.

Consider the flow-graph $\mathcal{G}(S'_v)$ and the max-flow $\tilde{h}'_v$ to which h'_v corresponds. We know that $\tilde{h}'_v(\tilde{e}, \tilde{u}) = 0$ for all $e \in E[v] \cap E^\phi$, for otherwise we have an alternating path $\tilde{u} \to \tilde{e} \to \tilde{v}$ so that we can reroute the flow $\tilde{e} \to \tilde{u} \to s^-$ to $e \to \tilde{v} \to s^-$, which is a contradiction to the fact that the max-flow we compute is the one that minimizes the flow from $\tilde{u}$ to s^-.

Let $S_0 := \{v\}$ and $E_0 := E[v] \cap E^\phi$. For $i \geq 1$, consider the sets S_i and E_i defined as

$$S_i := \bigcup_{e \in E_{i-1}} \{v' \colon v' \in e \cap S'_v\} \quad \text{and} \quad E_i := E[S_i] \cap E^\phi.$$

Note that, $u \notin S_i$ implies that $S_i \subsetneq S_{i+1}$, for otherwise S_i would be a subset of S that can fully-serve $E[S_i] \cap E^\phi$ since the beginning of the iteration, a contradiction to Lemma 2. Therefore $u \in S_j$ for some $j \geq 1$ since $|S_i| \leq |S'_v| < \infty$. Let j_0 be the smallest integer such that $u \in S_{j_0}$. By definition we have $S_0 \subsetneq S_1 \subsetneq \ldots \subsetneq S_{j_0} \subseteq S'_v$. This corresponds to an alternating path to which we can reroute the flow from u to v, a contradiction. See also Fig. 1 for an illustration. Therefore we have $D_{h'_v}(v) = m_v \cdot c_v$.

For the second half of this lemma, since $v \in V_S$, we know that $d^\phi(v) > c_v$ before it gets saturated. Therefore, by Proposition 4, we know that $y_e = z_v$ holds for all $e \in E[v]$ such that $h_{e,v} > 0$. It follows that $w_v = c_v \cdot z_v - \eta_v = c_v \cdot y_e - \eta_v$ and $w_v \cdot m_v = D_{h'_v}(v) \cdot y_e - m_v \cdot \eta_v$ as claimed. ◀

The following auxiliary lemma, which is carried over from the previous primal-dual framework, shows that, for any vertex v with $d^\phi(v) \leq c_v$ when saturated, we can locate at most c_v units of demands from $E[v]$ such that their dual value pays for w_v. This statement holds intuitively since v is saturated.

▶ **Lemma 6.** *For any $v \in V$ with $d^{\phi}(v) \le c_v$ when saturated, we can compute a function $\ell_v \colon E[v] \to \mathbb{R}^{\ge 0}$ such that the following holds:*
(a) $0 \le h_{e,v} \le \ell_v(e) \le d_e$, *for all $e \in E[v]$.*
(b) $\sum_{e \in E[v]} \ell_v(e) \le c_v$.
(c) $\sum_{e \in E[v]} \ell_v(e) \cdot y_e = w_v$.

Intuitively, Proposition 4 and Lemma 5 provide a solid upper-bound for vertices whose capacity is fairly used. However, we remark that, this approach does not yield a solid guarantee for vertices whose capacity is barely used, i.e., $D_{h'_v}(v) \ll c_v$. The reason is that the demand that is served (charged) by vertices that have been included in S, i.e., those discussed in Lemma 5, cannot be charged again since their dual values are inflated during the primal-dual process.

4 Augmented Cover

In this section we establish the following theorem:

▶ **Theorem 7.** *For any integer $k \ge 2$, we can compute an augmented $\Big(k, (1 + \frac{1}{k-1})(f-1)\Big)$-cover for VC-HC in polynomial time.*

Let $\Pi = (V, E, d, w, c, m)$ be the input instance. Let $m'_v := k \cdot m_v$ denote the augmented multiplicity function for each $v \in V$. We invoke algorithm Dual-VCHC on the instance $\Pi' = (V, E, d, w, c, m')$. Let h be the demand assignment and $\Psi = (y, z, g, \eta)$ be the dual solution output by the algorithm for Π'.

The following observation is crucial in establishing the bi-approximation ratio: The dual solution Ψ, which was computed for instance Π', is also feasible for input instance Π.

▶ **Lemma 8.** *Ψ is feasible for LP(2) with respect to Π. In other words, we have*

$$\sum_{e \in E} d_e \cdot y_e - \sum_{v \in V} m_v \cdot \eta_v \;\le\; OPT(\Pi).$$

Proof. The statements follow directly since LP(2) has the same feasible region for Π and Π'. ◀

It is also worth mentioning that, the assignment h computed by Dual-VCHC already gives an augmented $\Big(k, (1 + \frac{1}{k-1})f\Big)$-cover. To obtain our claimed ratio, however, we further modify some of the demand assignments in h to achieve better utilization on the residue capacity of the vertices. Below we describe this procedure and establish the bi-approximation ratio.

Let V_S denote the set of vertices that have been included in S. For each $v \in V$ such that $D_h(v) < c_v$, let ℓ_v denote the function given by Lemma 6 with respect to v. We use h^* to denote the resulting assignment to obtain, where h^* is initialized to be h. For each $e \in E$, we repeat the following operation until no such vertex pair can be found:

- Find a vertex pair $u \in e \setminus V_S$ and $v \in e$ such that

$$\begin{cases} h^*_{e,u} > 0, \\ D_{h^*}(u) > c_u, \end{cases} \quad \text{and} \quad \begin{cases} D_h(v) < c_v, \\ h^*_{e,v} < \ell_v(e). \end{cases}$$

Then reassign $\min\left\{\, h^*_{e,u},\;\; \ell_v(e) - h^*_{e,v} \,\right\}$ units of demand of e from u to v.

In particular, we set $\begin{cases} h^*_{e,u} = h^*_{e,u} - R_{u,v}, \\ h^*_{e,v} = h^*_{e,v} + R_{u,v}, \end{cases}$ where $R_{u,v} := \min\left\{h^*_{e,u},\;\; \ell_v(e) - h^*_{e,v}\right\}$.

Intuitively, in assignment h^* if some demand is currently assigned to a vertex in $V \setminus V_S$ that requires multiple multiplicities, then we try to reassign it to vertices that have surplus residue capacity (according to the function ℓ_v) to balance the load. Note that, in this process we do not use additional multiplicities of the vertices, and the reassignments are performed only between vertices not belonging to V_S.

The following lemma shows that, the cost incurred by vertices in $V \setminus V_S$ can be distributed to the dual variables of the edges.

▶ **Lemma 9.** *We have*

$$\sum_{v \in V \setminus V_S} w_v \cdot x_v^{(h^*)} \leq (f-1) \cdot \sum_{v \in V_S} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e + f \cdot \sum_{v \in V \setminus V_S} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e.$$

The following lemma provides a lower bound for OPT(Π) in terms of the net sum of the dual values over the edges.

▶ **Lemma 10.** *We have*

$$\sum_{e \in E} d_e \cdot y_e \ \leq \ \frac{k}{k-1} \cdot \mathrm{OPT}(\Pi).$$

Proof. For each $v \in V_S$, by Lemma 5 we have $\sum_{e \in E[v]} h^*_{e,v} = m'_v \cdot c_v = k \cdot m_v \cdot c_v$. Furthermore, by the way how η_v is raised, we know that $\eta_v \ \leq \ c_v \cdot z_v \ = \ c_v \cdot y_e$ holds for all $e \in E[v]$ such that $h^*_{e,v} > 0$. Therefore, it follows that

$$m_v \cdot \eta_v \ \leq \ m_v \cdot c_v \cdot y_e \ \leq \ \frac{1}{k} \cdot \sum_{e \in E[v]} h^*_{e,v} \cdot y_e. \tag{3}$$

By Inequality (3) and Lemma 8, it follows that

$$\sum_{e \in E} \left(d_e - \frac{1}{k} \cdot \sum_{v \in e \cap V_S} h^*_{e,v} \right) \cdot y_e \ \leq \ \mathrm{OPT}(\Pi). \tag{4}$$

Therefore,

$$\begin{aligned} \sum_{e \in E} d_e \cdot y_e \ &= \ \sum_{v \in V} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e \ \leq \ \sum_{e \in E} \left(\frac{k}{k-1} \cdot d_e - \frac{1}{k-1} \cdot \sum_{v \in e \cap V_S} h^*_{e,v} \right) \cdot y_e \\ &= \ \frac{k}{k-1} \cdot \left(\sum_{e \in E} \left(d_e - \frac{1}{k} \cdot \sum_{v \in V_S \cap e} h^*_{e,v} \right) \cdot y_e \right) \\ &\leq \ \frac{k}{k-1} \cdot \mathrm{OPT}(\Pi), \end{aligned}$$

where the last inequality follows from Inequality (4). ◀

In the following we establish the bi-criteria approximation factor and prove Theorem 7.

▶ **Lemma 11.** *We have*

$$w(h^*) \leq \left(1 + \frac{1}{k-1} \right) \cdot (f-1) \cdot OPT(\Pi)$$

for any integer $k \geq 2$.

Proof. By Lemma 5, we have $D_h(v) = m'_v \cdot c_v = k \cdot m_v \cdot c_v$ for any $v \in V_S$. Therefore,

$$w_v \cdot x_v^{(h^*)} = (c_v \cdot z_v - \eta_v) \cdot k \cdot m_v = \sum_{e \in E[v]} h^*_{e,v} \cdot y_e - k \cdot m_v \cdot \eta_v.$$

Applying Lemma 9, we obtain

$$\begin{aligned} w(h^*) &= \sum_{v \in V_S} w_v \cdot x_v^{(h^*)} + \sum_{v \in (V \setminus V_S)} w_v \cdot x_v^{(h^*)} \\ &\le \left(\sum_{v \in V_S} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e - k \cdot \sum_{v \in V} m_v \cdot \eta_v \right) \\ &\qquad + \left((f-1) \cdot \sum_{v \in V_S} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e + f \cdot \sum_{v \in (V \setminus V_S)} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e \right) \\ &= f \cdot \sum_{v \in V} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e - k \cdot \sum_{v \in V} m_v \cdot \eta_v \\ &= k \cdot \left(\sum_{v \in V} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e - \sum_{v \in V} m_v \cdot \eta_v \right) + (f-k) \cdot \sum_{v \in V} \sum_{e \in E[v]} h^*_{e,v} \cdot y_e. \end{aligned}$$

The former item is upper-bounded by $k \cdot \mathrm{OPT}(\Pi)$ by Lemma 8. Combing the above with Lemma 10, we obtain

$$w(h^*) \le \left(k + (f-k) \cdot \frac{k}{k-1} \right) \cdot \mathrm{OPT}(\Pi) = \left(1 + \frac{1}{k-1} \right) \cdot (f-1) \cdot \mathrm{OPT}(\Pi)$$

as claimed. ◀

5 Conclusion

We conclude with some future directions. In this paper we presented bi-approximations for augmented multiplicity constraints. It is also interesting to consider VC-HC with relaxed demand constraints, i.e., partial covers. The reduction framework for partial VC-HC provided by Cheung et al. [3] and the tight approximation for VC-HC provided by Kao [10] jointly provided an almost tight $f + \epsilon$-approximation when the vertices are unweighted.

When the vertices are weighted, it is known that $O(\frac{1}{\epsilon})f$ bi-approximations can be obtained via simple LP rounding. Comparing to the $O(\frac{1}{\epsilon})$ bi-approximation result we can obtain for classical vertex cover, there is still a gap, and this would be an interesting direction to explore.

References

1 Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981. `doi:10.1016/0196-6774(81)90020-1`.

2 Reuven Bar-Yehuda, Guy Flysher, Julián Mestre, and Dror Rawitz. Approximation of partial capacitated vertex cover. *SIAM Journal on Discrete Mathematics*, 24(4):1441–1469, 2010. `doi:10.1137/080728044`.

3 W.-C. Cheung, M. Goemans, and S. Wong. Improved algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *SODA'14*, 2014. doi:10.1137/1.9781611973402.124.

4 Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM Journal on Computing*, 36(2):498–515, August 2006. doi:10.1137/S0097539703422479.

5 Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Aravind Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *J. Comput. Syst. Sci.*, 72:16–33, February 2006. doi:10.1016/j.jcss.2005.06.004.

6 F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. A primal-dual bicriteria distributed algorithm for capacitated vertex cover. *SIAM J. Comput.*, 38(3), 2008. doi:10.1137/06065310X.

7 Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering. *Journal of Algorithms*, 48(1):257–270, August 2003. doi:10.1016/S0196-6774(03)00053-1.

8 Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982. doi:10.1137/0211045.

9 Mong-Jen Kao. *An Algorithmic Approach to Local and Global Resource Allocations*. PhD thesis, National Taiwan University, 2012.

10 Mong-Jen Kao. Iterative partial rounding for vertex cover with hard capacities. manuscript, 2016.

11 Mong-Jen Kao, Han-Lin Chen, and D.T. Lee. Capacitated domination: Problem complexity and approximation algorithms. *Algorithmica*, November 2013. doi:10.1007/s00453-013-9844-6.

12 Mong-Jen Kao, Chung-Shou Liao, and D. T. Lee. Capacitated domination problem. *Algorithmica*, 60(2):274–300, June 2011. doi:10.1007/s00453-009-9336-x.

13 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, May 2008. doi:10.1016/j.jcss.2007.06.019.

14 Barna Saha and Samir Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *ICALP'12*, pages 762–773, 2012. doi:10.1007/978-3-642-31594-7_64.

15 L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982. doi:10.1007/BF02579435.

Surrogate Optimization for p-Norms*

Yasushi Kawase[1] **and Kazuhisa Makino**[2]

1 **School of Engineering, Tokyo Institute of Technology, Japan**
`kawase.y.ab@m.titech.ac.jp`
2 **Research Institute for Mathematical Sciences, Kyoto University, Japan**
`makino@kurims.kyoto-u.ac.jp`

Abstract

In this paper, we study the effect of surrogate objective functions in optimization problems. We introduce *surrogate ratio* as a measure of such effect, where the surrogate ratio is the ratio between the optimal values of the original and surrogate objective functions.

We prove that the surrogate ratio is at most $\mu^{|1/p-1/q|}$ when the objective functions are p- and q-norms, and the feasible region is a μ-dimensional space (i.e., a subspace of $\mathbb{R}^\mu$), a μ-intersection of matroids, or a μ-extendible system. We also show that this is the best possible bound. In addition, for μ-systems, we demonstrate that the ratio becomes $\mu^{1/p}$ when $p < q$ and unbounded if $p > q$. Here, a μ-system is an independence system such that for any subset of ground set the ratio of the cardinality of the largest to the smallest maximal independent subset of it is at most μ. We further extend our results to the surrogate ratios for approximate solutions.

1998 ACM Subject Classification G.2.0 General

Keywords and phrases surrogate optimization, matroid, extendible system, p-norm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.41

1 Introduction

When we model real-world problems as mathematical optimization problems, we often face some difficulties choosing appropriate objective functions for the problems. This, for instance, follows from ambiguity and computational difficulty of real objective functions. We demonstrate such examples below. In order to overcome such difficulties, it is natural to make use of surrogate objective functions.

Ambiguity: The first example is for ambiguous objective functions. For instance, consider car navigation system to find a fastest route from an origin to a destination in a road network. It usually solves the shortest path problem with estimated transit time for each road. This is simply because we do not know the actual transit time which depends on traffic congestion. Thus, estimated objective functions are used as surrogate ones.

Computational difficulty: Second, we sometimes approximate the objective function to reduce computational cost. For example, consider the following sparse approximation problem: we are given a vector $b \in \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m\times n}$ $(n \gg m)$, and we are asked to find a vector x to minimize $\|x\|_0$ $(= |\operatorname{supp}(x)|)$ subject to $Ax = b$. Unfortunately, the problem is computationally intractable (NP-hard) [19], and it is often replaced to minimizing $\|x\|_1$ $(= \sum_i^n |x_i|)$ or $\|x\|_2$ $(= \sum_i^n x_i^2)$ (see, e.g., [20]). The resulting problem can be regarded as a linear or convex programming problem, and thus, we can efficiently solve the surrogate optimization problem.

* The first author is supported by JSPS KAKENHI Grant Number 26887014 and 16K16005. The second author is supported by JSPS KAKENHI Grant.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 41; pp. 41:1–41:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- **Multi-objective optimization:** Consider a scenario in which we have multiple objectives. In this case, we sometimes use their weighted sum as a surrogate objective function. For instance, in a mean-variance portfolio optimization: given an expected return vector $\bar{p} \in \mathbb{R}^n$ and a variance-covariance matrix $V \in \mathbb{R}^{n\times n}$, we are asked to find a weight vector $x \in \mathbb{R}^n$ ($\sum_{i=1}^{n} x_i = 1$, $x_i \geq 0$ $\forall i = 1,\dots,n$) to maximize the expected return $\bar{p}^\top x$ and minimize the risk $x^\top V x$. A standard way to obtain a solution for the problem is to maximize $\bar{p}^\top x - \lambda x^\top V x$, where λ is called a risk-aversion coefficient [4].
- **Fairness-efficiency trade-off:** The next example follows from the trade-off between efficiency and fairness. Consider the following facility location problem: given a set of demand points $D \subseteq V$ in a metric space (V, d), we are asked to select k facilities $F \subseteq V$ to open while minimizing $\sqrt[p]{\sum_{i\in D}(\min_{j\in F} d(i,j))^p}$ for $p \geq 1$. We can see that the case of $p = 1$ (i.e., k-median problem) is efficient and the other extreme case $p = +\infty$ (i.e., k-center problem) is fair. Furthermore, an optimal solution for each p can be regarded as a solution which balances the efficiency and the fairness (see discussion in [7]). Golovin et al. [7] have suggested that the optimal solution for sufficiently large p is good for this trade-off. Namely, sufficient large p provides a good surrogate objective function.
- **Potential games:** The last example is in *potential games*. A game is said to be a potential game if the incentive of all players to change their strategy can be expressed by using a single global function called *potential function* [18]. Potential games always admit a pure Nash equilibrium and, in particular, any minimizer of the potential function is a pure Nash equilibrium. Thus, the potential minimizers are recognized as important solution concept in potential games. On the other hand, the efficiency of a solution is measured by the *social cost*, which is, for example, the sum or maximum of players' cost. For instance, consider the following load balancing game [21]. There are n users N and m identical machines M. Each user $i \in N$ has a job with weight w_i and chooses a machine to place the job. A combination of choices yields an assignment $A : N \to M$. The load of machine $j \in M$ under assignment A is defined as $l_j(A) = \sum_{i\in N:A(i)=j} w_i$. The cost of user $i \in N$ corresponds to the load on machine $A(i)$, i.e., $l_{A(i)}(A)$. Then, a potential for the problem is $\|l(A)\|_2^2$ ($= \sum_{j\in M} l_j(A)^2$) and a social cost is (usually) the makespan $\|l(A)\|_\infty$ ($= \max_{j\in M} l_j(A)$).

In this paper, we quantify the effect of surrogate objective functions by introducing *surrogate ratio*. The surrogate ratio compares the optimal solutions with respect to the original and surrogate objective functions. Our approach is analogous to the worst-case performance analysis in algorithm theory, such as the approximation ratio, the competitive ratio, and the robustness factor. As the first step in analyzing the surrogate ratio, this paper focuses on optimization problems of maximizing p-norms. Maximizing p-norms are well studied in many areas as shown above. For n-dimensional real vector $x \in \mathbb{R}^n$ and positive real $p \in \mathbb{R}_+$, p-norm of x is defined by $\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}$. We remark that $\|\cdot\|_p$ for p with $p < 1$ is *not* a norm, but we treat $\|\cdot\|_p$ for any positive real p. We also use the notation that $\|x\|_\infty = \lim_{p\to\infty} \|x\|_p = \max_i |x_i|$ for a real vector $x \in \mathbb{R}^n$.

Our model

We discuss the following four types of surrogate ratios between p- and q-norms for maximization problems with a compact non-empty feasible region $S \subseteq \mathbb{R}^\mu$:

$$\rho(S,p,q) = \max\{\|x\|_p : x \in S\}/\min\{\|x\|_p : x \in \arg\max_{x\in S} \|x\|_q\},$$

$$\eta(S,p,q) = \max\{\|x\|_p : x \in S\}/\max\{\|x\|_p : x \in \arg\max_{x\in S} \|x\|_q\},$$

$$\rho_\alpha(S,p,q) = \max\{\|x\|_p : x \in S\} / \min\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S} \|x\|_q\},$$
$$\eta_\alpha(S,p,q) = \max\{\|x\|_p : x \in S\} / \max\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S} \|x\|_q\},$$

where $\alpha \geq 1$ and

$$\alpha\text{-}\arg\max_{x\in S} f(x) = \{x \in S : f(x) \geq f(x')/\alpha\ (\forall x' \in S)\}.$$

Here, we assume for simplicity that the ratios are 1 if $S = \{0\}$. Then any optimal (α-approximate) solution for q-norm is a $\rho(S,p,q)$-approximate ($\rho_\alpha(S,p,q)$-approximate) solution for p-norm and there exists an optimal (α-approximate) solution for q-norm that is an $\eta(S,p,q)$-approximate ($\eta_\alpha(S,p,q)$-approximate) solution for p-norm. The ratios above are respectively called *the worst surrogate ratio*, *the best surrogate ratio*, *the worst α-approximation surrogate ratio*, and *the best α-approximation surrogate ratio*. By definitions, we have

$$1 \leq \eta_\alpha(S,p,q) \leq \eta(S,p,q) \leq \rho(S,p,q) \leq \rho_\alpha(S,p,q)$$

for any S, $p,q > 0$, and $\alpha \geq 1$. Moreover, $\rho(S,p,q) = \rho_1(S,p,q)$ and $\eta(S,p,q) = \eta_1(S,p,q)$ hold. We remark that the surrogate ratios between general two functions can be defined in the same way.

In this paper, we deal with *maximum weight independent set problems*, which are fundamental in combinatorial optimization and contain a number of important problems such as maximum weight stable set problem, maximum weight matching problem, and knapsack problem (see, e.g., [15]). An independence system is a set system $(E,\mathcal{F})$, i.e., E is a finite set and $\mathcal{F}$ is a family of subsets of E, with the following two properties: (I1) $\emptyset \in \mathcal{F}$ and (I2) $Y \subseteq X \in \mathcal{F}$ implies $Y \in \mathcal{F}$. Given an independence system $(E,\mathcal{F})$, a subset F of E is called *independent set* if F belongs to $\mathcal{F}$, and an (inclusion-wise) maximal independent set is called a *base*. For an independence system $(E,\mathcal{F})$ and non-negative weight $w(e)$ for $e \in E$, the maximization problem with p-norm is defined as $\max\{w^p(X) : X \in \mathcal{F}\}$ where we define

$$w^p(X) = \sqrt[p]{\sum\nolimits_{e\in X} w(e)^p}.$$

An independence system $(E,\mathcal{F})$ is called *matroid* if $X, Y \in \mathcal{F}, |X| < |Y|$ implies the existence of $v \in Y \setminus X$ such that $X \cup \{v\} \in \mathcal{F}$, and *$\mu$-intersection* if it is an intersection of μ matroids defined over E. As extensions of μ-intersection, we consider μ-extendible systems and μ-systems. An independence system $(E,\mathcal{F})$ is called *μ-extendible*[1] if

$$\forall X, Y \in \mathcal{F},\ \forall e \in Y \setminus X,\ \exists Z \subseteq X \setminus Y \text{ such that } |Z| \leq \mu,\ X \cup \{e\} \setminus Z \in \mathcal{F},$$

and *μ-system* if for all $S \subseteq E$ the ratio of the cardinality of the largest to the smallest maximal independent subset of S is at most μ. For simplicity, we assume that $\mu \geq 1$ in this paper (although we can take $\mu = 0$ for $(E, 2^E)$). It is known that classes of these systems have the following relationships [17]:

$$\mu\text{-intersection} \ \subseteq\ \mu\text{-extendible} \ \subseteq\ \mu\text{-system}.$$

For an independence system $(E,\mathcal{F})$ with a weight $w : E \to \mathbb{R}_+$, we denote the best surrogate ratio as $\rho(E,\mathcal{F},w;p,q)$. We also define $\eta(E,\mathcal{F},w;p,q)$, $\rho_\alpha(E,\mathcal{F},w;p,q)$, and $\eta_\alpha(E,\mathcal{F},w;p,q)$ similarly.

[1] Kakimura and Makino [11] called this system *μ-exchangeable*.

Table 1 Summary of the surrogate ratios for maximization problems.

	μ-dimensional space	μ-intersection	μ-extendible	μ-system
$\rho,\ \eta$	$\mu^{\lvert 1/p-1/q\rvert}$ [Thms. 2, 3]	$\mu^{\lvert 1/p-1/q\rvert}$ [Thm. 6]	$\mu^{\lvert 1/p-1/q\rvert}$ [Thm. 6]	$\begin{cases}\mu^{1/p} & (p<q),\\ \infty & \binom{p>q}{\mu>1},\\ 1 & \text{(otherwise)}\end{cases}$ [Thm. 13]
η_α $(\alpha > 1)$	$\max\left\{1, \frac{\mu^{\lvert 1/p-1/q\rvert}}{\alpha}\right\}$ [Thm. 3]	$\max\left\{1, \frac{\mu^{\lvert 1/p-1/q\rvert}}{\alpha}\right\}$ [Thm. 3]	$\max\left\{1, \frac{\mu^{\lvert 1/p-1/q\rvert}}{\alpha}\right\}$ [Thm. 3]	$\begin{cases}\mu^{1/p} & (p<q),\\ \infty & \binom{p>q}{\mu^{1/q}>\alpha},\\ 1 & \text{(otherwise)}\end{cases}$ [Thm. 13]
ρ_α $(\alpha > 1)$	$\alpha \cdot \mu^{\lvert 1/p-1/q\rvert}$ [Thm. 2]	$\begin{cases}\infty & (p \neq q)\\ \alpha & (p = q)\end{cases}$ [Thm. 9]	$\begin{cases}\infty & (p \neq q)\\ \alpha & (p = q)\end{cases}$ [Thm. 9]	$\begin{cases}\infty & (p \neq q)\\ \alpha & (p = q)\end{cases}$ [Thm. 9]

Our results

In this paper, we analyze the surrogate ratios. For a μ-dimensional compact feasible region $S \subseteq \mathbb{R}^\mu$, we show that the best and the worst surrogate ratios are both $\mu^{\lvert 1/p-1/q\rvert}$. Analogously, we prove that the best and worst surrogate ratios for a μ-intersection of matroids and a μ-extendible system are $\mu^{\lvert 1/p-1/q\rvert}$. On the other hand, for a μ-system with $\mu > 1$, we cannot bound the surrogate ratios by $\mu^{\lvert 1/p-1/q\rvert}$. The ratios become $\mu^{1/p}$ if $p < q$, and unbounded if $p > q$. Note that, optimality of a matroid (when $\mu = 1$) is independent of p, since the greedy algorithm, which always produces an optimal solution, does not need the values of weights but the ordering of them. Thus, the surrogate ratios are 1 in this case. Moreover, for any independence system, the greedy solution[2] coincides with an optimal solution for a p-norm with a sufficiently large p. Our result for μ-systems implies that the greedy solution is a μ-approximate solution by choosing $p = 1$ and q as a sufficiently large number, that is also shown by Jenkyns [10] and Korte and Hausmann [14]. The best α-approximation surrogate ratio is basically α times smaller than the previous one, i.e. the case $\alpha = 1$. On the contrary, the worst α-approximation surrogate ratio goes to infinity except for the μ-dimensional compact case.

Our results are summarized as Table 1.

Related work

The surrogate ratio is not just an analogy of the ratios shown below, but also closely related to them.

For a multi-objective or robust optimization problem, a natural measure of goodness of a solution is the ratio between the value of the solution and the optimal one for the worst objective function. The ratio is called *robustness factor* [9] (also studied under the name of *simultaneous approximation ratio* [6] or *global approximation ratio* [16]). To be more precise, assume that we want to maximize $f_1, \ldots, f_n$ under the constraint $x \in S$. Then a solution $x \in S$ is β-robust if $f_i(x) \geq f_i(y)/\beta$ holds for all $y \in S$ and $i = 1, \ldots, n$. Thus, we can obtain a β-robust solution by maximizing g, if the surrogate ratio is at most β for each f_i and g.

[2] To be precise, "greedy solution" may not be unique when there exist ties in the weights. However, we can perturb the weights slightly to break the ties with arbitrarily small changes in the values of p-norms.

Azar et al. [3] introduced a concept of an all-norm ρ-approximation algorithm, which supplies one solution that guarantees ρ-approximation to p-norms simultaneously. They gave a 2-approximation polynomial time algorithm for the p-norm load balancing problem (or rather, the problem of restricted assignment model).

Hassin and Rubinstein [9] studied a robustness version of maximum weight independent set problem when the objective functions are the sum of the k largest weights of selected elements for all positive integer k. They showed that, when the family of independent set is that of matchings in a graph, the optimal solution for p-norm ($p \geq 1$) is $\min\{2^{(1/p)-1}, 2^{-1/p}\}$-robust. This implies the existence of $\sqrt{2}$-robust solution for any graph by choosing $p = 2$. They also proved that $\sqrt{2}$-robust is the best possible. Fujita et al. [5] extended the result to the matroid intersection case. Kakimura and Makino [11] further extended the result to the μ-extendible system and showed that the optimal solution with respect to p-norm is $\min\{\mu^{(1/p)-1}, \mu^{-1/p}\}$-robust. We remark that this result does not imply our results.

For a potential game, consider a surrogate ratio of a social cost and a potential function. Then the surrogate ratio can be used to quantify the inefficiency of selfish behavior in the game. In fact, the ratio is studied under the name of the *inefficiency ratio of stable equilibria* [2] or the *potential-optimal price of anarchy* [13]. Note that, in algorithmic game theory, one of the most famous measures to quantify the inefficiency of a game is the *price of stability*, which is the ratio of the social cost at the best equilibrium to the minimum social cost possible [1]. An upper bound on the price of stability is often calculated by using the surrogate ratio. This bounding technique is called *potential function method* [1]. Moreover, we can see that the price of stability is a surrogate ratio where we do not replace the objective function but the feasible region is restricted from all the possible strategy profiles to the set of equilibria.

2 Surrogate ratios for μ-dimensional space

In this section, we study the surrogate ratios for μ-dimensional space. The following proposition plays an important role to obtain upper bounds on the surrogate ratios.

▶ **Proposition 1** (Norm Inequalities (see, e.g., [8])). *For any n-dimensional vector $x \in \mathbb{R}^n$ and $0 < p \leq q \leq \infty$, it holds that $\|x\|_q \leq \|x\|_p \leq n^{\frac{1}{p}-\frac{1}{q}}\|x\|_q$.*

We first evaluate the worst (α-approximation) surrogate ratio.

▶ **Theorem 2.** *For any $0 < p, q \leq \infty$ and $\alpha \geq 1$, we have*

$$\max_{S\subseteq\mathbb{R}^\mu:\substack{\textit{non-empty}\\ \textit{compact}}} \frac{\max\{\|x\|_p : x \in S\}}{\min\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q\}} = \alpha \cdot \mu^{|1/q-1/p|}. \tag{1}$$

Proof. We first claim that the left hand side of (1) is upper bounded by $\alpha \cdot \mu^{|1/q-1/p|}$. Let $M = \max\{\|x\|_q : x \in S\}$. Then we have

$$\frac{\max\{\|x\|_p : x \in S\}}{\min\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q\}} \leq \frac{\max\{\|x\|_p : \|x\|_q \leq M\}}{\min\{\|x\|_p : \|x\|_q \geq M/\alpha\}}.$$

By Proposition 1, if $p \leq q$, we have $\max\{\|x\|_p : \|x\|_q \leq M\} \leq \mu^{1/p-1/q} \cdot M$ and $\min\{\|x\|_p : \|x\|_q \geq M/\alpha\} \geq M/\alpha$. Otherwise (i.e., $p > q$), we have $\max\{\|x\|_p : \|x\|_q \leq M\} \leq M$ and $\min\{\|x\|_p : \|x\|_q \geq M/\alpha\} \geq \mu^{1/p-1/q} \cdot M/\alpha$. Therefore, we obtain $\max\{\|x\|_p : \|x\|_q \leq M\}/\min\{\|x\|_p : \|x\|_q \geq M/\alpha\} = \alpha \cdot \mu^{|1/q-1/p|}$.

Next, we show that there exists a μ-dimensional compact set S that attains the maximum in (1). Let $A_\gamma = \{a_\gamma, \mathbf{1}\}$ where $a_\gamma = (\gamma \cdot \mu^{1/q}, 0, \ldots, 0)^\top \in \mathbb{R}^\mu$ and $\mathbf{1} = (1, 1, \ldots, 1)^\top \in \mathbb{R}^\mu$.

Then, we can observe that the worst α-approximation surrogate ratio of A_α or $A_{1/\alpha}$ is $\alpha \cdot \mu^{|1/q-1/p|}$. ◀

This theorem yields that any α-approximate solution for q-norm is $\alpha \cdot \mu^{|1/q-1/p|}$-approximate solution for p-norm.

Next, we evaluate the best (α-approximation) surrogate ratio.

▶ **Theorem 3.** *For any $0 < p, q \leq \infty$ and $\alpha \geq 1$, we have*

$$\sup_{\substack{S \subseteq \mathbb{R}^\mu: \text{non-empty} \\ \text{compact}}} \frac{\max\{\|x\|_p : x \in S\}}{\max\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S} \|x\|_q\}} = \max\{1, \mu^{|1/q-1/p|}/\alpha\}. \tag{2}$$

Proof. We first claim that $\max\{\|x\|_p : x \in S\}/\max\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q\} \leq \max\{1, \mu^{|1/q-1/p|}/\alpha\}$ holds for any non-empty, compact set $S \subseteq \mathbb{R}^\mu$. Let $x_p \in \arg\max_{x\in S} \|x\|_p$. If $\alpha \geq \mu^{|1/q-1/p|}$, then $x_p \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q$ by Theorem 2 (with $\alpha = 1$). Thus, $\max\{\|x\|_p : x \in S\}/\max\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q\} = 1$. Otherwise, i.e., $\mu^{|1/q-1/p|} > \alpha \geq 1$, let r satisfies $\alpha = \mu^{|1/q-1/r|}$ and $\min\{p,q\} \leq r \leq \max\{p,q\}$, and let $x_r \in \arg\max_{x\in S}\|x\|_r$. Then $x_r \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q$ by Theorem 2 and hence

$$\begin{aligned}\frac{\max\{\|x\|_p : x \in S\}}{\max\{\|x\|_p : x \in \alpha\text{-}\arg\max_{x\in S}\|x\|_q\}} &\leq \frac{\max\{\|x\|_p : x \in S\}}{\|x_r\|_p} \\ &\leq \frac{\max\{\|x\|_p : x \in S\}}{\min\{\|x\|_p : x \in \arg\max_{x\in S}\|x\|_r\}} \\ &\leq \mu^{|1/r-1/p|} = \frac{\mu^{|1/q-1/p|}}{\alpha}\end{aligned}$$

where the last inequality holds by Theorem 2.

Conversely, the best α-approximation surrogate ratio of $A_{\alpha+\varepsilon}$ or $A_{1/(\alpha+\varepsilon)}$ in the proof of Theorem 2 converges to $\max\{1, \mu^{|1/q-1/p|}/\alpha\}$ as ε goes to $+0$. Thus, we obtain the theorem. ◀

This theorem implies that there exists $x \in (\alpha\text{-}\arg\max_{x\in S}\|x\|_p) \cap (\beta\text{-}\arg\max_{x\in S}\|x\|_q)$ if $\alpha\beta \geq \mu^{|1/p-1/q|}$.

3 Independence systems

In this section, we study some properties of independence systems.

For an independence system $(E, \mathcal{F})$ and $A \subseteq E$, define $\mathcal{F}|A = \{X : A \supseteq X \in \mathcal{F}\}$. Then $(A, \mathcal{F}|A)$ is called the *restriction* of $(E, \mathcal{F})$ to A. Also, define for $(E, \mathcal{F})$ and $A \subseteq E$, $\mathcal{F} \setminus A = \{X \setminus A : X \in \mathcal{F}\}$ and $\mathcal{F}/A = \{X \setminus A : A \subseteq X \in \mathcal{F}\}$. Then $(E \setminus A, \mathcal{F} \setminus A)$ and $(E \setminus A, \mathcal{F}/A)$ are called the *deletion* and the *contraction* of $(E, \mathcal{F})$ by A, respectively. If an independence system $(E, \mathcal{F})$ is μ-extendible, then $(A, \mathcal{F}|A)$, $(A, \mathcal{F} \setminus A)$, and $(E \setminus A, \mathcal{F}/A)$ are also μ-extendible [11].

Since a μ-extendible system is a μ-system, we have the following proposition.

▶ **Proposition 4.** *If $(E, \mathcal{F})$ is a μ-extendible system, then we have $|X| \leq \mu \cdot |Y|$ for any bases X, Y of $(E, \mathcal{F})$.*

Next, we see that the supremum of the worst surrogate ratio coincides with that of the best surrogate ratio for any independence system.

▶ **Lemma 5.** *For any independence system* $(E, \mathcal{F})$ *and* $p, q > 0$*, we have*

$$\sup_{w:E\to\mathbb{R}_+} \rho(E,\mathcal{F},w;p,q) = \sup_{w':E\to\mathbb{R}_+} \eta(E,\mathcal{F},w';p,q). \tag{3}$$

4 Surrogate ratios for μ-intersection and μ-extendible systems

In this section, we provide the maximum value of the surrogate ratio for μ-intersection and μ-extendible system. By Lemma 5, we only need to consider the worst one. We remind that we use the notation $\rho(E,\mathcal{F},w;p,q) = \frac{\max\{w^p(X):X\in\mathcal{F}\}}{\min\{w^p(X):X\in\arg\max_{X\in\mathcal{F}} w^q(X)\}}$ for the worst surrogate ratio.

We show that the tight bound of the surrogate ratio is $\mu^{|1/q-1/p|}$.

▶ **Theorem 6.** *For any* $0 < p \leq \infty$ *and* $0 < q < \infty$*, we have*

$$\max_{\substack{(E,\mathcal{F}):\mu\text{-intersection}\\ w:E\to\mathbb{R}_+}} \rho(E,\mathcal{F},w;p,q) = \max_{\substack{(E,\mathcal{F}):\mu\text{-extendible}\\ w:E\to\mathbb{R}_+}} \rho(E,\mathcal{F},w;p,q) = \mu^{|1/q-1/p|}.$$

We first provide the lower bound.

▶ **Lemma 7.** *For any* $0 < p \leq \infty$*,* $0 < q < \infty$*, and* μ (≥ 1)*, there exists a* μ*-intersection of matroids* $(E,\mathcal{F})$ *and a weight* $w : E \to \mathbb{R}_+$ *such that*

$$\rho(E,\mathcal{F},w;p,q) = \mu^{|1/q-1/p|}.$$

Proof. Let $\mathcal{F} = \{X : X = \{e_0\} \text{ or } X \subseteq L\}$ for $L = \{e_1,\dots,e_\mu\}$. Here, $(E,\mathcal{F})$ is a μ-intersection of matroids. In fact, $\mathcal{F} = \bigcap_{i=1}^{\mu} \mathcal{F}_i$ holds for partition matroids $\mathcal{F}_i = \{X \subseteq \{e_0, e_1, \dots, e_\mu\} : |X \cap \{e_0, e_i\}| \leq 1\}$. Let $w(e_0) = \mu^{1/q}$, $w(e_1) = w(e_2) = \cdots = w(e_l) = 1$. Then $w^p(\{e_0\}) = w^q(\{e_0\}) = \mu^{1/q}$ and $w^p(L) = \mu^{1/p}$, $w^q(L) = \mu^{1/q}$. Thus, the surrogate ratio is

$$\frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\min\{w^p(X) : X \in \arg\max_{X\in\mathcal{F}} w^q(X)\}} = \frac{\max\{\mu^{1/p}, \mu^{1/q}\}}{\min\{\mu^{1/p}, \mu^{1/q}\}} = \mu^{|1/q-1/p|},$$

which proves the lemma. ◀

We next present the upper bound.

▶ **Lemma 8.** *For any* $0 < p \leq \infty$ *and* $0 < q < \infty$*, any* μ*-extendible independence system* $(E,\mathcal{F})$ $(\mu \geq 1)$ *and any weight* $w : E \to \mathbb{R}_+$*, we have*

$$\rho(E,\mathcal{F},w;p,q) \leq \mu^{|1/q-1/p|}.$$

Proof. We assume that $p \neq q$ since the claim is obvious for the case $p = q$. We show $\rho(E,\mathcal{F},w;p,q) \leq \mu^{|1/q-1/p|}$ for any μ-extendible system $(E,\mathcal{F})$ and any weight $w : E \to \mathbb{R}_+$ by contradiction. We only prove the case $p > q$ because the case $p < q$ can be observed in a similar way. Assume that there exists a μ-extendible system $(E,\mathcal{F})$ and a weight $w : E \to \mathbb{R}_+$ such that $\rho(E,\mathcal{F},w;p,q) > \mu^{|1/q-1/p|}$. We choose such $(E,\mathcal{F})$ so that $|E|$ is as small as possible and w so that $\rho(E,\mathcal{F},w;p,q)$ is maximal for $(E,\mathcal{F})$. Such a w exists since we can pick

$$w \in \arg\max\left\{u^p(X_p) : \substack{u^p(X_q)=1,\ u^q(X_q)\geq u^q(X)\ (\forall X\in\mathcal{F}),\ u_e\geq 0\ (\forall e\in E),\\ X_p, X_q\in\mathcal{F}}\right\},$$

where the objective function is continuous and the feasible region is non-empty and compact. Here, the feasible region is bounded because $u_e \leq u^q(X_q) \leq |X_q|^{|1/q-1/p|} \cdot u^p(X_q) \leq |E|^{|1/q-1/p|}$ for each $e \in E$ and closed because intersection or union of finitely many closed sets is closed. Let

$$X_p \in \operatorname*{arg\,max}_{X\in\mathcal{F}} w^p(X) \quad \text{and} \quad X_q \in \arg\min\{w^p(X) : X \in \operatorname*{arg\,max}_{X\in\mathcal{F}} w^q(X)\}.$$

Without loss of generality, we may assume X_p and X_q are bases of $(E, \mathcal{F})$. We consider the following seven cases.

Case 1. $|X_p| = 1$. Let $X_p = \{e^*\}$. Then $|X_q| \leq \mu$ by Proposition 4. Therefore, we have that the surrogate ratio is at most

$$\frac{w^p(X_p)}{w^p(X_q)} \leq \frac{w(e^*)}{\min\{u^p(X_q) : u^q(X_q) \geq w(e^*)\}} = \frac{w(e^*)}{\mu^{1/p-1/q} \cdot w(e^*)} = \mu^{|1/q-1/p|}$$

where the second equality holds by the norm inequality (Proposition 1) and $|X_q| \leq \mu$.

Case 2. $w(e) = 0$ for some $e \in E$. In this case, $\rho(E, \mathcal{F}, w; p, q) = \rho(E \setminus \{e\}, \mathcal{F} \setminus \{e\}, w; p, q)$ holds since $X_p \setminus \{e\} \in \arg\max_{X\in\mathcal{F}\setminus\{e\}} w^p(X)$ and $X_q \setminus \{e\} \in \arg\min\{w^p(X) : X \in \arg\max_{X\in\mathcal{F}\setminus\{e\}} w^q(X)\}$. This contradicts the minimality of $|E|$.

Case 3. $X_p \cup X_q \subsetneq E$. Let $e \in E \setminus (X_p \cup X_q)$. Then $\rho(E, \mathcal{F}, w; p, q) \leq \rho(E \setminus \{e\}, \mathcal{F} \setminus \{e\}, w; p, q)$ since $X_p \in \operatorname*{arg\,max}_{X\in\mathcal{F}\setminus\{e\}} w^p(X)$ and $X_q \in \arg\min\{w^p(X) : X \in \operatorname*{arg\,max}_{X\in\mathcal{F}\setminus\{e\}} w^q(X)\}$. This contradicts the minimality of $|E|$.

Case 4. $X_p \cap X_q \neq \emptyset$. Let $e \in X_p \cap X_q$. Then $\rho(E, \mathcal{F}, w; p, q) \leq \rho(E \setminus \{e\}, \mathcal{F}/\{e\}, w; p, q)$ since $X_p \setminus \{e\} \in \operatorname*{arg\,max}_{X\in\mathcal{F}/\{e\}} w^p(X)$ and $X_q \setminus \{e\} \in \arg\min\{w^p(X) : X \in \operatorname*{arg\,max}_{X\in\mathcal{F}/\{e\}} w^q(X)\}$. This contradicts the minimality of $|E|$.

Case 5. There exists $X'_p \in \arg\max_{X\in\mathcal{F}} w^p(X)$ such that $X'_p \neq X_p$. We may assume that $w(e) > 0$ for any $e \in E$ by Case 2, $X_p \cup X_q = E$ by Case 3, and $X_p \cap X_q = \emptyset$ by Case 4. Then $X'_p \cup X_q \subsetneq X_p \cup X_q = E$ holds because $X'_p \cup X_q = E$ implies $X'_p \supsetneq X_p$ and $w(X'_p) > w(X_p)$, a contradiction. Thus we have $X'_p \in \arg\max_{X\in\mathcal{F}|(X'_p\cup X_q)} w^p(X)$ and $X_q \in \arg\min\{w^p(X) : X \in \arg\max_{X\in\mathcal{F}|(X'_p\cup X_q)} w^q(X)\}$. This contradicts the minimality $|E|$.

Case 6. There exists $X'_q \in \arg\max_{X\in\mathcal{F}} w^q(X)$ such that $X'_q \notin \{X_p, X_q\}$. We may assume that $w(e) > 0$ for any $e \in E$ by Case 2, and $X_p \cup X_q = E$ and $X_p \cap X_q = \emptyset$ by Cases 3 and 4. Then $w^p(X_p)/w^p(X_q)$ is at most

$$\sqrt[p]{\max\left\{\frac{\sum_{e\in X_p\setminus X'_q} w(e)^p}{\sum_{e\in X_q\cap X'_q} w(e)^p}, \frac{\sum_{e\in X_p\cap X'_q} w(e)^p}{\sum_{e\in X_q\setminus X'_q} w(e)^p}\right\}} = \max\left\{\frac{w^p(X_p \setminus X'_q)}{w^p(X_q \cap X'_q)}, \frac{w^p(X_p \cap X'_q)}{w^p(X_q \setminus X'_q)}\right\}$$

by the mediate inequality. Let $\mathcal{F}_1 = (\mathcal{F}|(X_p \cup X'_q))/(X_p \cap X'_q)$ and $\mathcal{F}_2 = (\mathcal{F}|(X_q \cup X'_q))/(X_q \cap X'_q)$. Then $X_p \setminus X'_q \in \arg\max_{X\in\mathcal{F}_1} w^p(X)$, $X_q \cap X'_q \in \arg\max_{X\in\mathcal{F}_1} w^q(X)$, $X_p \cap X'_q \in \mathcal{F}_2$, and $X_q \setminus X'_q \in \arg\max_{X \in \mathcal{F}_2} w^q(X)$. Thus, we have

$$\frac{w^p(X_p \setminus X'_q)}{w^p(X_q \cap X'_q)} \le \frac{\max_{X \in \mathcal{F}_1} w^p(X)}{\min\{w^p(X) : X \in \arg\max_{X \in \mathcal{F}_1} w^q(X)\}} \le \mu^{|1/q-1/p|},$$
$$\frac{w^p(X_p \cap X'_q)}{w^p(X_q \setminus X'_q)} \le \frac{\max_{X \in \mathcal{F}_2} w^p(X)}{\min\{w^p(X) : X \in \arg\max_{X \in \mathcal{F}_2} w^q(X)\}} \le \mu^{|1/q-1/p|}$$

by the minimality of $|E|$ and hence we have $w^p(X_p)/w^p(X_q) \le \mu^{|1/q-1/p|}$, a contradiction.

Case 7. The other case, i.e., $|X_p| \ge 2$, $w^p(X_p) > w^p(X)$ for any $X \in \mathcal{F} \setminus \{X_p\}$, $w^q(X_q) > w^q(X)$ for any $X \in \mathcal{F} \setminus \{X_p, X_q\}$, $X_p \cap X_q = \emptyset$, $X_p \cup X_q = E$, and $w(e) > 0$ for any $e \in E$. Let $s, t \in X_p$ such that $s \neq t$ and $w(s) \ge w(t)$. For a sufficiently small positive number ε, define

$$\hat{w}_e = \begin{cases} w(e) & (e \in E \setminus \{s,t\}), \\ (w(e)^q + \varepsilon)^{1/q} & (e = s), \\ (w(e)^q - \varepsilon)^{1/q} & (e = t). \end{cases}$$

Recall that $q < \infty$. Then $X_p \in \arg\max_{X \in \mathcal{F}} \hat{w}^p(X_p)$ and $\{X_q\} = \arg\max_{X \in \mathcal{F}} \hat{w}^q(X)$. Here, $\hat{w}^p(X_p) > w^p(X_p)$ and $\hat{w}^p(X_q) = w^p(X_q)$. Thus $\rho(E, \mathcal{F}, w; p, q) = w^p(X_p)/w^p(X_q) < \hat{w}^p(X_p)/\hat{w}^p(X_q) = \rho(E, \mathcal{F}, \hat{w}; p, q)$, which contradicts the maximality of $\rho(E, \mathcal{F}, w; p, q)$. ◀

By Lemmas 7 and 8, we obtain Theorem 6.

5 Worst α-approximation surrogate ratio

In this section, we prove that the worst α-approximation surrogate ratio ($\alpha > 1$)

$$\rho_\alpha(E, \mathcal{F}, w; p, q) = \frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\min\{w^p(X) : X \in \alpha\text{-}\arg\max_{X \in \mathcal{F}} w^q(X)\}}$$

is unbounded even if $(E, \mathcal{F})$ is a free matroid (i.e., $\mathcal{F} = 2^E$), when $p \neq q$.

▶ **Theorem 9.** *For any $\alpha > 1$ and $0 < p \le \infty$, $0 < q < \infty$ ($p \neq q$), there exists a sequence of matroids $(E_k, \mathcal{F}_k)$ and non-negative weights $w_k : E_k \to \mathbb{R}_+$ such that $\lim_{k \to \infty} \rho_\alpha(E_k, \mathcal{F}_k, w_k; p, q) = \infty$.*

Proof. Let $E = \{e_1, \ldots, e_k, e_{k+1}\}$ and let $w_k(e_1) = \cdots = w_k(e_k) = 1$, $w_k(e_{k+1}) = \sqrt[q]{(\alpha^q - 1) \cdot k}$. Define $\mathcal{F} = 2^E$. Then $\max_{X \in \mathcal{F}} w_k^p(X) = w_k^p(E) = \sqrt[p]{((\alpha^q-1) \cdot k)^{p/q} + k}$ and $\max_{X \in \mathcal{F}} w_k^q(X) = w_k^q(E) = \alpha \cdot k^{1/q}$. Here, $A = \{e_1, \ldots, e_k\}$ is an α-approximate solution for w_k^q since $w_k^q(A) = k^{1/q}$. Thus, the surrogate ratio of α-approximation is at least

$$\frac{w_k^p(E)}{w_k^p(A)} = \frac{\sqrt[p]{((\alpha^q-1) \cdot k)^{p/q} + k}}{k^{1/p}} = \sqrt[p]{(\alpha^q - 1)^{p/q} \cdot k^{p/q-1} + 1} \to \infty \quad (k \to \infty)$$

when $p > q > 0$.

The proof for the case $q > p > 0$ is similar. ◀

We remark that $\rho_\alpha(E, \mathcal{F}, w; p, p) \le \alpha$ holds by the definition. In addition, it holds that $\rho_\alpha(E, \mathcal{F}, w; p, p) = \alpha$ when $E = \{x, y\}$, $\mathcal{F} = 2^E$, and $w(x) = \alpha - 1$, $w(y) = 1$.

6 Best α-approximation surrogate ratio

In this section, we provide the best α-approximation surrogate ratio ($\alpha > 1$)

$$\eta_\alpha(E, \mathcal{F}, w; p, q) = \frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\max\{w^p(X) : X \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)\}}$$

for μ-intersection of matroids and μ-extendible systems.

▶ **Theorem 10.** *For any $0 < p \leq \infty$, $0 < q < \infty$, and $\alpha \geq 1$, we have*

$$\sup_{\substack{(E,\mathcal{F}):\mu\text{-intersection}\\ w:E\to\mathbb{R}_+}} \eta_\alpha(E, \mathcal{F}, w; p, q) = \sup_{\substack{(E,\mathcal{F}):\mu\text{-extendible}\\ w:E\to\mathbb{R}_+}} \eta_\alpha(E, \mathcal{F}, w; p, q) = \max\left\{1, \frac{\mu^{|1/q-1/p|}}{\alpha}\right\}.$$

We first provide the lower bound.

▶ **Lemma 11.** *For any $0 < p \leq \infty$, $0 < q < \infty$, and $\alpha \geq 1$, integer μ (≥ 1), and $\varepsilon > 0$, there exists a μ-intersection of matroids $(E, \mathcal{F})$ and a weight $w : E \to \mathbb{R}_+$ such that*

$$\frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\max\{w^p(X) : X \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)\}} = \max\{1, \mu^{|1/q-1/p|}/(\alpha + \varepsilon)\}.$$

Proof. Let $\mathcal{F} = \{X : X = \{e_0\}$ or $X \subseteq B\}$ for $B = \{e_1, \dots, e_\mu\}$. Here, $(E, \mathcal{F})$ can be viewed as μ-intersection of matroids. In fact, $\mathcal{F} = \bigcap_{i=1}^{\mu} \mathcal{F}_i$ when $\mathcal{F}_i = \{X \subseteq E : |X \cap \{e_0, e_i\}| \leq 1\}$. Then the lemma holds, for the case $p < q$, by setting $w(e_0) = (\alpha + \varepsilon) \cdot \mu^{1/q}$ and $w(e_1) = w(e_2) = \cdots = w(e_\mu) = 1$. Also, for the case $p > q$, we can observe the lemma by analyzing the weights $u(e_0) = \mu^{1/q}/(\alpha + \varepsilon)$ and $u(e_1) = u(e_2) = \cdots = u(e_\mu) = 1$. ◀

We next present the upper bound.

▶ **Lemma 12.** *For any $p, q > 0$, $\alpha \geq 1$, a μ-extendible independence system $(E, \mathcal{F})$ and a weight $w : E \to \mathbb{R}_+$, we have*

$$\frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\max\{w^p(X) : X \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)\}} \leq \max\{1, \mu^{|1/q-1/p|}/\alpha\}.$$

Proof. Let $X_p \in \arg\max_{X\in\mathcal{F}} w^p(X)$ and $X_q \in \arg\max_{X\in\mathcal{F}} w^q(X)$. If $\alpha \geq \mu^{|1/q-1/p|}$, then $X_p \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)$ by Lemma 8. Thus, $\max\{w^p(X) : X \in \mathcal{F}\}/\max\{w^p(X) : X \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)\} = 1$.

Otherwise, i.e., $\mu^{|1/q-1/p|} > \alpha \geq 1$, let r satisfies $\alpha = \mu^{|1/q-1/r|}$ and $\min\{p, q\} \leq r \leq \max\{p, q\}$, and let $X_r \in \arg\max_{X\in\mathcal{F}} w^r(X)$. Then $X_r \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)$ by Lemma 8 and hence

$$\begin{aligned}
\frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\max\{w^p(X) : X \in \alpha\text{-}\arg\max_{X\in\mathcal{F}} w^q(X)\}} &\leq \frac{\max\{w^p(X) : X \in \mathcal{F}\}}{w^p(X_r)} \\
&\leq \frac{\max\{w^p(X) : X \in \mathcal{F}\}}{\min\{w^p(X) : X \in \arg\max_{X\in\mathcal{F}} w^r(X)\}} \\
&\leq \mu^{|1/r-1/p|} = \frac{\mu^{|1/q-1/p|}}{\alpha}
\end{aligned}$$

where the last inequality holds by Lemma 8. ◀

By Lemmas 11 and 12, we get Theorem 10.

7 Surrogate ratios for μ-system

In this section, we show the surrogate ratios for μ-systems. By Theorem 9, the worst α-approximation surrogate ratio ($\alpha > 1$) goes to infinity when $p \neq q$. Hence, we here only analyze the best (approximation) surrogate ratio.

▶ **Theorem 13.** *For any $p, q > 0$ and $\alpha \geq 1$, we have*

$$\sup_{\substack{(E,\mathcal{F}):\mu\text{-system}\\ w:E\to\mathbb{R}_+}} \eta_\alpha(E,\mathcal{F},w;p,q) = \begin{cases} \mu^{1/p} & (p<q),\\ \infty & (p>q,\ \mu^{1/q} > \alpha),\\ 1 & (\textit{otherwise}). \end{cases}$$

We first prove the lower bound.

▶ **Lemma 14.** *For any $p, q > 0$, μ (≥ 1), and $\alpha \geq 1$, there exists a sequence of μ-systems $(E_k, \mathcal{F}_k)$ and weights $w_k : E_k \to \mathbb{R}_+$ ($k = 1, 2, \dots$) such that*

$$\lim_{k\to\infty} \eta_\alpha(E_k,\mathcal{F}_k,w_k;p,q) = \begin{cases} \mu^{1/p} & (p<q),\\ \infty & (p>q,\ \mu^{1/q} > \alpha). \end{cases}$$

Proof. Let $E_k = \{e_1, e_2, \dots, e_{k\cdot\mu}, f\}$, $\mathcal{F}_k = \{F \subseteq E_k : f \notin F \text{ or } |F| \leq k\}$. Then $(E_k, \mathcal{F}_k)$ is a μ-system. Let $X = \{e_1, \dots, e_{k\cdot\mu}\}$ and $Y_\sigma = \{f, e_{\sigma(1)}, \dots, e_{\sigma(k-1)}\}$ $(1 \leq \sigma(1) < \cdots < \sigma(k-1) \leq k\cdot\mu)$. We can see the lemma, for the case $p > q > 0$ and $\mu^{1/q} > \alpha \geq 1$, by setting $w_k(e_i) = 1$ $(i = 1, \dots, k\cdot\mu)$, $w_k(f) = \sqrt[q]{k\cdot(\mu/\beta - 1) + 1}$ where β is an arbitrary number such that $\mu > \beta > \alpha^q$. Also we can observe the lemma, for the case $q > p > 0$, by choosing $w_k(e_i) = 1$ $(i = 1, \dots, k\cdot\mu)$ and $w_k(f) = \alpha\sqrt[q]{k\cdot\mu}$. ◀

We next provide the upper bound for the case $p < q$.

▶ **Lemma 15.** *For any $q > p > 0$, μ-system $(E, \mathcal{F})$ ($\mu \geq 1$), and weight $w : E \to \mathbb{R}_+$, we have $\rho(E,\mathcal{F},w;p,q) \leq \mu^{1/p}$.*

Proof. Let $X_q = \{a_1, \dots, a_k\} \in \arg\min\{w^p(X) : X \in \arg\max_{X\in\mathcal{F}} w^q(X)\}$ and $X_p = \{b_1, \dots, b_l\} \in \arg\max_{X\in\mathcal{F}} w^p(X)$. Without loss of generality, we may assume X_p and X_q are bases. Thus, we have $l \leq \mu\cdot k$ because $(E, \mathcal{F})$ is a μ-system. We additionally assume that $w(a_1) \geq w(a_2) \geq \cdots \geq w(a_k)$ and $w(b_1) \geq w(b_2) \geq \cdots \geq w(b_l)$. For simplicity, define $w(b_i) = 0$ for $i > l$. Since $(E, \mathcal{F})$ is a μ-system, there exists a feasible set $\{a_1, \dots, a_i, b_{j_{i+1}}, \dots, b_{j_k}\}$ for each $i \in \{0, 1, \dots, k-1\}$ such that $j_t \leq (t-1)\cdot\mu + 1$ $(t = i+1, \dots, k)$. As X_q is an optimal solution for w^q, we have $w^q(\{a_1, \dots, a_k\}) \geq w^q(\{a_1, \dots, a_i, b_{j_{i+1}}, \dots, b_{j_k}\}) \geq w^q(\{a_1, \dots, a_i, b_{i\cdot\mu+1}, \dots, b_{(k-1)\cdot\mu+1}\})$ and thus

$$w(a_{i+1})^q + \cdots + w(a_k)^q \geq w(b_{i\cdot\mu+1})^q + \cdots + w(b_{(k-1)\cdot\mu+1}) \quad (i = 0, \dots, k-1).$$

Hence, we have $w(a_k)^p + w(a_{k-1})^p + \cdots + w(a_1)^p \geq w(b_{(k-1)\mu+1})^p + w(b_{(k-2)\mu+1})^p + \cdots + w(b_1)^p$ by Karamata's inequality [12]. (Karamata's inequality is also known as Hardy–Littlewood–Pólya inequality [8].) Therefore, we obtain

$$w^p(X_q) = \sqrt[p]{\sum_{i=1}^{k} w(a_i)^p} \geq \sqrt[p]{\sum_{i=1}^{k} w(b_{(i-1)\mu+1})^p} \geq \sqrt[p]{\frac{1}{\mu}\sum_{i=1}^{\mu\cdot k} w(b_i)^p} = \frac{1}{\mu^{1/p}}\cdot w^p(X_p),$$

which proves the lemma. ◀

Finally, we show the upper bound for the case $p > q$ and $\mu^{1/q} \leq \alpha$.

▶ **Lemma 16.** *For any $p > q > 0$, $\mu^{1/q} \leq \alpha$, μ-system $(E, \mathcal{F})$ ($\mu \geq 1$) and any weight $w : E \to \mathbb{R}_+$, we have $\eta_\alpha(E, \mathcal{F}, w; p, q) \leq 1$.*

Proof. By Lemma 15, there exists $X^* \in \arg\max\{w^p(X) : X \in \mathcal{F}\}$ such that $w^q(X^*) \geq \max\{w^q(X) : X \in \mathcal{F}\}/\mu^{1/q}$. Thus, we have $\eta_\alpha(E, \mathcal{F}, w; p, q) = 1$. ◀

Therefore, we get Theorem 13 by Lemmas 14, 15, and 16.

References

1 E. Anshelevich, A. Dasgupta, J. M. Kleinberg, É. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

2 A. Asadpour and A. Saberi. On the inefficiency ratio of stable equilibria in congestion games. In *Proceedings of WINE 2009*, pages 545–552. Springer, 2009.

3 Y. Azar, L. Epstein, Y. Richter, and G. J. Woeginger. All-norm approximation algorithms. *Journal of Algorithms*, 52(2):120–133, 2004.

4 F. J. Fabozzi, S. M. Focardi, and P. N. Kolm. *Quantitative Equity Investing: Techniques and Strategies.* Wiley, 2010.

5 R. Fujita, Y. Kobayashi, and K. Makino. Robust matchings and matroid intersections. *SIAM Journal on Discrete Mathematics*, 27:1234–1256, 2013.

6 A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. 44:301–323, 2006.

7 D. Golovin, A. Gupta, A. Kumar, and K. Tangwongsan. All-norms and all-l_p-norms approximation algorithms. In *Proceedings of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 199–210, 2008.

8 G. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities.* Cambridge University Press, 1952.

9 R. Hassin and S. Rubinstein. Robust matchings. *SIAM Journal on Discrete Mathematics*, 15(4):530–537, 2002.

10 T. A. Jenkyns. The efficacy of the "greedy" algorithm. In *Proceedings of the 7th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica, Winnipeg*, pages 341–350, 1976.

11 N. Kakimura and K. Makino. Robust independence systems. *SIAM Journal on Discrete Mathematics*, 27(3):1257–1273, 2013.

12 J. Karamata. Sur une inégalité relative aux fonctions convexes. *Publications de l'Institut mathematique*, 1(1):145–147, 1932.

13 Y. Kawase and K. Makino. Nash equilibria with minimum potential in undirected broadcast games. *Theoretical Computer Science*, 482:33–47, 2013.

14 B. Korte and D. Hausmann. An analysis of the greedy heuristic for independence systems. *Algorithmic aspects of combinatorics*, 2:65–74, 1978.

15 B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21. Springer, fifth edition.

16 A. Kumar and J. Kleinberg. Fairness measures for resource allocation. 36:657–680, 2006.

17 J. Mestre. Greedy in approximation algorithms. In *Proceedings of the 14th European Symposium on Algorithms*, pages 528–539, 2006.

18 D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

19 B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

20 I. Rish and G. Grabarnik. *Sparse Modeling: Theory, Algorithms, and Applications.* CRC press, 2014.

21 B. Vöcking. Selfish load balancing. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 20, pages 517–542. Cambridge, 2007.

Optimal Composition Ordering Problems for Piecewise Linear Functions*

Yasushi Kawase[1], Kazuhisa Makino[2], and Kento Seimi[3]

1 School of Engineering, Tokyo Institute of Technology, Japan
kawase.y.ab@m.titech.ac.jp

2 Research Institute for Mathematical Sciences, Kyoto University, Japan
makino@kurims.kyoto-u.ac.jp

3 Tokyo, Japan
kento.seimi@gmail.com

Abstract

In this paper, we introduce maximum composition ordering problems. The input is n real functions $f_1, \dots, f_n : \mathbb{R} \to \mathbb{R}$ and a constant $c \in \mathbb{R}$. We consider two settings: total and partial compositions. The maximum total composition ordering problem is to compute a permutation $\sigma : [n] \to [n]$ which maximizes $f_{\sigma(n)} \circ f_{\sigma(n-1)} \circ \cdots \circ f_{\sigma(1)}(c)$, where $[n] = \{1, \dots, n\}$. The maximum partial composition ordering problem is to compute a permutation $\sigma : [n] \to [n]$ and a nonnegative integer k $(0 \le k \le n)$ which maximize $f_{\sigma(k)} \circ f_{\sigma(k-1)} \circ \cdots \circ f_{\sigma(1)}(c)$.

We propose $\mathrm{O}(n \log n)$ time algorithms for the maximum total and partial composition ordering problems for monotone linear functions f_i, which generalize linear deterioration and shortening models for the time-dependent scheduling problem. We also show that the maximum partial composition ordering problem can be solved in polynomial time if f_i is of the form $\max\{a_i x + b_i, c_i\}$ for some constants $a_i\,(\ge 0)$, b_i and c_i. As a corollary, we show that the two-valued free-order secretary problem can be solved in polynomial time. We finally prove that there exists no constant-factor approximation algorithm for the problems, even if f_i's are monotone, piecewise linear functions with at most two pieces, unless P=NP.

1998 ACM Subject Classification G.2.1 Combinatorial algorithms

Keywords and phrases function composition, time-dependent scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.42

1 Introduction

In this paper, we introduce *optimal composition ordering problems* and mainly study their time complexity. The input of the problems is n real functions $f_1, \dots, f_n : \mathbb{R} \to \mathbb{R}$ and a constant $c \in \mathbb{R}$. In this paper, we assume that the input functions are piecewise linear, and the input length of a piecewise linear function is the sum of the sizes of junctions and coefficients of linear functions. We consider two settings: *total* and *partial* compositions. The maximum total composition ordering problem is to compute a permutation $\sigma : [n] \to [n]$ that maximizes $f_{\sigma(n)} \circ f_{\sigma(n-1)} \circ \cdots \circ f_{\sigma(1)}(c)$, where $[n] = \{1, \dots, n\}$. The maximum partial composition ordering problem is to compute a permutation $\sigma : [n] \to [n]$ and a nonnegative integer k $(0 \le k \le n)$ that maximize $f_{\sigma(k)} \circ f_{\sigma(k-1)} \circ \cdots \circ f_{\sigma(1)}(c)$. For example, if the input consists of $f_1(x) = 2x - 6$, $f_2(x) = \frac{1}{2}x + 2$, $f_3(x) = x + 2$, and $c = 2$, then the ordering σ such

* The first author is supported by JSPS KAKENHI Grant Number 26887014 and 16K16005. The second author is supported by JSPS KAKENHI Grant.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 42; pp. 42:1–42:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that $\sigma(1) = 2$, $\sigma(2) = 3$, and $\sigma(3) = 1$ is optimal for the maximum total composition ordering problem. In fact, $f_1 \circ f_3 \circ f_2(c) = f_1(f_3(f_2(c))) = f_1(f_3(c/2 + 2)) = f_1(c/2 + 4) = c + 2 = 4$ provides the optimal value of the problem. The ordering σ above and $k = 2$ is optimal for the maximum partial composition ordering problem, where $f_3 \circ f_2(c) = 5$. We also consider the maximum exact k-composition ordering problem, which is a problem to compute a permutation $\sigma : [n] \to [n]$ that maximizes $f_{\sigma(k)} \circ f_{\sigma(k-1)} \circ \cdots \circ f_{\sigma(1)}(c)$ for given n functions $f_1, \dots, f_n : \mathbb{R} \to \mathbb{R}$, a constant $c \in \mathbb{R}$, and a nonnegative integer k $(0 \leq k \leq n)$. We remark that the minimization versions are reducible to the maximization ones.

As we will see in this paper, the optimal composition ordering problems are natural and fundamental in many fields such as artificial intelligence, computer science, and operations research. However, to the best of the authors' knowledge, no one explicitly studies the problems from the algorithmic point of view. We below describe single machine time-dependent scheduling problems and the free-order secretary problem, which can be formulated as the optimal composition ordering problems.

Time-dependent scheduling

Consider machine scheduling problems with time-dependent processing times, called *time-dependent scheduling problems* [6, 12].

Let J_i $(i = 1, \dots, n)$ denote a job with a ready time $r_i \in \mathbb{R}$, a deadline $d_i \in \mathbb{R}$, and a processing time $p_i : \mathbb{R} \to \mathbb{R}$, where $r_i \leq d_i$ is assumed. Different from the classical setting, the processing time p_i is *not* constant, but depends on the *starting* time of job J_i. The model has been studied to deal with learning and deteriorating effects, for example [13, 14, 15, 19, 20]. Here each p_i is assumed to satisfy $p_i(t) \leq s + p_i(t + s)$ for any t and $s \geq 0$, since we should be able to finish processing job J_i earlier if it starts earlier. Among time-dependent settings, we consider the single machine scheduling problem to minimize the makespan, where the input is the start time t_0 $(= 0)$ and a set of J_i $(i = 1, \dots, n)$ above. The makespan denotes the time when all the jobs have finished processing, and we assume that the machine can handle only one job at a time and preemption is not allowed. We show that the problem can be viewed as the minimum total composition ordering problem.

Define a function f_i by

$$f_i(t) = \begin{cases} r_i + p_i(r_i) & (t \leq r_i), \\ t + p_i(t) & (r_i < t \leq d_i - p_i(t)), \\ \infty & (t > d_i - p_i(t)). \end{cases}$$

Then the problem can be reduced to the minimum total composition ordering problem for $(f_i)_{i \in [n]}$ and $c = t_0$.

A number of restrictions on the processing time $p_i(t)$ has been studied in the literature (e.g., [3, 5, 17]).

In the *linear deterioration* model, the processing time p_i is restricted to be a monotone increasing linear function that satisfies $p_i(t) = a_i t + b_i$ for two positive constants a_i and b_i. Here a_i and b_i are respectively called the *deterioration rate* and the *basic processing time* of job J_i. Gawiejnowicz and Pankowska [13], Gupta and Gupta [14], Tanaev *et al.* [19], and Wajs [20] obtained the result that the time-dependent scheduling problem of this model (without the ready time r_i nor the deadline d_i) is solvable in $\mathrm{O}(n \log n)$ time by scheduling jobs in the nonincreasing order of the ratios b_i/a_i. As for the hardness results, it is known that the proportional deterioration model with ready time and deadline, the linear deterioration model with ready time, and the linear deterioration model with deadlines are all NP-hard [4, 11].

Another model is called the *linear shortening* model introduced by Ho *et al.* [15]. In this model, the processing time p_i is restricted to be a monotone decreasing linear function that satisfies $p_i(t) = -a_i t + b_i$ with a_i and b_i such that $1 > a_i > 0$, $b_i > 0$. They showed that the time-dependent scheduling problem of this model can be solved in $\mathrm{O}(n \log n)$ time by again scheduling jobs in the nonincreasing order of the ratios b_i/a_i.

Free-order secretary problem

The *free-order secretary problem* is another application of the optimal composition ordering problems, which is closely related to a branch of the problems such as the full-information secretary problem [9], knapsack and matroid secretary problems [1, 2, 18] and stochastic knapsack problems [7, 8]. Imagine that an administrator wants to hire the best secretary out of n applicants for a position. Each applicant i has a nonnegative independent random variable X_i as his ability for the secretary. Here $X_1, \dots, X_n$ are not necessarily based on the same probability distribution, and assume that the administrator knows all the probability distributions of X_i's before their interviews, where such information can be obtained by their curriculum vitae and/or results of some written examinations. The applicants are interviewed one-by-one, and the administrator can observe the value X_i during the interview of the applicant i. A decision on each applicant is to be made immediately after the interview. Once an applicant is rejected, he will never be hired. The interview process is finished if some applicant is chosen, where we assume that the last applicant is always chosen if he is interviewed since the administrator has to hire exactly one candidate. The objective is to find an optimal strategy for this interview process, i.e., to find an interview ordering together with the stopping rule that maximizes the expected value of the secretary hired.

Let $f_i(x) = \mathbf{E}[\max\{X_i, x\}]$. We now claim that our secretary problem can be represented by the maximum total composition ordering problem $((f_i)_{i\in[n]}, c = 0)$.

Let us first consider the best stopping rule for the interview to maximize the expected value for the secretary hired when the interview ordering is fixed in advance. Assume that the applicant i is interviewed in the ith place. Note that $\mathbf{E}[X_n]\,(= f_n(0))$ is the expected value under the condition that all the applicants except for the last one are rejected, since the last applicant is hired. Consider the situation that all the applicants except for the last two ones are rejected. Then it is a best stopping rule that the applicant $n-1$ is hired if and only if $X_{n-1} \geq f_n(0)$ is satisfied (i.e., the applicant n is hired if and only if $X_{n-1} < f_n(0)$), where $f_{n-1} \circ f_n(0)$ is the expected value for the best stopping rule, under this situation. By applying backward induction, we have the following best stopping rule: we hire the applicant $i\,(< n)$ and stop the interview process, if $X_i \geq f_{i+1} \circ \cdots \circ f_n(0)$ (otherwise, the next applicant is interviewed), and we hire the applicant n if no applicant $i\,(< n)$ is hired. It turns out that $f_1 \circ \cdots \circ f_n(0)$ is the maximum expected value for the secretary hired, if the interview ordering is fixed such that the applicant i is interviewed in the ith place.

Therefore, the secretary problem (i.e., finding an interview ordering, together with a stopping rule) can be formulated as the maximum total composition ordering problem $((f_i)_{i\in[n]}, c = 0)$.

In addition, let us assume that X_i is an m-valued random variable that takes the value a_i^j with probability $p_i^j \geq 0$ $(j = 1, \dots, m)$. Here we assume that $a_i^1 \geq \cdots \geq a_i^m \geq 0$ and

$\sum_{j=1}^{m} p_i^j = 1$. Then we have

$$f_i(x) = \sum_{j=1}^{m} p_i^j \max\{a_i^j, x\} = \max_{l=0,\ldots,m} \left\{ \sum\nolimits_{j=1}^{l} p_i^j a_i^j + \sum\nolimits_{j=l+1}^{m} p_i^j x \right\}.$$

Note that this f_i is a monotone convex piecewise linear function with at most $(m+1)$ pieces.

Main results obtained in this paper

In this paper, we consider the computational issues for the optimal composition ordering problems, when all f_i's are monotone and almost linear.

We first show that the problems become tractable if all f_i's are monotone and linear, i.e., $f_i(x) = a_i x + b_i$ for $a_i \geq 0$.

▶ **Theorem 1.** *The maximum partial and total composition ordering problems for monotone nondecreasing linear functions are both solvable in* $\mathrm{O}(n \log n)$ *time.*

Recall that the algorithm for the linear shortening model (resp., the linear deterioration model) for the time-dependent scheduling problem is easily generalized to the case when all a_i's satisfy $a_i < 1$ (resp., $a_i > 1$). The best composition ordering is obtained as the nondecreasing order of the ratios b_i/a_i. This idea can be extended to the maximum partial composition ordering problem in the mixed case (i.e., some $a_i > 1$ and some $a_{i'} < 1$) of Theorem 1. However, we cannot extend it to the maximum total composition ordering problem. In fact, we do not know if there exists such a simple criterion on the maximum total composition ordering. We instead present an efficient algorithm that chooses the best ordering among linearly many candidates.

We also provide a dynamic-programming based polynomial-time algorithm for the exact k-composition setting.

▶ **Theorem 2.** *The maximum exact k-composition ordering problem for monotone nondecreasing linear functions is solvable in* $\mathrm{O}(k \cdot n^2)$ *time.*

We next consider monotone, piecewise linear case. It can be directly shown from the time-dependent scheduling problem that the maximum total composition ordering problem is NP-hard, even if all f_i's are monotone, concave, and piecewise linear functions with at most two pieces, i.e., $f_i(x) = \min\{a_i^1 x + b_i^1, a_i^2 x + b_i^2\}$ for some constants a_i^1, a_i^2, b_i^1, and b_i^2 with $a_i^1, a_i^2 > 0$. It turns out that all the other cases become intractable, even if all f_i's are monotone and consist of at most two pieces. Furthermore, the problems are inapproximable.

▶ **Theorem 3.**

(i) *For any positive real number α (≤ 1), there exists no α-approximation algorithm for the maximum total (partial) composition ordering problem even if all f_i's are monotone,* ***concave****, and piecewise linear functions with at most two pieces, unless P=NP.*

(ii) *For any positive real number α (≤ 1), there exists no α-approximation algorithm for the maximum total (partial) composition ordering problem even if all f_i's are monotone,* ***convex****, and piecewise linear functions with at most two pieces, unless P=NP.*

Here f_i can be represented by $f_i(x) = \max\{a_i^1 x + b_i^1, a_i^2 x + b_i^2\}$ for some constants a_i^1, a_i^2, b_i^1, and b_i^2 with $a_i^1, a_i^2 > 0$ if f_i is a monotone, convex, and piecewise linear function with at most two pieces.

As for the positive side, if each f_i is a monotone, convex, and piecewise linear function with at most two pieces such that one of the pieces is constant, then we have the following result, which implies that the two-valued free-order secretary problem can be solved in $\mathrm{O}(n^2)$ time.

▶ **Theorem 4.** *Let $f_i(x) = \max\{a_i x + b_i, c_i\}$ for some constants $a_i\,(\geq 0)$, b_i and c_i. Then the maximum partial composition ordering problem is solvable in $\mathrm{O}(n^2)$ time.*

Due to space limitation, we omit several proofs. They can be found in the full version of this paper [16].

2 Maximum Partial Composition Ordering Problem

In this section, we discuss tractable results for the maximum partial composition ordering problem for monotone and almost-linear functions. We deal with the problem as the maximum total composition ordering problem for functions $\overline{f}_i$ $(i \in [n])$, where $\overline{f}_i(x) = \max\{f_i(x), x\}$. It is easy to see that the objective value of the maximum partial composition ordering problem $((f_i)_{i\in[n]}, c)$ is equal to the one of the maximum total composition ordering problem $((\overline{f}_i)_{i\in[n]}, c)$. Let us start with the maximum partial composition ordering problem for monotone linear functions $f_i(x) = a_i x + b_i\,(a_i \geq 0)$, i.e., the total composition ordering problem for $\overline{f}_i(x) = \max\{a_i x + b_i, x\}$ $(a_i \geq 0)$.

The following binary relation $\preceq$ plays an important role in the problem.

▶ **Definition 5.** For two functions $f, g : \mathbb{R} \to \mathbb{R}$, we write $f \preceq g$ (or $g \succeq f$) if $f\circ g(x) \leq g\circ f(x)$ for any $x \in \mathbb{R}$, $f \simeq g$ if $f \preceq g$ and $f \succeq g$ (i.e., $f \circ g(x) = g \circ f(x)$ for any $x \in \mathbb{R}$), and $f \prec g$ (or $g \succ f$) if $f \preceq g$ and $f \not\simeq g$.

Note that the relation $\preceq$ is not *total relation* in general, here a relation $\preceq$ is called total if $f \preceq g$ or $g \preceq f$ for any f, g. For example, let $f_1(x) = \max\{2x, 3x\}$ and $f_2(x) = \max\{2x - 1, 3x + 1\}$. Then $f_1 \circ f_2(0)\,(= 3)$ is greater than $f_2 \circ f_1(0)\,(= 1)$, but $f_1 \circ f_2(-2)\,(= -10)$ is less than $f_2 \circ f_1(-2)\,(= -9)$.

However, if two consecutive functions are comparable, then we have the following easy but useful lemma.

▶ **Lemma 6.** *Let $f_1, \ldots, f_n$ be monotone nondecreasing functions. If $f_i \preceq f_{i+1}$, then it holds that $f_n \circ \cdots \circ f_{i+2} \circ f_{i+1} \circ f_i \circ f_{i-1} \circ \cdots \circ f_1(x) \geq f_n \circ \cdots \circ f_{i+2} \circ f_i \circ f_{i+1} \circ f_{i-1} \circ \cdots \circ f_1(x)$ for any $x \in \mathbb{R}$.*

It follows from the lemma that, for monotone functions f_i, there exists a maximum total composition ordering $f_n \circ f_{n-1} \circ \cdots \circ f_1$ that satisfies $f_1 \preceq f_2 \preceq \cdots \preceq f_n$, if the relation is total. Moreover, if the relation $\preceq$ is in addition transitive (i.e., $f \preceq g$ and $g \preceq h$ imply $f \preceq h$), then it is not difficult to see that $f_1 \preceq f_2 \preceq \cdots \preceq f_n$ becomes a sufficient condition that $f_n \circ f_{n-1} \circ \cdots \circ f_1$ is a maximum total composition ordering, where the proof is given as the more general form in Lemma 8.

The relation is total if all functions are linear or of the form $\max\{ax + b, x\}$ with $a \geq 0$.

▶ **Lemma 7.** *The relation $\preceq$ is total for linear functions.*

Proof. Let $f_i(x) = a_i x + b_i$ and $f_j(x) = a_j x + b_j$. Then we have

$$\begin{aligned} f_i \preceq f_j &\iff f_i \circ f_j(x) \leq f_j \circ f_i(x) \text{ for any } x \in \mathbb{R} \\ &\iff a_i(a_j x + b_j) + b_i \leq a_j(a_i x + b_i) + b_j \text{ for any } x \in \mathbb{R} \\ &\iff b_i(1 - a_j) \leq b_j(1 - a_i). \end{aligned} \tag{1}$$

Since the last inequality consists of only constants, we have $f_i \preceq f_j$ or $f_i \succeq f_j$. ◀

When all functions are of the form $\max\{ax+b,x\}$ with $a \geq 0$, the totality of the relation is proven in Lemma 11.

We further note that the relation $\preceq$ is transitive for linear functions $f(x) = ax + b$ with $a > 1$, since (1) is equivalent to $b_i/(1-a_i) \leq b_j/(1-a_j)$, and hence the ordering $b_1/(1-a_1) \leq b_2/(1-a_2) \leq \cdots \leq b_n/(1-a_n)$ gives an optimal solution for the maximum total composition ordering problem. Therefore, it can be solved efficiently by sorting the elements by $b_i/(1-a_i)$. The same statement holds when all linear functions have slope less than 1. This idea is used for the linear deterioration and linear shortening models for time-dependent scheduling problems. However, in general, this is not the case, i.e., the relation $\preceq$ does not satisfy transitivity. Let $f_1(x) = 2x+1$, $f_2(x) = 2x-1$, and $f_3(x) = x/2$. Then we have $f_1 \prec f_2$, $f_2 \prec f_3$, and $f_3 \prec f_1$, which implies that the transitivity is not satisfied for linear functions, and $\overline{f}_1 \prec \overline{f}_2$, $\overline{f}_2 \prec \overline{f}_3$, and $\overline{f}_3 \prec \overline{f}_1$ hold, implying that the transitivity is not satisfied for the functions of the form $\max\{ax+b,x\}$ with $a \geq 0$. These show that the maximum total and partial composition ordering problems are not trivial, even when all functions are monotone and linear.

We first show the following key lemma which can be used even for *non-transitive* relations.

▶ **Lemma 8.** *For monotone nondecreasing functions $f_i : \mathbb{R} \to \mathbb{R}$ $(i \in [n])$, if a permutation $\sigma : [n] \to [n]$ satisfies that $i \leq j$ implies $f_{\sigma(i)} \preceq f_{\sigma(j)}$ for any $i, j \in [n]$, then σ is an optimal solution for the maximum total composition ordering problem $((f_i)_{i\in[n]}, c)$.*

Proof. Without loss of generality, we may assume that σ is the identity permutation. Let σ' be an optimal solution for the maximum total composition ordering problem such that it has the minimum inversion number, where the inversion number denotes the number of pairs (i,j) with $i < j$ and $\sigma'(i) > \sigma'(j)$. Then we show that σ' is the identity permutation by contradiction. Assume that $\sigma'(l) > \sigma'(l+1)$ for some l. Then consider the following permutation:

$$\tau(i) = \begin{cases} \sigma'(i) & (i \neq l,\ l+1), \\ \sigma'(l+1) & (i = l), \\ \sigma'(l) & (i = l+1). \end{cases}$$

Since $\sigma'(l+1) < \sigma'(l)$ implies $f_{\sigma'(l+1)} \preceq f_{\sigma'(l)}$ by the condition of the identity σ, Lemma 6 implies that τ is also optimal for the problem. Since τ has an inversion number smaller than the one for σ', we derive a contradiction. Therefore, σ' is the identity. ◀

As mentioned above, if the relation $\preceq$ is in addition transitive (i.e., $\preceq$ is a total preorder), then such a σ always exists.

To efficiently solve the maximum partial composition ordering problem for the linear functions, we show that for $\overline{f}_i(x) = \max\{a_i x + b_i, x\}$ $(a_i \geq 0)$, (i) there exists a permutation σ which satisfies the condition in Lemma 8 and (ii) the permutation σ can be computed efficiently. Let us analyze the relation $\preceq$ in terms of the following γ.

▶ **Definition 9.** For a linear function $f(x) = ax + b$, we define

$$\gamma(f) = \begin{cases} \frac{b}{1-a} & (a \neq 1), \\ +\infty & (a = 1 \text{ and } b < 0), \\ -\infty & (a = 1 \text{ and } b \geq 0). \end{cases}$$

Note that $\gamma(f)$ is the solution of the equation $f(x) = x$ if $\gamma(f) \neq -\infty, +\infty$.

In the rest of the paper, we assume without loss of generality that no f_i is the identity (i.e., $f_i(x) = x$), since we can ignore identity function for both the total and partial composition problems.

▶ **Lemma 10.** *Let $f_i(x) = a_i x + b_i$ and $f_j(x) = a_j x + b_j$ be (non-identity) monotone nondecreasing functions. Then we have the following statements;*
(a) *if $a_i, a_j = 1$, then $f_i \simeq f_j$,*
(b) *if $a_i, a_j \geq 1$ and $a_i \cdot a_j > 1$, then $f_i \preceq f_j \Leftrightarrow \gamma(f_i) \leq \gamma(f_j)$,*
(c) *if $a_i, a_j < 1$, then $f_i \preceq f_j \Leftrightarrow \gamma(f_i) \leq \gamma(f_j)$,*
(d) *if $a_i \geq 1$, $a_j < 1$, then $f_i \preceq f_j \Leftrightarrow \gamma(f_i) \geq \gamma(f_j)$ and $f_i \succeq f_j \Leftrightarrow \gamma(f_i) \leq \gamma(f_j)$.*

▶ **Lemma 11.** *For (non-identity) monotone nondecreasing linear functions $f_i(x) = a_i x + b_i$ and $f_j(x) = a_j x + b_j$, we have the following statements;*
(a) *if $a_i, a_j \geq 1$ and $\gamma(f_i) \leq \gamma(f_j)$, then $\overline{f}_i \preceq \overline{f}_j$,*
(b) *if $a_i, a_j < 1$ and $\gamma(f_i) \leq \gamma(f_j)$, then $\overline{f}_i \preceq \overline{f}_j$,*
(c) *if $a_i < 1$, $a_j \geq 1$, and $\gamma(f_i) \leq \gamma(f_j)$, then $\overline{f}_i \simeq \overline{f}_j$,*
(d) *if $a_i \geq 1$, $a_j < 1$, and $\gamma(f_i) \leq \gamma(f_j)$, then $\overline{f}_i \succeq \overline{f}_j$.*

Note that Lemma 11 implies that the relation $\preceq$ is total for the functions of the form $\max\{ax + b, x\}$ with $a \geq 0$. Moreover, it implies that the following permutation σ satisfies the condition in Lemma 8.

For a linear function $f(x) = ax + b$, let $\delta(f) = +1$ if $a \geq 1$ and otherwise $\delta(f) = -1$. Let $\sigma : [n] \to [n]$ denote a permutation that is *compatible* with the lexicographic ordering with respect to $(\delta(f_i), \gamma(f_i))$, i.e., $(\delta(f_{\sigma(i)}), \gamma(f_{\sigma(i)}))$ is lexicographically smaller than or equal to $(\delta(f_{\sigma(j)}), \gamma(f_{\sigma(j)}))$ if $i < j$. Namely, there exists an integer k such that $0 \leq k \leq n$, $\delta(f_{\sigma(1)}) = \cdots = \delta(f_{\sigma(k)}) = -1$, $\delta(f_{\sigma(k+1)}) = \cdots = \delta(f_{\sigma(n)}) = +1$, $\gamma(f_{\sigma(1)}) \leq \cdots \leq \gamma(f_{\sigma(k)})$, and $\gamma(f_{\sigma(k+1)}) \leq \cdots \leq \gamma(f_{\sigma(n)})$. Then we have the following lemma by Lemma 11.

▶ **Lemma 12.** *For (non-identity) monotone nondecreasing linear functions f_i ($i \in [n]$), let σ denote a permutation compatible with the lexicographic order with respect to $(\delta(f_i), \gamma(f_i))$. Then $i \leq j$ implies $\overline{f}_{\sigma(i)} \preceq \overline{f}_{\sigma(j)}$ for any $i, j \in [n]$.*

By Lemmas 8 and 12, the maximum partial composition ordering problem for the monotone nondecreasing linear functions f_i, equivalently, the maximum total composition ordering problem for the functions $\overline{f}_i$ such that f_i's are monotone nondecreasing linear functions can be solved by computing the lexicographic order with respect to $(\delta(f_i), \gamma(f_i))$. Therefore, it can be solved in $O(n \log n)$ time, which proves the partial composition part of Theorem 1. We remark that the time complexity $O(n \log n)$ of the problem is the best possible in the comparison model. We also remark that the optimal value for the maximum partial composition ordering problem for $f_i(x) = a_i x + b_i$ ($a_i \geq 0$) forms a piecewise linear function (in c) with at most $(n + 1)$ pieces.

We next extend this tractability result to Theorem 4. For $i \in [n]$, let $h_i(x) = a_i x + b_i$ be a monotone nondecreasing linear function, and let $f_i(x) = \max\{h_i(x), c_i\}$ for a constant c_i. We consider the maximum partial composition ordering problem for f_i's. As mentioned in the introduction, the problem includes the two-valued free-order secretary problem, and it is a generalization of the maximum partial composition ordering problem for monotone linear functions.

We instead consider the maximum total composition ordering problem for the functions

$$\overline{f}_i(x) = \max\{a_i x + b_i, c_i, x\} \text{ for } a_i \in \mathbb{R}_+, b_i, c_i \in \mathbb{R}. \tag{2}$$

▶ **Lemma 13.** *Let $c \in \mathbb{R}$, and let $\overline{f}_i$ $(i \in [n])$ be a function defined as (2). Then there exists an optimal solution σ for the maximum total composition ordering problem $((\overline{f}_i)_{i\in[n]}, c)$ such that no $i\,(> 1)$ satisfies $\overline{h}_{\sigma(i)} \circ \overline{f}_{\sigma(i-1)} \circ \cdots \circ \overline{f}_{\sigma(1)}(c) < c_{\sigma(i)}$, where $h_i(x) = a_i x + b_i$.*

Proof. Let σ denote an optimal solution for the problem. Assume that there exists an index i that satisfies the condition in the lemma. Let i^* denote the largest such i. Then by the definition of i^*, we have $\overline{f}_{\sigma(i^*)} \circ \cdots \circ \overline{f}_{\sigma(1)}(c) = \overline{f}_{\sigma(i^*)}(c) = c_{\sigma(i^*)}$. It holds that $c_{\sigma(i)} < c_{\sigma(i^*)}$ for any i with $0 \leq i < i^*$, since $c_{\sigma(i)} \leq \overline{f}_{\sigma(i)} \circ \cdots \circ \overline{f}_{\sigma(1)}(c) \leq \overline{f}_{\sigma(i^*-1)} \circ \cdots \circ \overline{f}_{\sigma(1)}(c) < c_{\sigma(i^*)}$, where $c_{\sigma(0)} = c$ is assumed. Thus, we have $\overline{f}_{\sigma(n)} \circ \cdots \circ \overline{f}_{\sigma(1)}(c) = \overline{f}_{\sigma(n)} \circ \cdots \circ \overline{f}_{\sigma(i^*)}(c) \leq \overline{f}_{\sigma(i^*-1)} \circ \cdots \circ \overline{f}_{\sigma(1)} \circ \overline{f}_{\sigma(n)} \circ \cdots \circ \overline{f}_{\sigma(i^*)}(c)$. This implies that $(\sigma(i^*), \ldots \sigma(n), \sigma(1) \ldots, \sigma(i^*-1))$ is also an optimal permutation for the problem. Moreover, in the composition according to this permutation, the constant part of $\overline{f}_i$ $(i \neq i^*)$ is not explicitly used by the definition of i^* and $c_{\sigma(i)} < c_{\sigma(i^*)}$ for any $i\,(< i^*)$, which completes the proof. ◀

Proof of Theorem 4. It follows from Lemma 13 that an optimal solution for the problem can be obtained by solving the following $n + 1$ instances of the maximum partial composition ordering problem for monotone nondecreasing linear functions $((h_i)_{i\in[n]}, c)$ and $((h_i)_{i\in[n]\setminus\{k\}}, c_k)$ for all $k \in [n]$.

Therefore, we have an $\mathrm{O}(n^2 \log n)$-time algorithm by directly applying Theorem 1 to the problem. Moreover, we note that the maximum partial composition ordering problem for monotone nondecreasing linear functions can be solved in linear time, if we know the lexicographic ordering with respect to (δ, γ). This implies that the problem can be solved in $\mathrm{O}(n^2)$ time by first computing the lexicographic order with respect to $(\delta(h_i), \gamma(h_i))$. ◀

3 Maximum Total Composition Ordering Problem

In this section we prove the total composition part of Theorem 1 and Theorem 2.

The following lemma shows the relationships between $\gamma(f_i)$, $\gamma(f_j)$, $\gamma(f_j \circ f_i)$ and $\gamma(f_i \circ f_j)$ for monotone linear functions.

▶ **Lemma 14.** *For monotone nondecreasing linear functions $f_i(x) = a_i x + b_i$ and $f_j(x) = a_j x + b_j$ $(a_i, a_j \geq 0)$, we have the following statements.*

(a) *If $\gamma(f_i) = \gamma(f_j)$, then $\gamma(f_i) = \gamma(f_j) = \gamma(f_j \circ f_i)$,*
(b) *If $\gamma(f_i) < \gamma(f_j)$ and $a_i, a_j \geq 1$, then $\gamma(f_i) \leq \gamma(f_j \circ f_i) \leq \gamma(f_j)$,*
(c) *If $\gamma(f_i) < \gamma(f_j)$ and $a_i, a_j < 1$, then $\gamma(f_i) \leq \gamma(f_j \circ f_i) \leq \gamma(f_j)$,*
(d) *If $\gamma(f_i) < \gamma(f_j)$, $a_i < 1$, $a_j \geq 1$, and $a_i \cdot a_j \geq 1$, then $\gamma(f_j \circ f_i) \geq \gamma(f_j)$ $(> \gamma(f_i))$,*
(e) *If $\gamma(f_i) < \gamma(f_j)$, $a_i < 1$, $a_j \geq 1$, and $a_i \cdot a_j < 1$, then $\gamma(f_j \circ f_i) \leq \gamma(f_i)$ $(< \gamma(f_j))$,*
(f) *If $\gamma(f_i) < \gamma(f_j)$, $a_i \geq 1$, $a_j < 1$, and $a_i \cdot a_j \geq 1$, then $\gamma(f_j \circ f_i) \leq \gamma(f_i)$ $(< \gamma(f_j))$,*
(g) *If $\gamma(f_i) < \gamma(f_j)$, $a_i \geq 1$, $a_j < 1$, and $a_i \cdot a_j < 1$, then $\gamma(f_j \circ f_i) \geq \gamma(f_j)$ $(> \gamma(f_i))$.*

By Lemmas 10 and 14, we have the following inequalities for compositions of four functions.

▶ **Lemma 15.** *For monotone nondecreasing linear functions $f_i(x) = a_i x + b_i$ $(i = 1, 2, 3, 4)$, if $a_1, a_3 \geq 1$, $a_2, a_4 < 1$ and $\gamma(f_1) \geq \gamma(f_2) \geq \gamma(f_3) \geq \gamma(f_4)$, then we have*

$$f_4 \circ f_3 \circ f_2 \circ f_1(x) \leq \max\{f_4 \circ f_1 \circ f_3 \circ f_2(x),\ f_3 \circ f_2 \circ f_4 \circ f_1(x)\} \text{ for any } x.$$

▶ **Lemma 16.** *For monotone nondecreasing linear functions $f_i(x) = a_i x + b_i$ $(i = 1, 2, 3, 4)$, if $a_1, a_3 < 1$, $a_2, a_4 \geq 1$ and $\gamma(f_1) \geq \gamma(f_2) \geq \gamma(f_3) \geq \gamma(f_4)$, then we have*

$$f_4 \circ f_3 \circ f_2 \circ f_1(x) \leq \max\{f_4 \circ f_1 \circ f_3 \circ f_2(x),\ f_3 \circ f_2 \circ f_4 \circ f_1(x)\} \text{ for any } x.$$

Proof. We only prove Lemma 15 since the proof of Lemma 16 is similar. Let $g(x) = f_3 \circ f_2(x)$. If $a_2 \cdot a_3 \geq 1$, then $\gamma(g) \leq \gamma(f_3) \leq \gamma(f_1)$ holds by 1 and 6 in Lemma 14, and $g \circ f_1(x) \leq f_1 \circ g(x)$ holds by 1 and 2 in Lemma 10. Thus, we have $f_4 \circ f_3 \circ f_2 \circ f_1(x) \leq f_4 \circ f_1 \circ f_3 \circ f_2(x)$.

On the other hand, if $a_2 \cdot a_3 < 1$, then $\gamma(g) \geq \gamma(f_2) \geq \gamma(f_4)$ holds by 1 and 7 in Lemma 14, and $f_4 \circ g(x) \leq g \circ f_4(x)$ holds by 3 in Lemma 10. Thus, we have $f_4 \circ f_3 \circ f_2 \circ f_1(x) \leq f_3 \circ f_2 \circ f_4 \circ f_1(x)$. ◀

By Lemmas 15 and 16, we obtain the following lemma.

▶ **Lemma 17.** *There exists an optimal permutation σ for the maximum total composition ordering problem for monotone nondecreasing functions f_i ($i \in [n]$) such that at most two i's satisfy $\delta(f_{\sigma(i)}) \cdot \delta(f_{\sigma(i+1)}) = -1$.*

Next, we provide inequalities for compositions of three functions.

▶ **Lemma 18.** *For monotone nondecreasing linear functions $f_i(x) = a_i x + b_i$ ($i = 1, 2, 3$), if $a_1, a_3 \geq 1$, $a_2 < 1$, $a_1 \cdot a_2 \cdot a_3 \geq 1$ and $\gamma(f_1) \geq \gamma(f_2) \geq \gamma(f_3)$, then we have*

$$f_3 \circ f_2 \circ f_1(x) \leq \max\{f_2 \circ f_1 \circ f_3(x),\ f_1 \circ f_3 \circ f_2(x)\} \text{ for any } x.$$

▶ **Lemma 19.** *For monotone nondecreasing linear functions $f_i(x) = a_i x + b_i$ ($i = 1, 2, 3$), if $a_1, a_3 < 1$, $a_2 \geq 1$, $a_1 \cdot a_2 \cdot a_3 < 1$ and $\gamma(f_1) \geq \gamma(f_2) \geq \gamma(f_3)$, then we have*

$$f_3 \circ f_2 \circ f_1(x) \leq \max\{f_2 \circ f_1 \circ f_3(x),\ f_1 \circ f_3 \circ f_2(x)\} \text{ for any } x.$$

Proof. We only prove Lemma 18 since the proof of Lemma 19 is similar. If $a_2 \cdot a_3 \geq 1$, then $\gamma(f_3 \circ f_2) \leq \gamma(f_3) \leq \gamma(f_1)$ by 1 and 6 in Lemma 14, and it implies $f_3 \circ f_2 \circ f_1(x) \leq f_1 \circ f_3 \circ f_2(x)$ by 1 and 2 in Lemma 10. If $a_2 \cdot a_3 < 1$ and $\gamma(f_3 \circ f_2) \geq \gamma(f_1)$, then $f_3 \circ f_2 \circ f_1(x) \leq f_1 \circ f_3 \circ f_2(x)$ by 4 in Lemma 10.

If $a_1 \cdot a_2 \geq 1$, then $\gamma(f_2 \circ f_1) \geq \gamma(f_1) \geq \gamma(f_3)$ by 1 and 4 in Lemma 14, and it implies $f_3 \circ f_2 \circ f_1(x) \leq f_2 \circ f_1 \circ f_3(x)$ by 1 and 2 in Lemma 10. If $a_1 \cdot a_2 < 1$ and $\gamma(f_2 \circ f_1) \leq \gamma(f_3)$, then $f_3 \circ f_2 \circ f_1(x) \leq f_2 \circ f_1 \circ f_3(x)$ by 4 in Lemma 10.

Otherwise, we have $a_2 \cdot a_3 < 1$, $a_1 \cdot a_2 < 1$, $\gamma(f_3 \circ f_2) < \gamma(f_1)$, and $\gamma(f_2 \circ f_1) > \gamma(f_3)$. Then we have $\gamma((f_3 \circ f_2) \circ f_1) \geq \gamma(f_1)$ by 4 in Lemma 14, and $\gamma(f_3 \circ (f_2 \circ f_1)) \leq \gamma(f_3)$ by 6 in Lemma 14 since $a_1 \cdot a_2 \cdot a_3 \geq 1$. Therefore $\gamma(f_1) = \gamma(f_2) = \gamma(f_3)$, This together with $\gamma(f_3 \circ f_2) < \gamma(f_1)$ contradicts 1 in Lemma 14. ◀

By Lemmas 10, 14, 17, 18, and 19, we get the following lemmas.

▶ **Lemma 20.** *If $\prod_{i=1}^n a_i \geq 1$, then there exists an optimal permutation σ such that, for some two integers s, t ($0 \leq s \leq t \leq n$), $\delta(f_{\sigma(t+1)}) = \cdots = \delta(f_{\sigma(n)}) = \delta(f_{\sigma(1)}) = \cdots = \delta(f_{\sigma(s)}) = -1$, $\delta(f_{\sigma(s+1)}) = \cdots = \delta(f_{\sigma(t)}) = 1$, $\gamma_{\sigma(t+1)} \leq \cdots \leq \gamma_{\sigma(n)} \leq \gamma_{\sigma(1)} \leq \cdots \leq \gamma_{\sigma(s)}$, and $\gamma_{\sigma(s+1)} \leq \cdots \leq \gamma_{\sigma(t)}$.*

▶ **Lemma 21.** *If $\prod_{i=1}^n a_i < 1$, then there exists an optimal permutation σ such that, for some two integers s, t ($0 \leq s \leq t \leq n$), $\delta(f_{\sigma(t+1)}) = \cdots = \delta(f_{\sigma(n)}) = \delta(f_{\sigma(1)}) = \cdots = \delta(f_{\sigma(s)}) = 1$, $\delta(f_{\sigma(s+1)}) = \cdots = \delta(f_{\sigma(t)}) = -1$, $\gamma_{\sigma(t+1)} \leq \cdots \leq \gamma_{\sigma(n)} \leq \gamma_{\sigma(1)} \leq \cdots \leq \gamma_{\sigma(s)}$, and $\gamma_{\sigma(s+1)} \leq \cdots \leq \gamma_{\sigma(t)}$.*

Proof. We only prove Lemma 20 since the proof of Lemma 21 is similar. By Lemma 17, there exists an optimal permutation σ and two integers s, t ($0 \leq s \leq t \leq n$) such that

$\delta(f_{\sigma(1)}) = \cdots = \delta(f_{\sigma(s)}) = -\delta(f_{\sigma(s+1)}) = \cdots = -\delta(f_{\sigma(t)}) = \delta(f_{\sigma(t+1)}) = \cdots = \delta(f_{\sigma(n)})$. By Lemma 10, we have

$$\gamma_{\sigma(1)} \leq \cdots \leq \gamma_{\sigma(s)},\ \gamma_{\sigma(s+1)} \leq \cdots \leq \gamma_{\sigma(t)},\ \gamma_{\sigma(t+1)} \leq \cdots \leq \gamma_{\sigma(n)}.$$

This implies that the lemma holds when $s = 0$ or $t = n$. For $0 < s \leq t < n$, we separately consider the following two cases.

Case 1: If $\delta(f_{\sigma(s+1)}) = \cdots = \delta(f_{\sigma(t)}) = +1$, let $g = f_{\sigma(n-1)} \circ \cdots \circ f_{\sigma(2)}$. Then Lemma 10 and the optimality of σ imply $\gamma(f_{\sigma(1)}) \geq \gamma(g) \geq \gamma(f_{\sigma(n)})$, since $-\delta(f_{\sigma(1)}) = \delta(g) = -\delta(f_{\sigma(n)}) = +1$. This proves the lemma.

Case 2: If $\delta(f_{\sigma(s+1)}) = \cdots = \delta(f_{\sigma(t)}) = -1$, then let $h_1 = f_{\sigma(s)} \circ \cdots \circ f_{\sigma(1)}$, $h_2 = f_{\sigma(t)} \circ \cdots \circ f_{\sigma(s+1)}$ and $h_3 = f_{\sigma(n)} \circ \cdots \circ f_{\sigma(t+1)}$. If $\gamma(h_1) < \gamma(h_2)$, then $h_3 \circ h_2 \circ h_1(x) \leq h_3 \circ h_1 \circ h_2(x)$ by 4 in Lemma 10. If $\gamma(h_2) < \gamma(h_3)$, then $h_3 \circ h_2 \circ h_1(x) \leq h_2 \circ h_3 \circ h_1(x)$ by 4 in Lemma 10. Otherwise (i.e., $\gamma(h_1) \geq \gamma(h_2) \geq \gamma(h_3)$), we have

$$h_3 \circ h_2 \circ h_1(x) \leq \max\{h_2 \circ h_1 \circ h_3(x),\ h_1 \circ h_3 \circ h_2(x)\}$$

by Lemma 18. In either case, we can obtain a desired optimal solution by modifying σ. ◀

By Lemmas 20 and 21, we obtain polynomial time algorithm for the maximum total composition ordering problem for monotone nondecreasing linear functions.

Proof of the total composition part of Theorem 1. By Lemmas 20 and 21, the total composition ordering problem for monotone nondecreasing linear functions can be computed as follows. Let $\sigma : [n] \to [n]$ be a permutation which satisfies $\delta(f_{\sigma(1)}) = \cdots = \delta(f_{\sigma(r)}) = -1$, $\delta(\sigma(r+1)) = \cdots = \delta(f_{\sigma(n)})$, $\gamma(f_{\sigma(1)}) \leq \cdots \leq \gamma(f_{\sigma(r)})$, and $\gamma(f_{\sigma(r+1)}) \leq \cdots \leq \gamma(f_{\sigma(n)})$. Then Lemmas 20 and 21 implies that there exists an optimal solution of the form $(\sigma(t), \sigma(t+1), \ldots, \sigma(n), \sigma(1), \sigma(2), \ldots, \sigma(t-1))$ for some t. Therefore, the problem can be computed in polynomial time by checking n permutations above. To reduce the time complexity, let $d_k = f_{\sigma(k-1)} \circ \cdots \circ f_{\sigma(1)} \circ f_{\sigma(n)} \circ \cdots \circ f_{\sigma(k)}(c)$ for $k = 1, \ldots, n$. Let $a = \prod_{i=1}^{n} a_i$. Then it is not difficult to see that $d_{k+1} = a_{\sigma(k)} \cdot (d_k - a \cdot c) - b_{\sigma(k)} \cdot (a-1) + a \cdot c$, and hence the problem is solvable in $\mathrm{O}(n \log n)$ time. ◀

Next, we prove Theorem 2. We use dynamic programming to find the optimal value.

Proof of Theorem 2. Without loss of generality, we may assume that the indices of functions are $\delta(f_1) = \cdots = \delta(f_r) = -1$, $\delta(f_{r+1}) = \cdots = \delta(f_n) = 1$, $\gamma(f_1) \leq \cdots \leq \gamma(f_r)$, and $\gamma(f_{r+1}) \leq \cdots \leq \gamma(f_n)$. We use dynamic programming to solve the problem. Let $m(i,j,l)$ be the maximum value of $f_{\sigma(l)} \circ \cdots \circ f_{\sigma(1)}(c)$ for a permutation σ such that $i \leq \sigma(1) < \sigma(2) < \cdots < \sigma(l) \leq i+j-1$ if $i+j-1 \leq n$, and $i \leq \sigma(1) < \cdots < \sigma(p) \leq n,\ 1 \leq \sigma(p+1) < \cdots < \sigma(l) \leq i+j-1-n$ for some p $(0 \leq p \leq l)$ if $i+j-1 > n$. We claim that the optimal value for the problem is $\max_{i=1}^{n} m(i,n,k)$.

Let $\sigma^* : [n] \to [n]$ be an optimal permutation for the problem. By Lemmas 20 and 21, we can assume that $i^* \leq \sigma^*(1) < \cdots < \sigma^*(p) \leq n,\ 1 \leq \sigma^*(p+1) < \cdots < \sigma^*(k) \leq i^*-1$ for some i^* and p. Therefore, we have $f_{\sigma^*(k)} \circ \cdots \circ f_{\sigma^*(1)}(c) \leq m(i^*, n, k) \leq \max_{i=1}^{n} m(i,n,k)$ and thus $\max_{i=1}^{n} m(i,n,k)$ is the optimal value for the problem.

For each i, j, l, the value $m(i, j, l)$ satisfies the following relation:

$$m(i,j,l) = \begin{cases} c & (l = 0), \\ f_{\sigma(j)}(m(i, j-1, l-1)) & (l \geq 1, j = l), \\ \max\{m(i, j-1, l), f_{\sigma(j)}(m(i, j-1, l-1))\} & (l \geq 1, j > l). \end{cases}$$

To evaluate $\max_{i=1}^{n} m(i, n, k)$, our algorithm calculate the values of $m(i, j, l)$ for $0 \leq i, j \leq n$ and $0 \leq l \leq k$. Therefore, we can obtain the optimal value for the problem in $\mathrm{O}(k \cdot n^2)$ time. ◀

4 Negative Results for the Optimal Composition Ordering Problems

In the previous sections, we show that both the total and partial composition ordering problems can be solved efficiently if all f_i's are monotone linear. It turns out that they cannot be generalized to nonlinear functions f_i. In this section, we show the optimal composition ordering problems are in general intractable, even if all f_i's are monotone increasing, piecewise linear functions with at most two pieces. We remark that the maximum total composition ordering problem is known to be NP-hard, even if all f_i's are monotone increasing, *concave*, piecewise linear functions with at most two pieces [4], which can be shown by considering the time-dependent scheduling problem.

Due to space limitation, we only provide the result for the concave case.

The concave case

In this section, we consider the case in which all f_i's are monotone increasing, concave, piecewise linear functions with at most two pieces, that is, f_i is given as $f_i(x) = \min\{a_i^1 x + b_i^1,\ a_i^2 x + b_i^2\}$ for some reals a_i^1, a_i^2, b_i^1 and b_i^2 with $a_i^1, a_i^2 > 0$. For our reductions, we use the Partition problem, which is known to be NP-complete [10].

Partition: Given n positive integers $a_1, \ldots, a_n$ with $\sum_{i=1}^{n} a_i = 2T$, ask whether exists a subset $I \subseteq [n]$ such that $\sum_{i \in I} a_i = T$.

Proof for Theorem 3(i). We show that Partition can be reduced to the problem. Let $a_1, \ldots, a_n$ denote positive integers with $\sum_{i=1}^{n} a_i = 2T$. We construct $n+2$ functions f_i $(i = 1, \ldots, n+2)$ as follows:

$$f_i(x) = \begin{cases} x + a_i & \text{if } i = 1, \ldots, n, \\ \min\left\{2x,\ \frac{1}{2}x + \frac{3}{2}T\right\} & \text{if } i = n+1, \\ 6\alpha T(x - (3T - \frac{1}{2})) + (3T - \frac{1}{2}) & \text{if } i = n+2. \end{cases}$$

It is clear that all f_i's are monotone, concave, and piecewise linear with at most two pieces. Moreover, we note that all f_i's $(i = 1, \ldots, n+1)$ satisfy $f_i(x) \geq x$ if $0 \leq x \leq 3T$, and $f_{n+2}(x) \leq x$ if $x \leq 3T - 1/2$. We claim that $3T$ is the optimal value for the maximum partial (total) composition ordering problem for f_i $(i = 1, \ldots, n+1)$ and $c = 0$ if there exists a partition $I \subseteq [n]$ such that $\sum_{i \in I} a_i = T$, and the optimal value is at most $3T - 1/2$ if $\sum_{i \in I} a_i \neq T$ for any partition $I \subseteq [n]$. This implies that the optimal value for the maximum partial (total) composition ordering problem for f_i $(i = 1, \ldots, n+2)$ and $c = 0$ is at least $3\alpha T$ if $\sum_{i \in I} a_i = T$ for some $I \subseteq [n]$, and at most $3T$ if $\sum_{i \in I} a_i \neq T$ for any partition $I \subseteq [n]$, since $f_{n+2}(3T) = 3\alpha T + 3T - 1/2 > 3\alpha T$ and $f_{n+2}(x) \leq x$ if $x \leq 3T - 1/2$. Thus, there exists no α-approximation algorithm for the problems unless P=NP.

Let $\sigma : [n+1] \to [n+1]$ denote a permutation with $\sigma(l) = n+1$. Then define $I = \{\sigma(i) : i = 1, \dots, l-1\}$ and $q = \sum_{i \in I} a_i$. Note that $\sum_{i=l+1}^{n+1} a_{\sigma(i)} = \sum_{i \notin I} a_i = 2T - q$. Consider the function composition by σ:

$$\begin{aligned} f_{\sigma(n+1)} \circ \cdots \circ f_{\sigma(l+1)} \circ f_{\sigma(l)} \circ f_{\sigma(l-1)} \circ \cdots \circ f_{\sigma(1)}(0) \\ &= f_{\sigma(n)} \circ \cdots \circ f_{\sigma(l+1)} \circ f_{n+1}(q) \\ &= f_{\sigma(n)} \circ \cdots \circ f_{\sigma(l+1)} \left(\min \left\{ 2q,\ \frac{1}{2}q + \frac{3}{2}T \right\} \right) \\ &= \min \left\{ 2q,\ \frac{1}{2}q + \frac{3}{2}T \right\} + 2T - q \ = \min \left\{ q,\ -\frac{1}{2}q + \frac{3}{2}T \right\} + 2T. \end{aligned}$$

Note that $\min\left\{q,\ -\frac{1}{2}q + \frac{3}{2}T\right\} \le T$ holds, where the equality holds only when $q = T$. This implies that

$$f_{\sigma(n+1)} \circ \cdots \circ f_{\sigma(l+1)} \circ f_{\sigma(l)} \circ f_{\sigma(l-1)} \circ \cdots \circ f_{\sigma(1)}(0) \begin{cases} = 3T & (q = T), \\ \le 3T - 1/2 & (q \ne T) \end{cases} \tag{3}$$

since q is an integer. This proves the claim. ◀

References

1 M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. A knapsack secretary problem with applications. In *Proceedings of APPROX/RANDOM 2007*, pages 16–28, 2007.

2 M. Babaioff, N. Immorlica, and R. Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of SODA 2007*, pages 434–443, 2007.

3 J.-Y. Cai, P. Cai, and Y. Zhu. On a scheduling problem of time deteriorating jobs. *J. Complexity*, 14(2):190–209, 1998.

4 T. C. E. Cheng and Q. Ding. The complexity of scheduling starting time dependent tasks with release times. *Inf. Process. Lett.*, 65(2):75–79, 1998.

5 T. C. E. Cheng, Q. Ding, M. Y. Kovalyov, A. Bachman, and A. Janiak. Scheduling jobs with piecewise linear decreasing processing times. *Naval Res. Logist.*, 50(6):531–554, 2003.

6 T. C. E. Cheng, Q. Ding, and B. M. T. Lin. A concise survey of scheduling with time-dependent processing times. *EJOR*, 152(1):1–13, 2004.

7 B. Dean, M. Goemans, and J. Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proceedings of SODA 2005*, pages 395–404, 2005.

8 B. Dean, M. Goemans, and J. Vondrák. Approximating the stochastic knapsack problem: the benefit of adaptivity. *Math. of OR*, 33(4):945–964, 2008.

9 T. S. Ferguson. Who solved the secretary problem? *Statist. Sci.*, 4(3):282–289, 1989.

10 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman New York, 1979.

11 S. Gawiejnowicz. Scheduling deteriorating jobs subject to job or machine availability constraints. *EJOR*, 180(1):472–478, 2007.

12 S. Gawiejnowicz. *Time-Dependent Scheduling.* Springer, 2008.

13 S. Gawiejnowicz and L. Pankowska. Scheduling jobs with varying processing times. *Inf. Process. Lett.*, 54(3):175–178, 1995.

14 J. N. Gupta and S. K. Gupta. Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, 14(4):387–393, 1988.

15 K. I.-J. Ho, J. Y.-T. Leung, and W.-D. Wei. Complexity of scheduling tasks with time-dependent execution times. *Inf. Process. Lett.*, 48(6):315–320, 1993.

16 Y. Kawase, K. Makino, and K. Seimi. Optimal composition ordering problems for piecewise linear functions. *CoRR*, abs/1601.05480, 2016.

17 O. I. Melnikov and Y. M. Shafransky. Parametric problem of scheduling theory. *Cybernetics*, 15:352–357, 1980.

18 S. Oveis Gharan and J. Vondrák. On variants of the matroid secretary problem. In *Proceedings of ESA 2011*, pages 335–346, 2011.

19 V. S. Tanaev, V. S. Gordon, and Y. M. Shafransky. *Scheduling Theory: Single-Stage Systems*. Kluwer Academic Publishers, 1994.

20 W. Wajs. Polynomial algorithm for dynamic sequencing problem. *Archiwum Automatyki i Telemechaniki*, 31(3):209–213, 1986.

Additive Approximation Algorithms for Modularity Maximization*

Yasushi Kawase[1], **Tomomi Matsui**[2], **and Atsushi Miyauchi**[3]

1 **Tokyo Institute of Technology, Tokyo, Japan**
kawase.y.ab@m.titech.ac.jp
2 **Tokyo Institute of Technology, Tokyo, Japan**
matsui.t.af@m.titech.ac.jp
3 **Tokyo Institute of Technology, Tokyo, Japan**
miyauchi.a.aa@m.titech.ac.jp

Abstract

The modularity is a quality function in community detection, which was introduced by Newman and Girvan (2004). Community detection in graphs is now often conducted through modularity maximization: given an undirected graph $G = (V, E)$, we are asked to find a partition $\mathcal{C}$ of V that maximizes the modularity. Although numerous algorithms have been developed to date, most of them have no theoretical approximation guarantee. Recently, to overcome this issue, the design of modularity maximization algorithms with provable approximation guarantees has attracted significant attention in the computer science community.

In this study, we further investigate the approximability of modularity maximization. More specifically, we propose a polynomial-time $\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right)$-additive approximation algorithm for the modularity maximization problem. Note here that $\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8} < 0.42084$ holds. This improves the current best additive approximation error of 0.4672, which was recently provided by Dinh, Li, and Thai (2015). Interestingly, our analysis also demonstrates that the proposed algorithm obtains a nearly-optimal solution for any instance with a high modularity value. Moreover, we propose a polynomial-time 0.16598-additive approximation algorithm for the maximum modularity cut problem. It should be noted that this is the first non-trivial approximability result for the problem. Finally, we demonstrate that our approximation algorithm can be extended to some related problems.

1998 ACM Subject Classification G.2.2 [Graph Theory] Network Problems

Keywords and phrases networks, community detection, modularity maximization, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.43

1 Introduction

Identifying community structure is a fundamental primitive in graph mining [11]. Roughly speaking, a *community* (also referred to as a *cluster* or *module*) in a graph is a subset of vertices densely connected with each other, but sparsely connected with the vertices outside the subset. Community detection in graphs is a powerful way to discover components that have some special roles or possess important functions. For example, consider the

* The first author is supported by a Grant-in-Aid for Young Scientists (B) (No. 16K16005). The third author is supported by a Grant-in-Aid for JSPS Fellows (No. 26-11908).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 43; pp. 43:1–43:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graph representing the World Wide Web, where vertices correspond to web pages and edges represent hyperlinks between pages. Communities in this graph are likely to be the sets of web pages dealing with the same or similar topics, or sometimes link spam [14].

To date, numerous community detection algorithms have been developed, most of which are designed to maximize a *quality function*. Quality functions in community detection return some value that represents the *community-degree* for a given partition of the set of vertices. The best known and widely used quality function is the *modularity*, which was introduced by Newman and Girvan [24]. Let $G = (V, E)$ be an undirected graph consisting of $n = |V|$ vertices and $m = |E|$ edges. The modularity, a quality function for a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V (i.e., $\bigcup_{i=1}^{k} C_i = V$ and $C_i \cap C_j = \emptyset$ for $i \neq j$), can be written as

$$Q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \left(\frac{m_C}{m} - \left(\frac{D_C}{2m} \right)^2 \right),$$

where m_C represents the number of edges whose endpoints are both in C, and D_C represents the sum of degrees of the vertices in C. The modularity represents the sum, over all communities, of the fraction of the number of edges within communities minus the expected fraction of such edges assuming that they are placed at random with the same degree distribution.

Although the modularity is known to have some drawbacks (e.g., the *resolution limit* [12]), community detection is now often conducted through modularity maximization: given an undirected graph $G = (V, E)$, we are asked to find a partition $\mathcal{C}$ of V that maximizes the modularity. Note that the modularity maximization problem has no restriction on the number of communities in the output partition; thus, the algorithms are allowed to specify the best number of communities by themselves. Brandes et al. [5] proved that modularity maximization is NP-hard. A wide variety of applications (and this hardness result) have promoted the development of modularity maximization heuristics. In fact, there are numerous algorithms based on various techniques such as greedy procedure [4, 24], simulated annealing [17], spectral optimization [23], and mathematical programming [1, 20, 6]. Although some of them are known to perform well in practice, they have no theoretical approximation guarantee at all.

Recently, to overcome this issue, the design of modularity maximization algorithms with provable approximation guarantees has attracted significant attention in the computer science community. DasGupta and Desai [8] designed a polynomial-time ϵ-additive approximation algorithm[1] for dense graphs (i.e., graphs with $m = \Omega(n^2)$) using an algorithmic version of the regularity lemma [13], where $\epsilon > 0$ is an arbitrary constant. Moreover, Dinh, Li, and Thai [9] very recently developed a polynomial-time 0.4672-additive approximation algorithm. This is the first polynomial-time additive approximation algorithm with a non-trivial approximation guarantee (that is applicable to any instance).[2] Note that, to our knowledge, this is the current best additive approximation error. Their algorithm is based on the semidefinite programming (SDP) relaxation and the hyperplane separation technique.

[1] A feasible solution is *α-additive approximate* if its objective value is at least the optimal value minus α. An algorithm is called an *α-additive approximation algorithm* if it returns an α-additive approximate solution for any instance. For an α-additive approximation algorithm, α is referred to as an *additive approximation error* of the algorithm.

[2] A 1-additive approximation algorithm is trivial because $Q(\{V\}) = 0$ and $Q(\mathcal{C}) < 1$ for any partition $\mathcal{C}$.

1.1 Our Contribution

In this study, we further investigate the approximability of modularity maximization. Our contribution can be summarized as follows:

1. We propose a polynomial-time $\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right)$-additive approximation algorithm for the modularity maximization problem. Note here that $\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8} < 0.42084$ holds; thus, this improves the current best additive approximation error of 0.4672, which was recently provided by Dinh, Li, and Thai [9]. Interestingly, our analysis also demonstrates that the proposed algorithm obtains a nearly-optimal solution for any instance with a high modularity value.
2. We propose a polynomial-time 0.16598-additive approximation algorithm for the maximum modularity cut problem. It should be noted that this is the first non-trivial approximability result for the problem.
3. We demonstrate that our additive approximation algorithm for the modularity maximization problem can be extended to some related problems.

First result

Let us describe our first result in details. Our additive approximation algorithm is also based on the SDP relaxation and the hyperplane separation technique. However, our analysis is essentially different from the one by Dinh, Li, and Thai [9], and is much more effective for many practical instances.

The algorithm by Dinh, Li, and Thai [9] reduces the SDP relaxation for the modularity maximization problem to the one for MaxAgree problem arising in correlation clustering (e.g., see [2] or [7]) by adding an appropriate constant to the objective function. Then, the algorithm adopts the SDP-based 0.7664-approximation algorithm[3] for MaxAgree problem [7]. Specifically, their algorithm generates 2 and 3 random hyperplanes to obtain feasible solutions, and then returns the better one. Their analysis of the additive approximation guarantee depends heavily on the above reduction; the additive approximation error of 0.4672 is just derived from $2(1-\kappa)$, where κ represents the approximation ratio of the SDP-based algorithm for MaxAgree problem (i.e., $\kappa = 0.7664$). The analysis of the SDP-based algorithm for MaxAgree problem [7] aims at multiplicative approximation rather than additive one. As a result, the analysis by Dinh, Li, and Thai [9] has caused a gap in terms of additive approximation. In fact, as shown in our analysis, their algorithm already has the approximation error of $\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}$ (< 0.42084).

In contrast, our algorithm and analysis do not depend on such a reduction. In fact, our algorithm just solves the SDP relaxation for the modularity maximization problem without any transformation. Moreover, our algorithm employs a hyperplane separation procedure that extends the one used in their algorithm. Specifically, our algorithm chooses an appropriate number of hyperplanes using the information of the optimal solution to the SDP relaxation so that the lower bound on the expected modularity value is maximized. It should be emphasized that our analysis directly evaluates an additive approximation error of the proposed algorithm, unlike the analysis by Dinh, Li, and Thai [9]. As a result, our analysis improves their additive approximation error, and demonstrates that the proposed

[3] A feasible solution is *α-approximate* if its objective value is at least α times the optimal value. An algorithm is called an *α-approximation algorithm* if it returns an α-approximate solution for any instance. For an α-approximation algorithm, α is referred to as an *approximation ratio* of the algorithm.

algorithm has a much better lower bound on the expected modularity value for many practical instances. In particular, for any instance with optimal value close to 1 (a trivial upper bound), our algorithm obtains a nearly-optimal solution. Note here that, as reported in previous work [4, 6, 25], there are many large real-world networks that have a partition with a very high modularity value. At the end of our analysis, we summarize a lower bound on the expected modularity value with respect to the optimal value of a given instance.

Second result

Here we describe our second result in details. The modularity maximization problem has no restriction on the number of clusters in the output partition. On the other hand, there also exist a number of problem variants with such a restriction. The maximum modularity cut problem is a typical one, where given an undirected graph $G = (V, E)$, we are asked to find a partition $\mathcal{C}$ of V consisting of at most two components (i.e., a bipartition $\mathcal{C}$ of V) that maximizes the modularity. This problem appears in many contexts in community detection. For example, a few hierarchical divisive heuristics for the modularity maximization problem repeatedly solve this problem either exactly [6] or heuristically [1], to obtain a partition $\mathcal{C}$ of V. Brandes et al. [5] proved that the maximum modularity cut problem is NP-hard (even on dense graphs). More recently, DasGupta and Desai [8] showed that the problem is NP-hard even on d-regular graphs with any fixed $d \geq 9$. However, to our knowledge, there exists no approximability result for the problem.

Our additive approximation algorithm adopts the SDP relaxation and the hyperplane separation technique, which is identical to the subroutine of the hierarchical divisive heuristic proposed by Agarwal and Kempe [1]. Specifically, our algorithm first solves the SDP relaxation for the maximum modularity cut problem (rather than the modularity maximization problem), and then generates a random hyperplane to obtain a feasible solution for the problem. Although the computational experiments by Agarwal and Kempe [1] demonstrate that their hierarchical divisive heuristic maximizes the modularity quite well in practice, the approximation guarantee of the subroutine in terms of the maximum modularity cut was not analyzed. Our analysis shows that the proposed algorithm is a 0.16598-additive approximation algorithm for the maximum modularity cut problem. At the end of our analysis, we again present a lower bound on the expected modularity value with respect to the optimal value of a given instance. This reveals that for any instance with optimal value close to 1/2 (a trivial upper bound in the case of bipartition), our algorithm obtains a nearly-optimal solution.

Third result

Finally, we describe our third result. We first extend our additive approximation algorithm for the modularity maximization problem to the well-known graph partitioning problem called the clique partitioning problem [16]. Then, we apply the result for the following three special cases of the clique partitioning problem: the weighted modularity maximization problem [22], the directed modularity maximization problem [19], and Barber's bipartite modularity maximization problem [3], all of which are NP-hard (see [8] and [21]).

1.2 Related Work

Multiplicative approximation algorithms

There also exist multiplicative approximation algorithms for modularity maximization. DasGupta and Desai [8] designed an $\Omega(1/\log d)$-approximation algorithm for the modularity maximization problem on d-regular graphs with $d \leq \frac{n}{2\log n}$. They extended the approximation algorithm to the weighted modularity maximization problem. On the other hand, Dinh and Thai [10] developed algorithms for scale-free graphs with a prescribed degree sequence. In their graphs, the number of vertices with degree d is fixed to some value proportional to $d^{-\gamma}$, where $-\gamma$ is called the power-law exponent.

Inapproximability results

There are some inapproximability results for the modularity maximization problem. DasGupta and Desai [8] showed that it is NP-hard to obtain a $(1-\epsilon)$-approximate solution for some constant $\epsilon > 0$ (even for complements of 3-regular graphs). More recently, Dinh, Li, and Thai [9] proved a much stronger statement, that is, there exists no polynomial-time $(1-\epsilon)$-approximation algorithm for *any* $\epsilon > 0$, unless P = NP. It should be noted that these results are on multiplicative approximation rather than additive one. In fact, there exist no inapproximability results in terms of additive approximation for modularity maximization.

1.3 Preliminaries

Here we introduce definitions and notation used in this paper. Let $G = (V, E)$ be an undirected graph consisting of $n = |V|$ vertices and $m = |E|$ edges. Let $P = V \times V$. By simple calculation, as mentioned in Brandes et al. [5], the modularity can be rewritten as

$$Q(\mathcal{C}) = \frac{1}{2m} \sum_{(i,j)\in P} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \delta(\mathcal{C}(i), \mathcal{C}(j)),$$

where A_{ij} is the (i, j) component of the adjacency matrix A of G, d_i is the degree of $i \in V$, $\mathcal{C}(i)$ is the (unique) community to which $i \in V$ belongs, and δ is the Kronecker symbol equal to 1 if two arguments are identical and 0 otherwise. This form is useful to write mathematical programming formulations for modularity maximization. For convenience, we define

$$q_{ij} = \frac{A_{ij}}{2m} - \frac{d_i d_j}{4m^2} \quad \text{for each } (i, j) \in P.$$

We can divide the set P into the following two disjoint subsets: $P_{\geq 0} = \{(i, j) \in P \mid q_{ij} \geq 0\}$ and $P_{<0} = \{(i, j) \in P \mid q_{ij} < 0\}$. Clearly, we have $\sum_{(i,j)\in P_{\geq 0}} q_{ij} + \sum_{(i,j)\in P_{<0}} q_{ij} = \sum_{(i,j)\in P} q_{ij} = 0$, and thus $\sum_{(i,j)\in P_{\geq 0}} q_{ij} = \sum_{(i,j)\in P_{<0}} -q_{ij}$. We denote this value by q, i.e., $q = \sum_{(i,j)\in P_{\geq 0}} q_{ij}$. Note that for any instance, we have $q < 1$.

1.4 Paper Organization

In Section 2, we revisit the SDP relaxation for the modularity maximization problem, and then describe an outline of our algorithm. In Section 3, the approximation guarantee of the proposed algorithm is carefully analyzed. In Section 4, we present our additive approximation algorithm for the maximum modularity cut problem. We mention the extension of our additive approximation algorithm to some related problems in Section 5. Due to space limitations, some proofs are omitted, which can be found in the full version [18].

2 Algorithm

The modularity maximization problem can be formulated as follows:

$$\text{max.} \sum_{(i,j)\in P} q_{ij} \left(\boldsymbol{y}_i \cdot \boldsymbol{y}_j \right) \quad \text{s.t.} \quad \boldsymbol{y}_i \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\} \ (\forall i \in V),$$

```
Algorithm 1 Hyperplane($k$)
Input: Graph $G = (V, E)$
Output: Partition $\mathcal{C}$ of $V$
1: Obtain an optimal solution $X^* = (x^*_{ij})$ to SDP
2: Generate $k$ random hyperplanes and obtain a partition $\mathcal{C}_k = \{C_1, \ldots, C_{2^k}\}$ of $V$
3: return $\mathcal{C}_k$
```

where $\boldsymbol{e}_k$ $(1 \leq k \leq n)$ represents the vector that has 1 in the kth coordinate and 0 elsewhere. We denote by OPT the optimal value of this original problem. Note that for any instance, we have $\mathsf{OPT} \in [0, 1)$. We introduce the following semidefinite relaxation problem:

$$\mathsf{SDP}: \ \text{max.} \sum_{(i,j)\in P} q_{ij}x_{ij} \quad \text{s.t.} \ \ x_{ii} = 1 \ (\forall i \in V), \ \ x_{ij} \geq 0 \ (\forall i, j \in V), \ \ X = (x_{ij}) \in \mathcal{S}^n_+,$$

where $\mathcal{S}^n_+$ represents the cone of $n \times n$ symmetric positive semidefinite matrices. It is easy to see that every feasible solution $X = (x_{ij})$ of SDP satisfies $x_{ij} \leq 1$ for any $(i, j) \in P$. Although the algorithm by Dinh, Li, and Thai [9] reduces SDP to the one for MaxAgree problem by adding an appropriate constant to the objective function, our algorithm just solves SDP without any transformation. Let $X^* = (x^*_{ij})$ be an optimal solution to SDP, which can be computed (with an arbitrarily small error) in time polynomial in n and m. Using the optimal solution X^*, we define the following two values:

$$z^*_+ = \frac{1}{q} \sum_{(i,j)\in P_{\geq 0}} q_{ij}x^*_{ij} \quad \text{and} \quad z^*_- = \frac{1}{q} \sum_{(i,j)\in P_{<0}} q_{ij}x^*_{ij},$$

both of which are useful in the analysis of the approximation guarantee of our algorithm. Clearly, we have $0 \leq z^*_+ \leq 1$ and $-1 \leq z^*_- \leq 0$.

We apply the hyperplane separation technique to obtain a feasible solution of the modularity maximization problem. Specifically, we consider the following general procedure: generate k random hyperplanes to separate the vectors corresponding to the optimal solution X^*, and then obtain a partition $\mathcal{C}_k = \{C_1, \ldots, C_{2^k}\}$ of V. For reference, the procedure is described in Algorithm 1. Note here that at this time, we have not yet mentioned how to determine the number k of hyperplanes we generate. As revealed in our analysis, we can choose an appropriate number of hyperplanes using the value of z^*_+ so that the lower bound on the expected modularity value of the output of Hyperplane(k) is maximized.

3 Analysis

In this section, we first analyze an additive approximation error of Hyperplane(k) for each positive integer $k \in \mathbb{Z}_{>0}$. Then, we provide an appropriate number $k^* \in \mathbb{Z}_{>0}$ of hyperplanes, which completes the design of our algorithm. Finally, we present a lower bound on the expected modularity value of the output of Hyperplane(k^*) with respect to the value of OPT.

When k random hyperplanes are generated independently, the probability that two vertices $i, j \in V$ are in the same cluster is given by $\left(1 - \arccos(x^*_{ij})/\pi\right)^k$, as mentioned in previous works (e.g., see [7] or [15]). For simplicity, we define the function $f_k(x) = (1 - \arccos(x)/\pi)^k$ for $x \in [0, 1]$. Here we present the lower convex envelope of each of $f_k(x)$ and $-f_k(x)$.

▶ **Lemma 1.** *For any positive integer k, the lower convex envelope of $f_k(x)$ is given by $f_k(x)$ itself, and the lower convex envelope of $-f_k(x)$ is given by the linear function $h_k(x) = -1/2^k + (1/2^k - 1)x$ for $x \in [0, 1]$.*

The following lemma lower bounds the expected modularity value of the output of Hyperplane(k).

▶ **Lemma 2.** *Let $\mathcal{C}_k$ be the output of* Hyperplane(k). *For any positive integer k, it holds that*

$$\mathrm{E}[Q(\mathcal{C}_k)] \geq q\left(f_k(z_+^*) + h_k(-z_-^*)\right).$$

Proof. Recall that $\mathcal{C}_k(i)$ for each $i \in V$ denotes the (unique) cluster in $\mathcal{C}_k$ that includes the vertex i. Note here that $\delta(\mathcal{C}_k(i), \mathcal{C}_k(j))$ for each $(i,j) \in P$ is a random variable, which takes 1 with probability $f_k(x_{ij}^*)$ and 0 with probability $1 - f_k(x_{ij}^*)$. The expectation $\mathrm{E}[Q(\mathcal{C}_k)]$ can be transformed as follows:

$$\begin{aligned}\mathrm{E}[Q(\mathcal{C}_k)] &= \mathrm{E}\left[\sum_{(i,j)\in P} q_{ij}\delta(\mathcal{C}_k(i), \mathcal{C}_k(j))\right] \\ &= \sum_{(i,j)\in P} q_{ij} f_k(x_{ij}^*) = \sum_{(i,j)\in P_{\geq 0}} q_{ij} f_k(x_{ij}^*) + \sum_{(i,j)\in P_{<0}} -q_{ij}\cdot(-f_k(x_{ij}^*)).\end{aligned}$$

Using Lemma 1, we have

$$\begin{aligned}\mathrm{E}[Q(\mathcal{C}_k)] &\geq \sum_{(i,j)\in P_{\geq 0}} q_{ij} f_k(x_{ij}^*) + \sum_{(i,j)\in P_{<0}} -q_{ij} h_k(x_{ij}^*) \\ &= q\left(\sum_{(i,j)\in P_{\geq 0}} \left(\frac{q_{ij}}{q}\right) f_k(x_{ij}^*) + \sum_{(i,j)\in P_{<0}} \left(\frac{-q_{ij}}{q}\right) h_k(x_{ij}^*)\right) \\ &\geq q\left(f_k\left(\sum_{(i,j)\in P_{\geq 0}} \left(\frac{q_{ij}}{q}\right) x_{ij}^*\right) + h_k\left(\sum_{(i,j)\in P_{<0}} \left(\frac{-q_{ij}}{q}\right) x_{ij}^*\right)\right) \\ &= q\left(f_k(z_+^*) + h_k(-z_-^*)\right),\end{aligned}$$

where the last inequality follows from Jensen's inequality. ◀

The following lemma provides an additive approximation error of Hyperplane(k) by evaluating the above lower bound on $\mathrm{E}[Q(\mathcal{C}_k)]$ using the value of OPT.

▶ **Lemma 3.** *For any positive integer k, it holds that*

$$\mathrm{E}[Q(\mathcal{C}_k)] \geq \mathsf{OPT} - q\left(z_+^* - f_k(z_+^*) + \frac{1}{2^k}\right).$$

Proof. Clearly, (z_+^*, z_-^*) satisfies $q(z_+^* + z_-^*) \geq \mathsf{OPT}$ and $z_-^* \leq 0$. Thus, we obtain

$$\begin{aligned}q\left(f_k(z_+^*) + h_k(-z_-^*)\right) &\geq \left(\mathsf{OPT} - q(z_+^* + z_-^*)\right) + q\left(f_k(z_+^*) + h_k(-z_-^*)\right) \\ &= \left(\mathsf{OPT} - q(z_+^* + z_-^*)\right) + q\left(f_k(z_+^*) - 1/2^k + (1/2^k - 1)(-z_-^*)\right) \\ &= \mathsf{OPT} - q\left(z_+^* - f_k(z_+^*) + 1/2^k + (1/2^k)z_-^*\right) \\ &\geq \mathsf{OPT} - q\left(z_+^* - f_k(z_+^*) + 1/2^k\right).\end{aligned}$$

Combining this with Lemma 2, we have $\mathrm{E}[Q(\mathcal{C}_k)] \geq q\left(f_k(z_+^*) + h_k(-z_-^*)\right) \geq \mathsf{OPT} - q\left(z_+^* - f_k(z_+^*) + 1/2^k\right)$, as desired. ◀

For simplicity, we define the function $g_k(x) = x - f_k(x) + 1/2^k$ for $x \in [0,1]$. Then, the inequality of the above lemma can be rewritten as

$$\mathrm{E}[Q(\mathcal{C}_k)] \geq \mathsf{OPT} - q \cdot g_k(z_+^*).$$

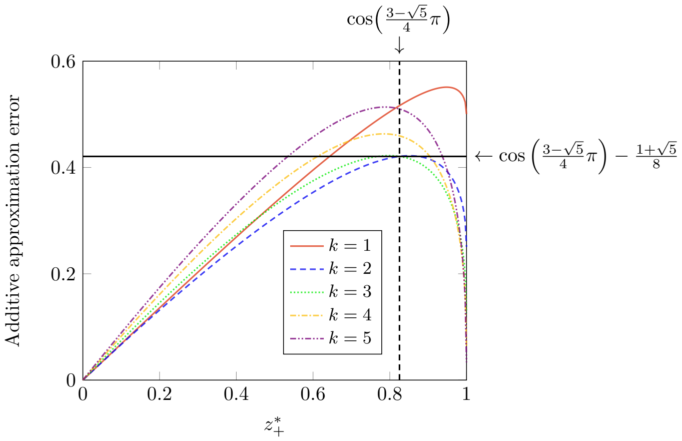


Figure 1 A brief illustration of the additive approximation error of Hyperplane(k) with respect to the value of z_+^*. For simplicity, we replace q by its upper bound 1. Specifically, the function $g_k(x) = x - f_k(x) + 1/2^k$ for $x \in [0,1]$ is plotted for $k = 1,2,3,4$, and 5, as examples. The point $\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right), \cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right)$ is an intersection of the functions $g_2(x)$ and $g_3(x)$.

Figure 1 plots the above additive approximation error of Hyperplane(k) with respect to the value of z_+^*.

As can be seen, the appropriate number of hyperplanes (i.e., the number of hyperplanes that minimizes the additive approximation error) depends on the value of z_+^*. Intuitively, we wish to choose k^{**} that satisfies $k^{**} \in \arg\min_{k\in\mathbb{Z}_{>0}} g_k(z_+^*)$. However, it is not clear whether Hyperplane(k^{**}) runs in polynomial time. In fact, the number k^{**} becomes infinity if the value of z_+^* approaches 1. Therefore, alternatively, our algorithm chooses

$$k^* \in \underset{k\in\{1,\dots,\max\{3,\lceil\log_2 n\rceil\}\}}{\arg\min} g_k(z_+^*).$$

First, we analyze the worst-case performance of Hyperplane(k^*). The following lemma says that (i) the worst-case performance of Hyperplane(k^*) is exactly the same as that of Hyperplane(k^{**}); and moreover (ii) to achieve the same worst-case performance as that of Hyperplane(k^{**}), it suffices to choose k from the set $\{2,3\}$.

▶ **Lemma 4.** *Let S be one of the sets $\{2,3\}$, $\{1,\dots,\max\{3,\lceil\log_2 n\rceil\}\}$, and $\mathbb{Z}_{>0}$. It holds that*

$$\max_{x\in[0,1]} \min_{k\in S} g_k(x) = \cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}.$$

▶ Remark. Here we consider the algorithm that executes Hyperplane(2) and Hyperplane(3), and then returns the better solution. Note that this algorithm is essentially the same as that proposed by Dinh, Li, and Thai [9]. The above lemma implies that the algorithm by Dinh, Li, and Thai [9] already has the worst-case performance exactly the same as that of Hyperplane(k^*) (and Hyperplane(k^{**})). However, as shown below, Hyperplane(k^*) has a much better lower bound on the expected modularity value for many instances.

Finally, we present a lower bound on the expected modularity value of the output of Hyperplane(k^*) with respect to the value of OPT (rather than z_+^*). The following lemma is useful to show that the lower bound on the expected modularity value with respect to the value of OPT is not affected by the change from k^{**} to k^*.

▶ **Lemma 5.** *For any $k' \in \arg\min_{k\in\mathbb{Z}_{>0}} g_k(\mathsf{OPT})$, it holds that $k' \leq \max\{3, \lceil \log_2 n\rceil\}$.*

We are now ready to prove the main result of this paper.

▶ **Theorem 6.** *Let $\mathcal{C}_{k^*}$ be the output of* Hyperplane(k^*)*. It holds that*

$$\mathrm{E}[Q(\mathcal{C}_{k^*})] \geq \mathsf{OPT} - q\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right).$$

In particular, if $\mathsf{OPT} \geq \cos\left(\frac{3-\sqrt{5}}{4}\pi\right)$ holds, then $\mathrm{E}[Q(\mathcal{C}_{k^*})] > \mathsf{OPT} - q\min_{k\in\mathbb{Z}_{>0}} g_k(\mathsf{OPT})$. Note here that $q < 1$ and $\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8} < 0.42084$.

Proof. From Lemmas 3 and 4, it follows directly that

$$\mathrm{E}[Q(\mathcal{C}_{k^*})] \geq \mathsf{OPT} - q\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right).$$

Here we prove the remaining part of the theorem. Assume that $\mathsf{OPT} \geq \cos\left(\frac{3-\sqrt{5}}{4}\pi\right)$ holds. By simple calculation, for any $k \in \mathbb{Z}_{>0}$, we have $g_k''(x) < 0$ for $x \in (0,1)$. This means that for any $k \in \mathbb{Z}_{>0}$, the function $g_k(x)$ is strictly concave, and moreover, so is the function $\min_{k\in\{1,\ldots,\max\{3,\lceil\log_2 n\rceil\}\}} g_k(x)$. From Lemma 4 and the fact that $\min_{k\in\{1,\ldots,\max\{3,\lceil\log_2 n\rceil\}\}} g_k\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right)\right) = \cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}$ holds, we see that the function $\min_{k\in\{1,\ldots,\max\{3,\lceil\log_2 n\rceil\}\}} g_k(x)$ attains its maximum at $x = \cos\left(\frac{3-\sqrt{5}}{4}\pi\right)$. Thus, the function $\min_{k\in\{1,\ldots,\max\{3,\lceil\log_2 n\rceil\}\}} g_k(x)$ is strictly monotonically decreasing over the interval $\left[\cos\left(\frac{3-\sqrt{5}}{4}\pi\right), 1\right]$. Therefore, we have

$$\begin{aligned}\mathrm{E}[Q(\mathcal{C}_{k^*})] &\geq \mathsf{OPT} - q\min_{k\in\{1,\ldots,\max\{3,\lceil\log_2 n\rceil\}\}} g_k(z_+^*)\\ &> \mathsf{OPT} - q\min_{k\in\{1,\ldots,\max\{3,\lceil\log_2 n\rceil\}\}} g_k(\mathsf{OPT}) = \mathsf{OPT} - q\min_{k\in\mathbb{Z}_{>0}} g_k(\mathsf{OPT}),\end{aligned}$$

where the second inequality follows from $z_+^* \geq \mathsf{OPT}/q > \mathsf{OPT}$ and the last equality follows from Lemma 5. ◀

Figure 2 depicts the above lower bound on $\mathrm{E}[Q(\mathcal{C}_{k^*})]$. As can be seen, if OPT is close to 1, then Hyperplane(k^*) obtains a nearly-optimal solution. For example, for any instance with $\mathsf{OPT} \geq 0.99990$, it holds that $\mathrm{E}[Q(\mathcal{C}_{k^*})] > 0.96109$, i.e., the additive approximation error is less than 0.03891. For such instances, the reduction-based analysis by Dinh, Li, and Thai [9] provides no guarantee better than the worst-case performance.

▶ Remark. The additive approximation error of Hyperplane(k^*) depends on the value of $q < 1$; the less the value of q, the better the additive approximation error. Thus, it is interesting to find some graphs that have a small value of q. For instance, for any regular graph G that satisfies $m = \frac{\alpha}{2}n^2$, it holds that $q = 1-\alpha$, where α is an arbitrary constant in $(0,1)$. Here we prove the statement. Since G is regular, we have $d_i = 2m/n = \alpha n$ for any $i \in V$. Hence, for any $\{i,j\} \in E$, it holds that $q_{ij} = \frac{A_{ij}}{2m} - \frac{d_i d_j}{4m^2} = \frac{1}{\alpha n^2} - \frac{1}{n^2} > 0$. Therefore, we have

$$q = \sum_{(i,j)\in P_{\geq 0}} q_{ij} = 2\sum_{\{i,j\}\in E} q_{ij} = 2m\left(\frac{1}{\alpha n^2} - \frac{1}{n^2}\right) = 1-\alpha.$$

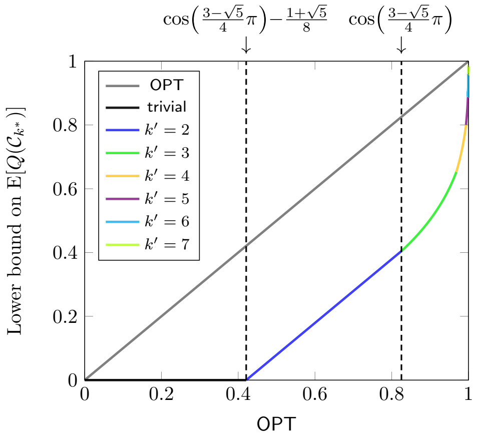


Figure 2 A brief illustration of the lower bound on the expected modularity value of the output of Hyperplane(k^*) with respect to the value of OPT. For simplicity, we replace q by its upper bound 1. Note that $k' \in \arg\min_{k\in\mathbb{Z}_{>0}} g_k(\mathsf{OPT})$.

Algorithm 2 Modularity Cut

```
Input: Graph $G = (V, E)$
Output: Bipartition $\mathcal{C}$ of $V$
Obtain an optimal solution $X^* = (x^*_{ij})$ to $\mathsf{SDP}_{\mathsf{cut}}$
Generate a random hyperplane and obtain a bipartition $\mathcal{C}_{\text{out}} = \{C_1, C_2\}$ of $V$
return $\mathcal{C}_{\text{out}}$
```

4 Maximum Modularity Cut

The maximum modularity cut problem can be formulated as follows:

$$\text{max.} \quad \frac{1}{2}\sum_{(i,j)\in P} q_{ij}(y_i y_j + 1) \quad \text{s.t.} \quad y_i \in \{-1, 1\} \ (\forall i \in V).$$

We denote by $\mathsf{OPT}_{\mathsf{cut}}$ the optimal value of this original problem. Note that for any instance, it holds that $\mathsf{OPT}_{\mathsf{cut}} \in [0, 1/2]$, as shown in DasGupta and Desai [8]. We introduce the following semidefinite relaxation problem:

$$\mathsf{SDP}_{\mathsf{cut}}: \ \text{max.} \quad \frac{1}{2}\sum_{(i,j)\in P} q_{ij}(x_{ij} + 1) \quad \text{s.t.} \quad x_{ii} = 1 \ (\forall i \in V), \quad X = (x_{ij}) \in \mathcal{S}^n_+,$$

where recall that $\mathcal{S}^n_+$ represents the cone of $n \times n$ symmetric positive semidefinite matrices. Let $X^* = (x^*_{ij})$ be an optimal solution to $\mathsf{SDP}_{\mathsf{cut}}$, which can be computed (with an arbitrarily small error) in time polynomial in n and m. Note here that x^*_{ij} may be negative for $(i, j) \in P$ with $i \neq j$, unlike SDP in the previous section.

We generate a random hyperplane to separate the vectors corresponding to the optimal solution X^*, and then obtain a bipartition $\mathcal{C} = \{C_1, C_2\}$ of V. For reference, the procedure is described in Algorithm 2. As mentioned above, this algorithm is identical to the subroutine of the hierarchical divisive heuristic for the modularity maximization problem, which was proposed by Agarwal and Kempe [1].

The main result of this section is the following theorem.

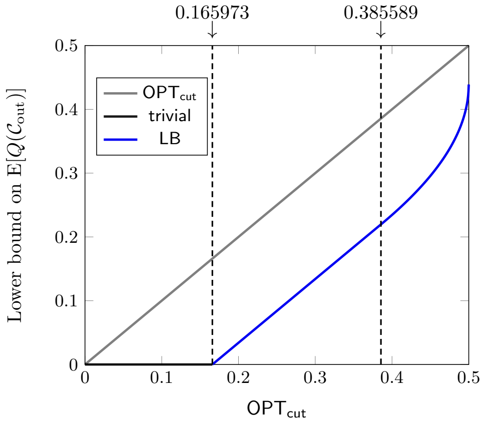


Figure 3 An illustration of the lower bound on the expected modularity value of the output of Algorithm 2.

▶ **Theorem 7.** *Let* $\alpha = \min_{-1<x<1} \frac{1-\arccos(x)/\pi}{(x+1)/2}$ $(\simeq 0.878567)$*. Let* $\mathcal{C}_{\mathrm{out}}$ *be the output of Algorithm 2. It holds that*

$$\mathrm{E}[Q(\mathcal{C}_{\mathrm{out}})] > \mathsf{OPT}_{\mathsf{cut}} - 0.16598.$$

In particular, if $\mathsf{OPT}_{\mathsf{cut}} \geq \frac{\sqrt{\pi^2-4}}{2\pi}$ $(\simeq 0.385589)$ *holds, then* $\mathrm{E}[Q(\mathcal{C}_{\mathrm{out}})] \geq \frac{\alpha}{2} - \frac{\arccos(2\cdot\mathsf{OPT}_{\mathsf{cut}})}{\pi}$.

Figure 3 depicts the above lower bound on $\mathrm{E}[Q(\mathcal{C}_{\mathrm{out}})]$. As can be seen, if $\mathsf{OPT}_{\mathsf{cut}}$ is close to 1/2, then Algorithm 2 obtains a nearly-optimal solution.

5 Extension

We first extend our additive approximation algorithm for the modularity maximization problem to the clique partitioning problem. In the clique partitioning problem, we are given a finite set V and a weight function $c : V \times V \to \mathbb{R}$. Let $P = V \times V$. The aim is to find a partition $\mathcal{C}$ of V that maximizes the sum of weights of the pairs within the same clusters, i.e.,

$$Q_{\mathsf{CPP}}(\mathcal{C}) = \sum_{(i,j)\in P} c_{ij}\delta(\mathcal{C}(i), \mathcal{C}(j)).$$

Although our definition is slightly different from the traditional one (see e.g., [16]), it remains essentially the same. Clearly, the modularity maximization problem can be reduced to the clique partitioning problem. In fact, it suffices to set $c_{ij} = q_{ij}$ for each $(i,j) \in P$.

We can extend $\mathsf{Hyperplane}(k)$ to the clique partitioning problem by replacing q_{ij} with c_{ij} in the description of the algorithm. Furthermore, we can also extend our analysis of the additive approximation error of $\mathsf{Hyperplane}(k)$. Note here that we should redefine

$$z_+^* = \frac{1}{c_+} \sum_{(i,j)\in P_{\geq 0}} c_{ij}x_{ij}^* \quad \text{and} \quad z_-^* = \frac{1}{c_-} \sum_{(i,j)\in P_{<0}} c_{ij}x_{ij}^*,$$

where $c_+ = \sum_{(i,j)\in P_{\geq 0}} c_{ij}$ and $c_- = \sum_{(i,j)\in P_{<0}} c_{ij}$. This is due to the fact that $c_+ = c_-$ does not necessarily hold. For the clique partitioning problem, we have the following key lemma, which is a generalization of Lemma 3.

▶ **Lemma 8.** *Let $\mathcal{C}_k$ be the output of* Hyperplane(k). *It holds that*

$$\mathrm{E}[Q_{\mathsf{CPP}}(\mathcal{C}_k)] \geq \mathsf{OPT} - \left(c_+ z_+^* - c_+ f_k(z_+^*) + c_- \frac{1}{2^k}\right).$$

Using this lemma, we can choose an appropriate number k^* of hyperplanes and analyze the additive approximation error of Hyperplane(k^*). Note that if $c_+ = c_-$ holds, then we obtain the additive approximation error of $c_+ \left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right)$.

Finally, we mention the results for the following three problems: the weighted modularity maximization problem [22], the directed modularity maximization problem [19], and Barber's bipartite modularity maximization problem [3]. These problems are all special cases of the clique partitioning problem, where $c_+ = c_- < 1$ holds. For the detailed description of the above problems, see the full version [18]. We have the following corollary.

▶ **Corollary 9.** *There exist polynomial-time* $\left(\cos\left(\frac{3-\sqrt{5}}{4}\pi\right) - \frac{1+\sqrt{5}}{8}\right)$*-additive approximation algorithms for the weighted modularity maximization problem, the directed modularity maximization problem, and Barber's bipartite modularity maximization problem.*

References

1 G. Agarwal and D. Kempe. Modularity-maximizing graph communities via mathematical programming. *Eur. Phys. J. B*, 66(3):409–418, 2008.

2 N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004.

3 M. J. Barber. Modularity and community detection in bipartite networks. *Phys. Rev. E*, 76:066102, 2007.

4 V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech.: Theory Exp.*, 2008:P10008, 2008.

5 U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *IEEE Trans. Knowl. Data Eng.*, 20(2):172–188, 2008.

6 S. Cafieri, A. Costa, and P. Hansen. Reformulation of a model for hierarchical divisive graph modularity maximization. *Ann. Oper. Res.*, 222(1):213–226, 2014.

7 M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

8 B. DasGupta and D. Desai. On the complexity of Newman's community finding approach for biological and social networks. *J. Comput. Syst. Sci.*, 79(1):50–67, 2013.

9 T. N. Dinh, X. Li, and M. T. Thai. Network clustering via maximizing modularity: Approximation algorithms and theoretical limits. In *ICDM'15*, pages 101–110, 2015.

10 T. N. Dinh and M. T. Thai. Community detection in scale-free networks: Approximation algorithms for maximizing modularity. *IEEE J. Sel. Areas Commun.*, 31(6):997–1006, 2013.

11 S. Fortunato. Community detection in graphs. *Phys. Rep.*, 486(3):75–174, 2010.

12 S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proc. Natl. Acad. Sci. U.S.A.*, 104(1):36–41, 2007.

13 A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *FOCS'96*, pages 12–20, 1996.

14 D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB'05*, pages 721–732, 2005.

15 M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

16 M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Math. Program.*, 45(1-3):59–96, 1989.

17 R. Guimerà and L. A. N. Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005.

18 Yasushi Kawase, Tomomi Matsui, and Atsushi Miyauchi. Additive approximation algorithms for modularity maximization. *arXiv:1601.03316*, 2016.

19 E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Phys. Rev. Lett.*, 100:118703, 2008.

20 A. Miyauchi and Y. Miyamoto. Computing an upper bound of modularity. *Eur. Phys. J. B*, 86:302, 2013.

21 A. Miyauchi and N. Sukegawa. Maximizing Barber's bipartite modularity is also hard. *Optim. Lett.*, 9(5):897–913, 2015.

22 M. E. J. Newman. Analysis of weighted networks. *Phys. Rev. E*, 70:056131, 2004.

23 M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. U.S.A.*, 103(23):8577–8582, 2006.

24 M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, 2004.

25 L. Waltman and N. J. van Eck. A smart local moving algorithm for large-scale modularity-based community detection. *Eur. Phys. J. B*, 86(11):1–14, 2013.

The Densest Subgraph Problem with a Convex/Concave Size Function*

Yasushi Kawase[1] and Atsushi Miyauchi[2]

1 **Tokyo Institute of Technology, Tokyo, Japan**
`kawase.y.ab@m.titech.ac.jp`

2 **Tokyo Institute of Technology, Tokyo, Japan**
`miyauchi.a.aa@m.titech.ac.jp`

Abstract

Given an edge-weighted undirected graph $G = (V, E, w)$, the *density* of $S \subseteq V$ is defined as $w(S)/|S|$, where $w(S)$ is the sum of weights of the edges in the subgraph induced by S. The densest subgraph problem asks for $S \subseteq V$ that maximizes the density $w(S)/|S|$. The problem has received significant attention recently because it can be solved exactly in polynomial time. However, the densest subgraph problem has a drawback; it may happen that the obtained subset is too large or too small in comparison with the desired size of the output.

In this study, we address the size issue by generalizing the density of $S \subseteq V$. Specifically, we introduce the *f-density* of $S \subseteq V$, which is defined as $w(S)/f(|S|)$, where $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a monotonically non-decreasing function. In the *f-densest subgraph problem* (f-DS), we are asked to find $S \subseteq V$ that maximizes the f-density $w(S)/f(|S|)$. Although f-DS does not explicitly specify the size of the output subset of vertices, we can handle the above size issue using a *convex* size function f or a *concave* size function f appropriately. For f-DS with convex function f, we propose a nearly-linear-time algorithm with a provable approximation guarantee. In particular, for f-DS with $f(x) = x^\alpha$ ($\alpha \in [1, 2]$), our algorithm has an approximation ratio of $2 \cdot n^{(\alpha-1)(2-\alpha)}$. On the other hand, for f-DS with concave function f, we propose a linear-programming-based polynomial-time exact algorithm. It should be emphasized that this algorithm obtains not only an optimal solution to the problem but also subsets of vertices corresponding to the extreme points of the upper convex hull of $\{(|S|, w(S)) \mid S \subseteq V\}$, which we refer to as the *dense frontier points*. We also propose a flow-based combinatorial exact algorithm for unweighted graphs that runs in $O(n^3)$ time. Finally, we propose a nearly-linear-time 3-approximation algorithm.

1998 ACM Subject Classification G.2.2 Graph Theory: Network problems

Keywords and phrases graphs, dense subgraph extraction, densest subgraph problem, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.44

1 Introduction

Finding dense components in a graph is an active research topic in graph mining. Techniques for identifying dense subgraphs have been used in various applications. For example, in Web graph analysis, they are used for detecting communities (i.e., sets of web pages dealing with the same or similar topics) [9] and spam link farms [12]. As another example, in bioinformatics, they are used for finding molecular complexes in protein interaction networks [4] and

[*] The first author is supported by a Grant-in-Aid for Young Scientists (B) (No. 16K16005). The second author is supported by a Grant-in-Aid for JSPS Fellows (No. 26-11908).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 44; pp. 44:1–44:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

identifying regulatory motifs in DNA [10]. Furthermore, they are also used for expert team formation [6, 17] and real-time story identification in micro-blogging streams [2].

To date, various optimization problems have been considered to find dense components in a graph. The densest subgraph problem is one of the most well-studied optimization problems. Let $G = (V, E, w)$ be an edge-weighted undirected graph consisting of $n = |V|$ vertices, $m = |E|$ edges, and a weight function $w : E \to \mathbb{Q}_{>0}$, where $\mathbb{Q}_{>0}$ is the set of positive rational numbers. For a subset of vertices $S \subseteq V$, let $G[S]$ be the subgraph induced by $S \subseteq V$, i.e., $G[S] = (S, E(S))$, where $E(S) = \{\{i, j\} \in E \mid i, j \in S\}$. The *density* of $S \subseteq V$ is defined as $w(S)/|S|$, where $w(S) = \sum_{e \in E(S)} w(e)$. In the (weighted) densest subgraph problem, given an (edge-weighted) undirected graph $G = (V, E, w)$, we are asked to find $S \subseteq V$ that maximizes the density $w(S)/|S|$.

The densest subgraph problem has received significant attention recently because it can be solved exactly in time polynomial in n and m. In fact, there exist a flow-based exact algorithm [13] and a linear-programming-based (LP-based) exact algorithm [7]. Moreover, Charikar [7] demonstrated that the greedy algorithm designed by Asahiro et al. [3], which is called the *greedy peeling*, obtains a 2-approximate solution[1] for any instance. This algorithm runs in $O(m + n)$ time for unweighted graphs and $O(m + n \log n)$ time for weighted graphs.

However, the densest subgraph problem has a drawback; it may happen that the obtained subset is too large or too small in comparison with the desired size of the output. To overcome this issue, some variants of the problem have often been employed. The densest k-subgraph problem (DkS) is a straightforward size-restricted variant of the densest subgraph problem. In this problem, given an additional input k being a positive integer, we are asked to find $S \subseteq V$ of size k that maximizes the density $w(S)/|S|$. Note that in this problem, the objective function can be replaced by $w(S)$ since $|S|$ is fixed to k. Unfortunately, it is known that this size restriction makes the problem much harder to solve. In fact, Khot [14] proved that DkS has no PTAS under some reasonable computational complexity assumption. The current best approximation algorithm has an approximation ratio of $O(n^{1/4+\epsilon})$ for any $\epsilon > 0$ [5].

Furthermore, Andersen and Chellapilla [1] introduced two relaxed versions of DkS. The first problem, the densest at-least-k-subgraph problem (Dal$k$S), asks for $S \subseteq V$ that maximizes the density $w(S)/|S|$ under the size constraint $|S| \geq k$. For this problem, Andersen and Chellapilla [1] adopted the greedy peeling, and demonstrated that the algorithm yields a 3-approximate solution for any instance. Later, Khuller and Saha [15] investigated the problem more deeply. They proved that Dal$k$S is NP-hard, and designed a flow-based algorithm and an LP-based algorithm. These algorithms have an approximation ratio of 2, which improves the above approximation ratio of 3. The second problem is called the densest at-most-k-subgraph problem (Dam$k$S), which asks for $S \subseteq V$ that maximizes the density $w(S)/|S|$ under the size constraint $|S| \leq k$. The NP-hardness is immediate since finding a maximum clique can be reduced to it. Khuller and Saha [15] proved that approximating Dam$k$S is as hard as approximating DkS within a constant factor.

1.1 Our Contribution

In this study, we address the above size issue by generalizing the density of $S \subseteq V$. Specifically, we introduce the *f-density* of $S \subseteq V$, which is defined as $w(S)/f(|S|)$, where $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$

[1] A feasible solution is called a *γ-approximate solution* if its objective value times γ is greater than or equal to the optimal value. An algorithm is called a *γ-approximation algorithm* if it runs in polynomial time and returns a γ-approximate solution for any instance. For a γ-approximation algorithm, γ is referred to as an *approximation ratio* of the algorithm.

is a monotonically non-decreasing function with $f(0) = 0$.[2] Note that $\mathbb{Z}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$ are the set of nonnegative integers and the set of nonnegative real numbers, respectively. In the *f-densest subgraph problem* (f-DS), we are asked to find $S \subseteq V$ that maximizes the f-density $w(S)/f(|S|)$. For simplicity, we assume that $E \neq \emptyset$. Thus, any optimal solution S^* satisfies $|S^*| \geq 2$. Although f-DS does not explicitly specify the size of the output subset of vertices, we can handle the above size issue using a *convex* size function f or a *concave* size function f appropriately. In fact, we see that any optimal solution to f-DS with convex (resp. concave) function f has a size smaller (resp. larger) than or equal to that of the densest subgraph. For details, see Section 2 and Section 3.

Here we mention the relationship between our problem and DkS. Any optimal solution S^* to f-DS is a maximum weight subset of size $|S^*|$, i.e., $\mathrm{argmax}\{w(S) \mid S \subseteq V,\ |S| = |S^*|\}$, which implies that S^* is also optimal to DkS with $k = |S^*|$. Furthermore, a γ-approximation algorithm for DkS implies a γ-approximation algorithm for f-DS. Using the above $O(n^{1/4+\epsilon})$-approximation algorithm for DkS, we can obtain an $O(n^{1/4+\epsilon})$-approximation algorithm for f-DS.

We summarize our results for each of the case where f is convex and f is concave.

The case where f is convex. Let us describe our results for the case where the size function f is convex. A function $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is called *convex* if $f(x) - 2f(x+1) + f(x+2) \geq 0$ holds for any $x \in \mathbb{Z}_{\geq 0}$. We first prove the NP-hardness of f-DS with a certain convex function by a reduction from Dam$k$S. Thus, for f-DS with convex function f, one of the best possible ways is to design algorithms with a provable approximation guarantee.

To this end, we propose a $\min\left\{\frac{2f(n)/n}{f(|S^*|)-f(|S^*|-1)}, \frac{f(2)/2}{f(|S^*|)/|S^*|^2}\right\}$-approximation algorithm, where $S^* \subseteq V$ is an optimal solution to f-DS with convex function f. Our algorithm consists of the following two procedures, and outputs the better solution found by them. The first one is based on the brute-force search, which obtains an $\frac{f(2)/2}{f(|S^*|)/|S^*|^2}$-approximate solution in $O(m+n)$ time. The second one adopts the greedy peeling, which obtains a $\frac{2f(n)/n}{f(|S^*|)-f(|S^*|-1)}$-approximate solution in $O(m + n\log n)$ time. Thus, the total running time of our algorithm is $O(m + n\log n)$. Our analysis on the approximation ratio of the second procedure extends the analysis by Charikar [7] for the densest subgraph problem.

At the end of our analysis, we observe the behavior of the approximation ratio of our algorithm for three size functions. We consider size functions *between* linear and quadratic because, as we will see later, f-DS with any super-quadratic size function only produces constant-size optimal solutions. The first example is $f(x) = x^\alpha$ ($\alpha \in [1,2]$). We show that the approximation ratio of our algorithm is $2 \cdot n^{(\alpha-1)(2-\alpha)}$, where the worst-case performance of $2 \cdot n^{1/4}$ is attained at $\alpha = 1.5$. The second example is $f(x) = \lambda x + (1-\lambda)x^2$ ($\lambda \in [0,1)$). For this case, the approximation ratio of our algorithm is $(2-\lambda)/(1-\lambda)$, which is a constant for a fixed λ. The third example is $f(x) = x^2/(\lambda x + (1-\lambda))$ ($\lambda \in [0,1]$). Note that this size function is derived by density function $\lambda\frac{w(S)}{|S|} + (1-\lambda)\frac{w(S)}{|S|^2}$. The approximation ratio of our algorithm is $4/(1+\lambda)$, which is at most 4.

The case where f is concave. Next let us describe our results for the case where the size function f is concave. A function $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is called *concave* if $f(x) - 2f(x+1) +$

[2] To handle various types of functions (e.g., $f(x) = x^\alpha$ for $\alpha > 0$), we set the codomain of the function f to be the set of nonnegative *real* numbers. We assume that we can compare $p \cdot f(i)$ and $q \cdot f(j)$ in constant time for any $p, q \in \mathbb{Q}$ and $i, j \in \mathbb{Z}_{\geq 0}$.

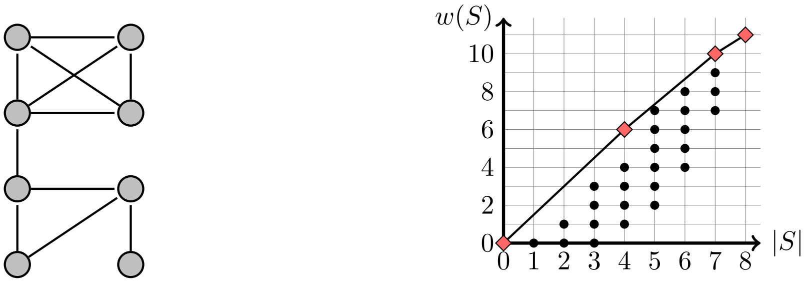


Figure 1 An example graph and corresponding points in $\mathcal{P} = \{(|S|, w(S)) \mid S \subseteq V\}$. The diamond-shaped points, i.e., $(0,0), (4,6), (7,10)$, and $(8,11)$, are the dense frontier points.

$f(x+2) \leq 0$ holds for any $x \in \mathbb{Z}_{\geq 0}$. Unlike the above convex case, f-DS in this case can be solved exactly in polynomial time.

In fact, we present an LP-based exact algorithm, which extends Charikar's exact algorithm for the densest subgraph problem [7] and Khuller and Saha's 2-approximation algorithm for Dal$k$S [15]. It should be emphasized that our LP-based algorithm obtains not only an optimal solution to f-DS but also some attractive subsets of vertices. Let us see an example in Figure 1. The graph consists of 8 vertices and 11 unweighted edges. For this graph, we plotted all the points contained in $\mathcal{P} = \{(|S|, w(S)) \mid S \subseteq V\}$. We refer to the extreme points of the upper convex hull of $\mathcal{P}$ as the *dense frontier points*. The densest subgraph is a typical subset of vertices that corresponds to a dense frontier point. Our LP-based algorithm obtains a corresponding subset of vertices for every dense frontier point.

Moreover, in this concave case, we design a combinatorial exact algorithm for unweighted graphs. Our algorithm is based on the standard technique for fractional programming. By using the technique, we can reduce f-DS to a sequence of submodular function minimizations. However, applying a submodular function minimization algorithm leads to a computationally expensive algorithm, which runs in $O(n^5(m+n) \cdot \log n)$ time. To reduce the computation time, we replace a submodular function minimization algorithm with a much faster flow-based algorithm that substantially extends a technique of Goldberg's flow-based algorithm for the densest subgraph problem [13]. The total running time of our algorithm is $O(n^3)$.

Although our flow-based algorithm is much faster than the reduction-based algorithm, the running time is still long for large-sized graphs. To design an algorithm with much higher scalability, we adopt the greedy peeling. As mentioned above, this algorithm runs in $O(m+n)$ time for unweighted graphs and $O(m + n \log n)$ time for weighted graphs. We see that the algorithm yields a 3-approximate solution for any instance.

1.2 Related Work

Tsourakakis et al. [17] introduced a general optimization problem to find dense subgraphs, which is referred to as the optimal (g, h, α)-edge-surplus problem. The problem asks for $S \subseteq V$ that maximizes $\mathsf{edge\text{-}surplus}_\alpha(S) = g(|E(S)|) - \alpha h(|S|)$, where g and h are strictly monotonically increasing functions, and $\alpha > 0$ is a constant. The intuition behind this optimization problem is the same as that of ours. In fact, the first term $g(|E(S)|)$ prefers $S \subseteq V$ that has a large number of edges, whereas the second term $-\alpha h(|S|)$ penalizes $S \subseteq V$ with a large size. Tsourakakis et al. [17] were motivated by finding near-cliques (i.e., relatively small dense subgraphs), and they derived the function $\mathsf{OQC}_\alpha(S) = |E(S)| - \alpha\binom{|S|}{2}$, which is called the OQC function, by setting $g(x) = x$ and $h(x) = x(x-1)/2$. For OQC function maximization, they adopted the greedy peeling and a simple local search heuristic.

Recently, Yanagisawa and Hara [18] introduced density function $|E(S)|/|S|^\alpha$ for $\alpha \in (1,2]$, which they called the discounted average degree. For discounted average degree maximization, they designed an integer-programming-based exact algorithm, which is applicable only to graphs with thousands of edges. They also designed a local search heuristic, which is applicable to Web-scale graphs but has no provable approximation guarantee. As mentioned above, our algorithm for f-DS with convex function f runs in $O(m + n\log n)$ time, and has an approximation ratio of $2 \cdot n^{(\alpha-1)(2-\alpha)}$ for $f(x) = x^\alpha$ ($\alpha \in [1,2]$).

2 Convex Case

In this section, we investigate f-DS with convex function f. A function $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is called *convex* if $f(x) - 2f(x+1) + f(x+2) \geq 0$ holds for any $x \in \mathbb{Z}_{\geq 0}$. We remark that $f(x)/x$ is monotonically non-decreasing for x since we assume that $f(0) = 0$. It should be emphasized that any optimal solution to f-DS with convex function f has a size smaller than or equal to that of the densest subgraph. To see this, let $S^* \subseteq V$ be an optimal solution to f-DS and $S^*_{\mathrm{DS}} \subseteq V$ be the densest subgraph. Then we have

$$\frac{f(|S^*|)}{|S^*|} = \frac{w(S^*)/|S^*|}{w(S^*)/f(|S^*|)} \leq \frac{w(S^*_{\mathrm{DS}})/|S^*_{\mathrm{DS}}|}{w(S^*_{\mathrm{DS}})/f(|S^*_{\mathrm{DS}}|)} = \frac{f(|S^*_{\mathrm{DS}}|)}{|S^*_{\mathrm{DS}}|}. \tag{1}$$

This implies the statement because $f(x)/x$ is monotonically non-decreasing.

2.1 Hardness

We first state that f-DS with convex function f contains Dam$k$S as a special case.

▶ **Theorem 1.** *For any integer $k \in [2,n]$, $S \subseteq V$ is optimal to DamkS if and only if S is optimal to f-DS with (convex) function $f(x) = \max\{x,\ 2w(V)\cdot(x-k)/w(e)+k\}$, where e is an arbitrary edge.*

2.2 Our Algorithm

In this subsection, we provide an algorithm for f-DS with convex function f. Our algorithm consists of the following two procedures, and outputs the better solution found by them. Let S^* be an optimal solution to the problem. The first one is based on the brute-force search, which obtains an $\frac{f(2)/2}{f(|S^*|)/|S^*|^2}$-approximate solution in $O(m+n)$ time. The second one adopts the greedy peeling [3], which obtains a $\frac{2f(n)/n}{f(|S^*|)-f(|S^*|-1)}$-approximate solution in $O(m + n\log n)$ time. Combining these results, which will be proved later, we have the following theorem.

▶ **Theorem 2.** *Let $S^* \subseteq V$ be an optimal solution to f-DS with convex function f. For the problem, our algorithm runs in $O(m + n\log n)$ time, and it has an approximation ratio of*

$$\min\left\{\frac{2f(n)/n}{f(|S^*|)-f(|S^*|-1)},\ \frac{f(2)/2}{f(|S^*|)/|S^*|^2}\right\}.$$

2.2.1 Brute-Force Search

As will be shown below, to obtain an approximation ratio of $\frac{f(2)/2}{f(|S^*|)/|S^*|^2}$, it suffices to find the heaviest edge. Clearly, this algorithm runs in $O(m+n)$ time. However, we present a more general algorithm, which is useful for some case. Our algorithm examines all the

Algorithm 1 Brute-force search

```
for $i \leftarrow 2, \dots, k$
    Find $S_i^* \in \mathrm{argmax}\{w(S) \mid S \subseteq V,\ |S| = i\}$ by examining all the candidate subsets
return $S \in \{S_2^*, \dots, S_k^*\}$ that maximizes $w(S)/f(|S|)$
```

subsets of vertices of size at most k, and then returns an optimal subset among them, where k is a constant and $k \geq 2$. For reference, we describe the procedure in Algorithm 1.

This algorithm can be implemented to run in $O((m+n)n^k)$ time because the number of subsets with at most k vertices is $\sum_{i=0}^{k} \binom{n}{i} = O(n^k)$ and the value of $w(S)/f(|S|)$ for $S \subseteq V$ can be computed in $O(m+n)$ time.

We analyze the approximation ratio of the algorithm. Let S_i^* denote a maximum weight subset of size i, i.e., $S_i^* \in \mathrm{argmax}\{w(S) \mid S \subseteq V,\ |S| = i\}$. We refer to $w(S_i^*)/\binom{i}{2}$ as the *edge density* of i vertices. The following lemma gives a fundamental property of the edge density.

▶ **Lemma 3.** *The edge density is monotonically non-increasing for the number of vertices, i.e., $w(S_i^*)/\binom{i}{2} \geq w(S_j^*)/\binom{j}{2}$ holds for any $1 \leq i \leq j \leq n$.*

Using the above lemma, we can provide the result of the approximation ratio.

▶ **Lemma 4.** *Let $S^* \subseteq V$ be an optimal solution to f-DS with convex function f. If $|S^*| \leq k$, then Algorithm 1 obtains an optimal solution. If $|S^*| \geq k$, then it holds that*

$$\frac{w(S^*)}{f(|S^*|)} \leq \frac{2 \cdot f(k)/k^2}{f(|S^*|)/|S^*|^2} \cdot \frac{w(S_k^*)}{f(k)}.$$

Proof. If $|S^*| \leq k$, then Algorithm 1 obtains an optimal solution since $S^* \in \{S_2^*, \dots, S_k^*\}$. If $|S^*| \geq k$, then we have

$$\begin{aligned}\frac{w(S^*)}{f(|S^*|)} &\leq \frac{f(k)/\binom{k}{2}}{f(|S^*|)/\binom{|S^*|}{2}} \cdot \frac{w(S_k^*)}{f(k)} \\ &= \frac{1-1/|S^*|}{1-1/k} \cdot \frac{f(k)/k^2}{f(|S^*|)/|S^*|^2} \cdot \frac{w(S_k^*)}{f(k)} \leq \frac{2 \cdot f(k)/k^2}{f(|S^*|)/|S^*|^2} \cdot \frac{w(S_k^*)}{f(k)},\end{aligned}$$

where the first inequality follows from Lemma 3, and the last inequality follows from $|S^*| \geq k \geq 2$. ◀

From this lemma, we see that Algorithm 1 with $k = 2$ has an approximation ratio of $\frac{f(2)/2}{f(|S^*|)/|S^*|^2}$.

2.2.2 Greedy Peeling

Here we adopt the greedy peeling. For $S \subseteq V$ and $v \in S$, let $d_S(v)$ denote the weighted degree of the vertex v in the induced subgraph $G[S]$, i.e., $d_S(v) = \sum_{\{u,v\} \in E(S)} w(\{u,v\})$. Our algorithm iteratively removes the vertex with the smallest weighted degree in the currently remaining graph, and then returns $S \subseteq V$ with maximum $w(S)/f(|S|)$ over the iteration. For reference, we describe the procedure in Algorithm 2. This algorithm runs in $O(m+n)$ time for unweighted graphs and $O(m + n \log n)$ time for weighted graphs.

The following lemma provides the result of the approximation ratio.

▶ **Lemma 5.** *Let S^* be an optimal solution to f-DS with convex function f. Algorithm 2 returns a solution $S \subseteq V$ that satisfies*

$$\frac{w(S)}{f(|S|)} \geq \frac{1}{2} \cdot \frac{f(|S^*|) - f(|S^*|-1)}{f(n)/n} \cdot \frac{w(S^*)}{f(|S^*|)}.$$

Algorithm 2 Greedy peeling

```
$S_n \leftarrow V$
for $i \leftarrow n, \dots, 2$
    Find $v_i \in \mathrm{argmin}_{v \in S_i} d_{S_i}(v)$ and $S_{i-1} \leftarrow S_i \setminus \{v_i\}$
return $S \in \{S_1, \dots, S_n\}$ that maximizes $w(S)/f(|S|)$
```

Proof. Choose an arbitrary vertex $v \in S^*$. By the optimality of S^*, we have

$$\frac{w(S^*)}{f(|S^*|)} \geq \frac{w(S^* \setminus \{v\})}{f(|S^*|-1)}.$$

By using the fact that $w(S^* \setminus \{v\}) = w(S^*) - d_{S^*}(v)$, this inequality can be transformed to

$$d_{S^*}(v) \geq (f(|S^*|) - f(|S^*|-1)) \cdot \frac{w(S^*)}{f(|S^*|)}. \tag{2}$$

Let l be the smallest index that satisfies $S^* \subseteq S_l$. Note that $v_l \in S^*$. Using inequality (2), we have

$$\begin{aligned}\frac{w(S)}{f(|S|)} &\geq \frac{w(S_l)}{f(l)} = \frac{1}{2} \cdot \frac{\sum_{u \in S_l} d_{S_l}(u)}{f(l)} \geq \frac{1}{2} \cdot \frac{l \cdot d_{S_l}(v_l)}{f(l)} \geq \frac{1}{2} \cdot \frac{d_{S^*}(v_l)}{f(l)/l} \\ &\geq \frac{1}{2} \cdot \frac{f(|S^*|) - f(|S^*|-1)}{f(l)/l} \cdot \frac{w(S^*)}{f(|S^*|)} \geq \frac{1}{2} \cdot \frac{f(|S^*|) - f(|S^*|-1)}{f(n)/n} \cdot \frac{w(S^*)}{f(|S^*|)},\end{aligned}$$

where the second inequality follows from the greedy choice of v_l, the third inequality follows from $S_l \supseteq S^*$, and the last inequality follows from the monotonicity of $f(x)/x$. ◀

2.3 Examples

Here we observe the behavior of the approximation ratio of our algorithm for three convex size functions. We consider size functions *between* linear and quadratic because f-DS with any super-quadratic size function only produces constant-size optimal solutions. This follows from the inequality $\frac{f(2)/2}{f(|S^*|)/|S^*|^2} \geq 1$ (i.e., $f(2)/2 \geq f(|S^*|)/|S^*|^2$) by Lemma 4.

(i) The case where $f(x) = x^\alpha$ ($\alpha \in [1, 2]$). The following corollary provides the approximation ratio of our algorithm.

▶ **Corollary 6.** *For f-DS with $f(x) = x^\alpha$ ($\alpha \in [1, 2]$), our algorithm has an approximation ratio of $2 \cdot n^{(\alpha-1)(2-\alpha)}$.*

Proof. Let $s = |S^*|$. By Theorem 2, the approximation ratio is at most

$$\begin{aligned}\min\left\{\frac{2f(n)/n}{f(s) - f(s-1)}, \frac{f(2)/2}{f(s)/s^2}\right\} &= \min\left\{\frac{2n^{\alpha-1}}{s^\alpha - (s-1)^\alpha}, 2^{\alpha-1} \cdot s^{2-\alpha}\right\} \\ &\leq \min\left\{\frac{2n^{\alpha-1}}{s^{\alpha-1}}, 2 \cdot s^{2-\alpha}\right\} \leq 2 \cdot n^{(\alpha-1)(2-\alpha)}.\end{aligned}$$

The first inequality follows from the fact that $s^\alpha - (s-1)^\alpha = s^\alpha - (s-1)^{\alpha-1}(s-1) \geq s^\alpha - s^{\alpha-1}(s-1) = s^{\alpha-1}$. The last inequality follows from the fact that the first term and the second term of the minimum function are monotonically non-increasing and non-decreasing for s, respectively, and they have the same value at $s = n^{\alpha-1}$. ◀

Note that an upper bound on $2 \cdot n^{(\alpha-1)(2-\alpha)}$ is $2 \cdot n^{1/4}$, which is attained at $\alpha = 1.5$.

(ii) The case where $\boldsymbol{f(x) = \lambda x + (1-\lambda)x^2}$ $\boldsymbol{(\lambda \in [0,1))}$. The following corollary provides an approximation ratio of Algorithm 1, which is a constant for a fixed λ.

▶ **Corollary 7.** *For f-DS with $f(x) = \lambda x + (1-\lambda)x^2$ $(\lambda \in [0,1))$, Algorithm 1 with $k = 2$ has an approximation ratio of $(2-\lambda)/(1-\lambda)$. Furthermore, for any $\epsilon > 0$, Algorithm 1 with $k \geq \frac{2}{\epsilon} \cdot \frac{\lambda}{1-\lambda}$ has an approximation ratio of $2 + \epsilon$.*

(iii) The case where $\boldsymbol{f(x) = x^2/(\lambda x + (1-\lambda))}$ $\boldsymbol{(\lambda \in [0,1])}$. This size function is derived by density function $\lambda \frac{w(S)}{|S|} + (1-\lambda)\frac{w(S)}{|S|^2}$. The following corollary provides an approximation ratio of our algorithm, which is at most 4.

▶ **Corollary 8.** *For f-DS with $f(x) = x^2/(\lambda x + (1-\lambda))$ $(\lambda \in [0,1))$, our algorithm has an approximation ratio of $4/(1+\lambda)$.*

3 Concave Case

In this section, we investigate f-DS with concave function f. A function $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is called *concave* if $f(x) - 2f(x+1) + f(x+2) \leq 0$ holds for any $x \in \mathbb{Z}_{\geq 0}$. We remark that $f(x)/x$ is monotonically non-increasing for x since we assume that $f(0) = 0$. Note that any optimal solution to f-DS with concave function f has a size larger than or equal to that of the densest subgraph. This follows from inequality (1) and the monotonicity of $f(x)/x$.

3.1 Dense Frontier Points

Here we define the dense frontier points and prove some basic properties. We denote by $\mathcal{P}$ the set $\{(|S|, w(S)) \mid S \subseteq V\}$. A point $(x, y) \in \mathcal{P}$ is called a *dense frontier point* if it is a unique maximizer of $y - \lambda x$ over $\mathcal{P}$ for some $\lambda > 0$. In other words, the extreme points of the upper convex hull of $\mathcal{P}$ are the dense frontier points. The densest subgraph is a typical subset of vertices corresponding to a dense frontier point. We prove that (i) for any dense frontier point, there exists some concave function f such that any optimal solution to f-DS with the function f corresponds to the dense frontier point, and conversely, (ii) for any strictly concave function f (i.e., f that satisfies $f(x) - 2f(x+1) + f(x+2) < 0$ for any $x \in \mathbb{Z}_{\geq 0}$), any optimal solution to f-DS with the function f corresponds to a dense frontier point.

We first prove (i). Note that each dense frontier point can be written as $(i, w(S_i^*))$ for some i, where S_i^* is a maximum weight subset of size i. Let $(k, w(S_k^*))$ be a dense frontier point and assume that it is a unique maximizer of $y - \hat{\lambda}x$ over $\mathcal{P}$ for $\hat{\lambda} > 0$. Consider the concave function f such that $f(x) = \hat{\lambda}x + w(S_k^*) - \hat{\lambda}k$ for $x > 0$ and $f(0) = 0$. Then, any optimal solution S^* to f-DS with the function f corresponds to the dense frontier point (i.e., $(|S^*|, w(S^*)) = (k, w(S_k^*))$ holds) because $w(S)/f(|S|)$ is greater than or equal to 1 if and only if $w(S) - \hat{\lambda}|S| \geq w(S_k^*) - \hat{\lambda}k$.

We next prove (ii). Let f be any strictly concave function. Let S_k^* be an optimal solution to f-DS with the function f, and take $\hat{\lambda}$ that satisfies $(f(k) - f(k-1)) \cdot \frac{w(S_k^*)}{f(k)} > \hat{\lambda} > (f(k+1) - f(k)) \cdot \frac{w(S_k^*)}{f(k)}$. Note that the strict concavity of f guarantees the existence of such $\hat{\lambda}$. Since f is strictly concave, we have $w(S_k^*) + \hat{\lambda}(|S| - k) \geq \frac{w(S_k^*)}{f(k)} \cdot f(|S|) \geq \frac{w(S)}{f(|S|)} \cdot f(|S|) = w(S)$ for any $S \subseteq V$, and the equalities hold only when $(|S|, w(S)) = (k, w(S_k^*))$. Thus, $(k, w(S_k^*))$ is a unique maximizer of $y - \hat{\lambda}x$ over $\mathcal{P}$, and hence is a dense frontier point.

Algorithm 3 LP-based algorithm

```
for $k \leftarrow 1, \dots, n$
    Solve $\mathrm{LP}_k$ and obtain an optimal solution $(x^k, y^k)$
    Compute $r_k^*$ that maximizes $w(S^k(r))/f(|S^k(r)|)$
return $S \in \{S^1(r_1^*), \dots, S^n(r_n^*)\}$ that maximizes $w(S)/f(|S|)$
```

3.2 LP-Based Algorithm

We provide an LP-based polynomial-time exact algorithm. We introduce a variable x_e for each $e \in E$ and a variable y_v for each $v \in V$. For $k = 1, \dots, n$, we construct the following linear programming problem:

$$\mathrm{LP}_k: \ \text{max.} \sum_{e \in E} w(e) \cdot x_e \ \text{ s.t.} \sum_{v \in V} y_v = k, \quad x_e \le y_u, \ x_e \le y_v \ (\forall e = \{u, v\} \in E),$$
$$x_e, y_v \in [0, 1] \ (\forall e \in E, \ \forall v \in V).$$

For an optimal solution (x^k, y^k) to LP_k and a real parameter r, we define a sequence of subsets $S^k(r) = \{v \in V \mid y_v^k \ge r\}$. For $k = 1, \dots, n$, our algorithm solves LP_k to obtain an optimal solution (x^k, y^k), and computes r_k^* that maximizes $w(S^k(r))/f(|S^k(r)|)$. Note here that to find such an r_k^*, it suffices to check all the distinct sets $S^k(r)$ by simply setting $r = y_v^k$ for every $v \in V$. The algorithm returns $S \in \{S^1(r_1^*), \dots, S^n(r_n^*)\}$ that maximizes $w(S)/f(|S|)$. For reference, we describe the procedure in Algorithm 3. Clearly, the algorithm runs in polynomial time.

In what follows, we demonstrate that Algorithm 3 obtains an optimal solution to f-DS with concave function f. The following lemma provides a lower bound on the optimal value of LP_k.

▶ **Lemma 9.** *For any $S \subseteq V$, the optimal value of $\mathrm{LP}_{|S|}$ is at least $w(S)$.*

We have the following key lemma.

▶ **Lemma 10.** *Let $S^* \subseteq V$ be an optimal solution to f-DS with concave function f, and let $k^* = |S^*|$. Furthermore, let (x^*, y^*) be an optimal solution to LP_{k^*}. Then, there exists a real number r such that $S^{k^*}(r)$ is optimal to f-DS with concave function f.*

Proof. For each $e = \{u, v\} \in E$, we have $x_e^* = \min\{y_u^*, y_v^*\}$ from the optimality of (x^*, y^*). Without loss of generality, we relabel the indices of (x^*, y^*) so that $y_1^* \ge \cdots \ge y_n^*$. Then we have

$$\begin{aligned}
\int_0^{y_1^*} w(S^{k^*}(r))dr &= \int_0^{y_1^*} \left(\sum_{e=\{u,v\} \in E} w(e) \cdot [y_u^* \ge r \text{ and } y_v^* \ge r] \right) dr \\
&= \sum_{e=\{u,v\} \in E} \int_0^{y_1^*} (w(e) \cdot [y_u^* \ge r \text{ and } y_v^* \ge r])\, dr \\
&= \sum_{e=\{u,v\} \in E} w(e) \cdot \min\{y_u^*, y_v^*\} \ge \sum_{e \in E} w(e) \cdot x_e^* \ge w(S^*),
\end{aligned}$$

where $[y_u^* \ge r \text{ and } y_v^* \ge r]$ is 1 if the condition in the square bracket is satisfied and 0 otherwise, and the last inequality follows from Lemma 9. Moreover, we have

$$\begin{aligned}\int_0^{y_1^*} f(|S^{k^*}(r)|)dr &= \sum_{h=1}^{n} f(h)\cdot(y_h^* - y_{h+1}^*) = \sum_{h=1}^{n}(f(h)-f(h-1))\cdot y_h^* \\ &\le \sum_{h=1}^{k^*}(f(h)-f(h-1)) = f(k^*)-f(0) = f(k^*),\end{aligned}$$

where we assume that $y_{n+1}^* = 0$, and the inequality holds by the concavity of f (i.e., $f(h+2)-f(h+1) \le f(h+1)-f(h)$), $\sum_{h=1}^n y_h^* = k^*$, and $y_h^* \le 1$.

Let r^* be a real number that maximizes $w(S^{k^*}(r))/f(|S^{k^*}(r)|)$ in $[0, y_1^*]$. Using the above two inequalities, we have

$$\begin{aligned}\frac{w(S^*)}{f(k^*)} &\le \frac{\int_0^{y_1^*} w(S^{k^*}(r))dr}{\int_0^{y_1^*} f(|S^{k^*}(r)|)dr} = \frac{\int_0^{y_1^*}\left(\frac{w(S^{k^*}(r))}{f(|S^{k^*}(r)|)}\cdot f(|S^{k^*}(r)|)\right)dr}{\int_0^{y_1^*} f(|S^{k^*}(r)|)dr} \\ &\le \frac{\int_0^{y_1^*}\left(\frac{w(S^{k^*}(r^*))}{f(|S^{k^*}(r^*)|)}\cdot f(|S^{k^*}(r)|)\right)dr}{\int_0^{y_1^*} f(|S^{k^*}(r)|)dr} = \frac{w(S^{k^*}(r^*))}{f(|S^{k^*}(r^*)|)}.\end{aligned}$$

This completes the proof. ◀

Clearly, Algorithm 3 examines $S^{k^*}(r^*)$ as a candidate subset of the output. Therefore, we have the result.

▶ **Theorem 11.** *Algorithm 3 is a polynomial-time exact algorithm for f-DS with concave function f.*

By Lemma 10, for any concave function f, an optimal solution to f-DS with the function f is contained in $\{S^k(r) \mid k = 1, \dots, n,\ r \in [0,1]\}$ whose cardinality is at most n^2. As shown above, for any dense frontier point, there exists some concave function f such that any optimal solution to f-DS with the function f corresponds to the dense frontier point. Thus, we have the following result.

▶ **Theorem 12.** *We can find a corresponding subset of vertices for every dense frontier point in polynomial time.*

3.3 Flow-Based Algorithm

Here we provide a combinatorial exact algorithm for unweighted graphs (i.e., $w(e) = 1$ for every $e \in E$). We first show that using the standard technique for fractional programming, we can reduce f-DS with concave function f to a sequence of submodular function minimizations. The critical fact is that $\max_{S\subseteq V} w(S)/f(|S|)$ is at least β if and only if $\min_{S\subseteq V}(\beta\cdot f(|S|) - w(S))$ is at most 0. Note that for $\beta \ge 0$, the function $\beta \cdot f(|S|) - w(S)$ is submodular because $\beta \cdot f(|S|)$ and $-w(S)$ are submodular [11]. Thus, we can calculate $\min_{S\subseteq V}(\beta\cdot f(|S|) - w(S))$ in $O(n^5(m+n))$ time using Orlin's algorithm [16], which implies that we can determine $\max_{S\subseteq V} w(S)/f(|S|) \ge \beta$ or not in $O(n^5(m+n))$ time. Hence, we can obtain the value of $\max_{S\subseteq V} w(S)/f(|S|)$ by binary search. Note that the objective function of unweighted f-DS may have at most $O(mn)$ distinct values since $w(S)$ is a nonnegative integer at most m. Thus, the procedure yields an optimal solution in $O(\log(nm)) = O(\log n)$ iterations. The total running time is $O(n^5(m+n)\cdot \log n)$.

To reduce the computation time, we replace Orlin's algorithm with a much faster algorithm that substantially extends a technique of Goldberg's flow-based algorithm for the densest

Algorithm 4 Flow-based algorithm

```
Construct the (unweighted) directed network $(U, A)$
$\{b_1, \dots, b_r\} = \{p/f(q) \mid p = 0, 1, \dots, m,\ q = 2, 3, \dots, n\}$ such that $b_1 < \cdots < b_r$
$i_{\min} \leftarrow 1$ and $i_{\max} \leftarrow r$
while TRUE
    $i \leftarrow \lfloor (i_{\max} + i_{\min})/2 \rfloor$
    Compute a minimum $s$–$t$ cut $(X, Y)$ in $(U, A, w_{b_i})$
    if the cost of $(X, Y)$ is larger than $w(V)$ then $i_{\max} \leftarrow i - 1$
    else if the cost of $(X, Y)$ is less than $w(V)$ then $i_{\min} \leftarrow i + 1$
    else return $X \cap V$
```

subgraph problem [13]. The key technique is to represent the value of $\beta \cdot f(|S|) - w(S)$ using the cost of minimum cut of a certain directed network constructed from G and $\beta \geq 0$.

For a given graph $G = (V, E, w)$ and a real number $\beta \geq 0$, we construct a directed network (U, A, w_β) as follows. The vertex set U is defined by $U = V \cup P \cup \{s, t\}$, where $P = \{p_1, \dots, p_n\}$. The edge set A is given by $A = A_s \cup A_t \cup A_1 \cup A_2$, where

$$A_s = \{(s, v) \mid v \in V\},\ A_t = \{(p, t) \mid p \in P\},$$
$$A_1 = \{(u, v), (v, u) \mid \{u, v\} \in E\},\text{ and } A_2 = \{(v, p) \mid v \in V,\ p \in P\}.$$

The edge weight $w_\beta : A \to \mathbb{R}_{\geq 0}$ is defined by

$$w_\beta(e) = \begin{cases} d(v)/2 & (e = (s, v) \in A_s), \\ \beta \cdot k \cdot a_k & (e = (p_k, t) \in A_t), \\ 1/2\ (= w(\{u, v\})/2) & (e = (u, v) \in A_1), \\ \beta \cdot a_k & (e = (v, p_k) \in A_2), \end{cases}$$

where $d(v)$ is the degree of vertex v (i.e., $d(v) = |\{u \in V \mid \{u, v\} \in E\}|$), and

$$a_k = \begin{cases} 2f(k) - f(k+1) - f(k-1) & (k = 1, \dots, n-1), \\ f(n) - f(n-1) & (k = n). \end{cases}$$

Note that $a_k \geq 0$ holds since f is a monotonically non-decreasing concave function.

The following lemma reveals the relationship between an s–t cut in (U, A, w_β) and the value of $\beta \cdot f(|S|) - w(S)$.

▶ **Lemma 13.** *Let (X, Y) be any s–t cut in the network (U, A, w_β), and let $S = X \cap V$. Then, the cost of (X, Y) is equal to $w(V) + \beta \cdot f(|S|) - w(S)$.*

From this lemma, we see that the minimum s–t cut value is $w(V) + \min_{S \subseteq V}(\beta \cdot f(|S|) - w(S))$. Therefore, for a given value $\beta \geq 0$, we can determine whether there exists $S \subseteq V$ that satisfies $w(S)/f(|S|) \geq \beta$ by checking the minimum s–t cut value is at most $w(V)$ or not. Our algorithm applies binary search for β within the possible objective values of f-DS (i.e., $\{p/f(q) \mid p = 0, 1, \dots, m,\ q = 2, 3, \dots, n\}$). For reference, we describe the procedure in Algorithm 4. The minimum s–t cut problem can be solved in $O(N^3/\log N)$ time for a graph with N vertices [8]. Thus, the running time of our algorithm is $O(\frac{n^3}{\log n} \cdot \log(mn)) = O(n^3)$ since $|U| = 2n + 2$. We summarize the result in the following theorem.

▶ **Theorem 14.** *Algorithm 4 is an $O(n^3)$-time exact algorithm for unweighted f-DS with concave function f.*

Finally, we remark that Algorithm 4 can be modified for weighted f-DS with concave function f so that a $(1 + \epsilon)$-approximate solution is obtained in $O(n^3 \cdot \log(1/\epsilon))$ time.

3.4 Greedy Peeling

Here we provide an approximation algorithm with much higher scalability. Specifically, we see that the greedy peeling (Algorithm 2) has an approximation ratio of 3 for f-DS with concave function f. As mentioned above, the algorithm runs in $O(m+n)$ time for unweighted graphs and $O(m+n\log n)$ time for weighted graphs. The proof of the following theorem relies on the monotonicity of $f(x)/x$ and the fact that the greedy peeling is a 3-approximation algorithm for Dal$k$S [1].

▶ **Theorem 15.** *The greedy peeling (Algorithm 2) has an approximation ratio of* 3 *for f-DS with concave function f.*

References

1 R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW'09*, pages 25–37, 2009.

2 A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. In *VLDB'12*, pages 574–585, 2012.

3 Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *J. Algorithms*, 34(2):203–221, 2000.

4 G. D. Bader and C. W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):1–27, 2003.

5 A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: An $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC'10*, pages 201–210, 2010.

6 F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD'14*, pages 1316–1325, 2014.

7 M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX'00*, pages 84–95, 2000.

8 J. Cheriyan, T. Hagerup, and K. Mehlhorn. An $o(n^3)$-time maximum-flow algorithm. *SIAM J. Comput.*, 25(6):1144–1170, 1996.

9 Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense communities in the web. In *WWW'07*, pages 461–470, 2007.

10 E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. MotifCut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

11 S. Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, 2005.

12 D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB'05*, pages 721–732, 2005.

13 A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California Berkeley, 1984.

14 S. Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, 2006.

15 S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP'09*, pages 597–608, 2009.

16 J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Math. Program.*, 118(2):237–251, 2009.

17 C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *KDD'13*, pages 104–112, 2013.

18 H. Yanagisawa and S. Hara. Axioms of density: How to define and detect the densest subgraph. Technical report, IBM Research – Tokyo, 2016.

On the Classes of Interval Graphs of Limited Nesting and Count of Lengths*†

Pavel Klavík[1], Yota Otachi[2], and Jiří Šejnoha[3]

1 Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
klavik@iuuk.mff.cuni.cz

2 School of Information Science, Japan Advanced Institute of Science and Technology, Japan
otachi@jaist.ac.jp

3 Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
jirka.sejnoha@gmail.com

Abstract

In 1969, Roberts introduced *proper* and *unit interval graphs* and proved that these classes are equal. Natural generalizations of unit interval graphs called *k-length interval graphs* were considered in which the number of different lengths of intervals is limited by k. Even after decades of research, no insight into their structure is known and the complexity of recognition is open even for $k = 2$. We propose generalizations of proper interval graphs called *k-nested interval graphs* in which there are no chains of $k+1$ intervals nested in each other. It is easy to see that k-nested interval graphs are a superclass of k-length interval graphs.

We give a linear-time recognition algorithm for k-nested interval graphs. This algorithm adds a missing piece to Gajarský et al. [FOCS 2015] to show that testing FO properties on interval graphs is FPT with respect to the nesting k and the length of the formula, while the problem is W[2]-hard when parameterized just by the length of the formula. Further, we show that a generalization of recognition called *partial representation extension* is polynomial-time solvable for k-nested interval graphs, while it is NP-hard for k-length interval graphs, even when $k = 2$.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases interval graphs, proper and unit interval graphs, recognition, partial representation extension

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.45

1 Introduction

An *interval representation* $\mathcal{R}$ of a graph G is a collection $\big\{\langle u\rangle : u \in V(G)\big\}$ of intervals of the real line such that $uv \in E(G)$ if and only if $\langle u\rangle \cap \langle v\rangle \neq \emptyset$. A graph is an *interval graph* if it has an interval representation, and we denote this class by INT.

An interval representation is called *proper* if $\langle u\rangle \subseteq \langle v\rangle$ implies $\langle u\rangle = \langle v\rangle$, and *unit* if the length of all intervals $\langle u\rangle$ is one. The classes of *proper* and *unit interval graphs* (denoted PROPER INT and UNIT INT) consist of all interval graphs which have proper and unit interval representations, respectively. Roberts [27] proved that PROPER INT $=$ UNIT INT.

[*] We omit many details and proofs, for the full version, see [20].

[†] The first author is supported by CE-ITI (GAČR P202/12/G061) and Charles University as GAUK 196213.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 45; pp. 45:1–45:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

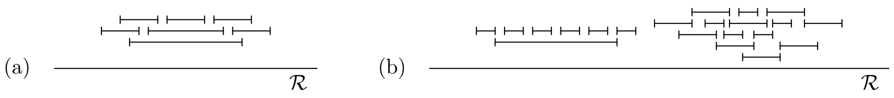


Figure 1 (a) An interval representation with the nesting three. (b) The disjoint union of two components with the minimum nesting two requiring three different lengths of intervals. On the left, the shorter intervals are shorter than $\frac{1}{4}$ of the longer ones. On the right, they are longer than $\frac{1}{3}$.

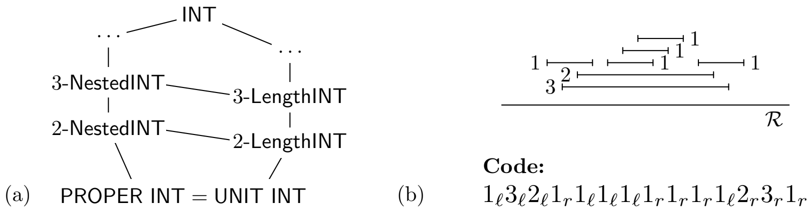


Figure 2 (a) The Hasse diagram of proper inclusions of the considered classes. (b) We can label each interval by the length of a maximal chain of nested intervals ending in it. We code the graph by the left-to-right sequence of left endpoints ℓ and right endpoints r together with their labels.

The Studied Classes. In this paper, we consider two hierarchies of subclasses of interval graphs which generalize proper and unit interval graphs. The class k-NestedINT consists of all interval graphs which have representations with no $k+1$ intervals $\langle u_0\rangle, \dots, \langle u_k\rangle$ such that $\langle u_0\rangle \subsetneq \langle u_1\rangle \subsetneq \cdots \subsetneq \langle u_k\rangle$; see Fig. 1a. The class k-LengthINT consists of all interval graphs which have representations having at most k different lengths of intervals; see Fig. 1b. We know by [27] that 1-NestedINT = PROPER INT = UNIT INT = 1-LengthINT.

For an interval graph G, we denote the minimum nesting over all interval representations by $\nu(G)$, and the minimum number of interval lengths by $\lambda(G)$. Since nested intervals have different lengths, we know that $\nu(G) \le \lambda(G)$ and this inequality may be strict (as in Fig. 1b). For each $k \ge 2$, $(k-1)$-LengthINT $\subsetneq$ k-LengthINT $\subsetneq$ k-NestedINT $\subsetneq$ $(k+1)$-NestedINT. Fishburn [10, Theorem 5, p. 177] shows that graphs G in 2-NestedINT have unbounded $\lambda(G)$, so 2-NestedINT $\not\subseteq$ k-LengthINT for each k. See Figure 2a.

Partial Representation Extension. This problem was introduced by Klavík et al. [19]. A *partial representation* $\mathcal{R}'$ of G is an interval representation $\{\langle x\rangle' : x \in V(G')\}$ of an induced subgraph G' of G. The vertices of G' and the intervals of $\mathcal{R}'$ are called *pre-drawn*. A representation $\mathcal{R}$ of G *extends* $\mathcal{R}'$ if and only if it assigns the same intervals to the vertices of G': $\langle x\rangle = \langle x\rangle'$ for every $x \in V(G')$.

Problem:	Partial Representation Extension – RepExt($\mathcal{C}$)
Input:	A graph G and a partial representation $\mathcal{R}'$ of an induced subgraph G'.
Output:	Is there an interval representation of $\mathcal{C}$ of G extending $\mathcal{R}'$?

An $\mathcal{O}(n^2)$-time algorithm for RepExt(INT) was given in [19]. There are two different linear-time algorithms [3, 18] for this problem. Minimal obstructions making partial representations non-extendible are described in [21]. A linear-time algorithm for proper interval graphs [16] and a quadratic-time algorithm for unit interval graphs [30] are known.

The partial representation extension problems were considered for other graph classes. Polynomial-time algorithms are known for circle graphs [7], and permutation and function graphs [15]. The problems are NP-hard for chordal graphs [17] and contact representations of planar graphs [6]. The complexity is open for circular-arc and trapezoid graphs.

Previous Results and Motivation. The classes k-LengthINT were introduced by Graham as a natural hierarchy between unit interval graphs and interval graphs; see Fig. 2a. Unfortunately, even after decades, the only results known are curiosities that illustrate the incredibly complex structure of k-LengthINT, very different from the case of unit interval graphs. For instance, k-LengthINT is not closed under disjoint unions; see Fig. 1b.

Leibowitz et al. [23] show that the class 2-LengthINT contains caterpillars, threshold, and unit interval graphs with one additional vertex. Further, there exist graphs G with $\lambda(G) > 2$ such that $\lambda(G \setminus x) \leq \lambda(G) - 2$ for $x \in V(G)$ [23]. Fishburn [9] shows that there are infinitely many forbidden induced subgraphs for 2-LengthINT (where UNIT INT are interval graphs just without $K_{1,3}$). It is also known [8] that there are graphs in 2-LengthINT such that, when the shorter length is fixed to 1, the longer one can be one of the real numbers belonging to arbitrary many distinct intervals of the real line, arbitrary far from each other.

Not much is known about the computational complexity of problems involving k-LengthINT, even recognition is open for $k = 2$. In [5], a polynomial-time algorithm is given for computing $\lambda(G)$ for interval graphs G which are extended bull-free or almost threshold (which highly restricts them). Skrien [29] characterized 2-LengthINT which can be realized by lengths zero (points) and one (unit intervals), leading to a linear-time recognition algorithm. As most of the efficient algorithms for intersection graph classes require representations, very little is known how to algorithmically use that a given interval graph can be represented by k lengths. In this paper, we show that partial representation extension is NP-hard already for 2-LengthINT.

All these difficulties lead us to introduce another hierarchy of k-NestedINT which generalizes proper interval graphs; see Fig. 2. We illustrate the nice structure of k-NestedINT by describing a relatively simple linear-time recognition algorithm which we also generalize to a more involved polynomial-time algorithm for partial representation extension. To the best of our knowledge, the only reference is Fishburn's book [10] in which the parameter $\nu(G)$ called *depth* is considered and linked to k-LengthINT. There are some different generalizations of proper interval graphs [26], which are less rich and not linked to k-LengthINT.

Since k-NestedINT seem to share many properties with proper interval graphs, several future directions of research are immediately offered. It should be possible to describe minimal forbidden induced subgraphs. For the computational problems which are tractable for proper interval graphs and hard for interval graphs, the complexity of the intermediate problems for k-NestedINT can be studied. (One such problem is FO property checking, discussed below.)

Our Results. In [14], a polynomial-time algorithm is given for recognizing 2-LengthINT when intervals are partitioned into two subsets A and B, each of one length, and both $G[A]$ and $G[B]$ are connected. This approach might be generalized for partial representation extension, but we show that removing the connectedness condition makes it hard:

▶ **Theorem 1.** *The problem* RepExt(2-LengthINT) *is* NP*-hard when every pre-drawn interval is of one length a. It remains* NP*-hard even when* (i) *the input prescribes two lengths $a = 1$ and b, and* (ii) *for every interval, the input assigns one of the lengths a or b. Also, it is* W[1]*-hard when parameterized by the number of pre-drawn intervals.*

In comparison, we give a dynamic programming algorithm for recognizing k-NestedINT, based on a data structure called an MPQ-tree. We show that we can optimize nesting greedily from the bottom to the top. We compute three different representations for each subtree and we show how to combine them.

▶ **Theorem 2.** *The minimum nesting number $\nu(G)$ can be computed in time $\mathcal{O}(n+m)$ where n is the number of vertices and m is the number of edges. Therefore, the problem* Recog(k-NestedINT) *can be solved in linear time.*

This result has the following application in the computational complexity of deciding logic formulas over graphs. Let φ be the length of a first-order logic formula for graphs. By the locality, this formula can be decided in G in time $n^{\mathcal{O}(\varphi)}$. Since it is W[2]-hard to decide it for general graphs when parameterized by φ, it is natural to ask for which graph classes there exists an FPT algorithm running in time $\mathcal{O}(n^c \cdot f(\varphi))$.

In [13], it is shown that the problem above is W[2]-hard even for interval graphs. On the other hand, if an interval graph is given together with a k-length interval representation, [13] gives an FPT algorithm with respect to the parameters φ and the particular lengths of the intervals. It was not clear whether such an algorithm is inherently geometrical. Recently, Gajarský et al. [12] give a different FPT algorithm for FO property testing for interval graphs parameterized by φ and the nesting k, assuming that a k-nested interval representation is given by the input. By our result, this assumption can be removed since we can compute an interval representation of the optimal nesting in linear time.

The problem RepExt(k-NestedINT) is more involved since a straightforward greedy optimization from the bottom to the top does not work. We describe a complex dynamic programming algorithm which computes with exponentially many possibilities efficiently.

▶ **Theorem 3.** *The problem* RepExt(k-NestedINT) *can be solved in polynomial time.*

The last result contrasts with Theorem 1. Partial representation extension of k-NestedINT and k-LengthINT are problems for which the geometrical version (at most k lengths) is much harder than the corresponding topological problem (the left-to-right ordering of endpoints of intervals). A generalization of partial representation extension called bounded representations is NP-complete for unit interval graphs [16], but polynomial-time solvable for proper interval graphs [2]. Partially embedded planar graphs can be extended in linear time [1], but extension of geometric straight-line embeddings is NP-hard [25].

2 Extending Partial Representations with Two Lengths

The complexity of recognizing k-LengthINT is a long-standing open problem, even for $k = 2$. In this section, we show that RepExt is NP-complete even when $k = 2$.

Proof of Theorem 1. Assume (i) and (ii). We adapt the reduction from 3-Partition used in [17, 16]. Let $A_1, \dots, A_{3s}$ and M be input of 3-Partition, with $\frac{M}{2} < A_i < \frac{M}{4}$ and $\sum A_i = Ms$. The task is to split A_i's into s triples, each summing to exactly M.

We solve this problem by constructing an interval graph G and a partial representation $\mathcal{R}'$ as depicted in Fig. 3. We claim that $\mathcal{R}'$ can be extended using two lengths of intervals if and only if the original instance of 3-Partition is solvable. We set $a = 1$ and $b = s \cdot (M+1)$. The partial representation $\mathcal{R}'$ consists of $s+1$ disjoint pre-drawn intervals $v_0, \dots, v_s$ of length a, with their pre-drawn positions $\ell'(v_i) = i \cdot (M+2)$. So they split the real line into s equal gaps of size $M+1$ and two infinite regions.

Aside $v_0, \dots, v_s$, the graph G contains exactly one vertex w of the length b, adjacent to every vertex in G. Further, for each A_i, the graph $G \setminus w$ contains P_{2A_i} (a path with $2A_i$ vertices) as one component, with each vertex of the length a.

This reduction is clearly polynomial. It remains to show that $\mathcal{R}'$ is extendible if and only if the instance of 3-Partition is solvable. First, because of the length constraint, in

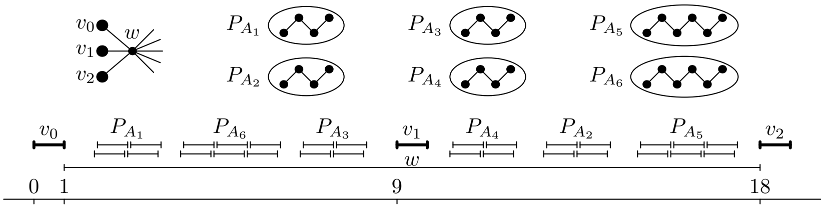


Figure 3 We consider the following input for 3-Partition: $s = 2$, $M = 7$, $A_1 = A_2 = A_3 = A_4 = 2$ and $A_5 = A_6 = 3$. The associated graph G is depicted on top, and at the bottom we find one of its extending representations, giving the 3-partitioning $\{A_1, A_3, A_6\}$ and $\{A_2, A_4, A_5\}$.

every extending representation has $\ell(w) = 1$; otherwise it would not intersect either v_0, or v_s. Therefore, each of the paths P_{2A_i} has to be placed in exactly one of the s gaps. In every representation of P_{2A_i}, it requires the space at least $A_i + \varepsilon$ for some $\varepsilon > 0$. Therefore, three paths can be packed into the same gap if and only if their three integers sum to at most M. Therefore, an extending representation $\mathcal{R}'$ gives a solution to 3-Partition, and vice versa. We get W[1]-hardness by a similar reduction from BinPacking, as in [17].

The above reduction can be easily modified when (i) and (ii) are avoided. We add two extra vertices: w_0 adjacent to v_0 and w_s adjacent to v_s such that both are non-adjacent to w. It forces the length of w to be in the interval $[s \cdot (M + 1), s \cdot (M + 1) + 2)$, so the length b does not have to be prescribed. Also, this reduction works even when each interval does not have a length assigned, aside the pre-drawn intervals in $\mathcal{R}'$ indeed. ◀

3 Basic Properties of k-Nested Interval Graphs

The nesting defines a partial ordering $\subsetneq$ of intervals and we denote by $\nu(u)$ the length of the longest chain of nested intervals ending with $\langle u \rangle$. By Pred(u), we denote the set of all direct predecessors of $\langle u \rangle$ in $\subsetneq$.

▶ **Lemma 4.** *An interval graph belongs to k-NestedINT if and only if it has an interval representation which can be partitioned into k proper interval representations.*

Proof. Let $\mathcal{R}$ be an interval representation partitioned into proper interval representations $\mathcal{R}_1, \dots, \mathcal{R}_k$. No chain of nested intervals contains two intervals from some $\mathcal{R}_i$, so the nesting is at most k. On the other hand, let $\mathcal{R}$ be a k-nested interval representation. We label each interval by the length of the longest chain of nested intervals ending in it; see Fig. 2b. Notice that the intervals of each label $i \in \{1, \dots, k\}$ form a proper interval representation $\mathcal{R}_i$. ◀

Interval graphs and the subclasses k-NestedINT and k-LengthINT are closed under induced subgraphs, so they can be characterized by minimal forbidden induced subgraphs. Lekkerkerker and Boland [24] describe them for interval graphs, and Roberts [27] proved that 1-NestedINT=1-LengthINT are claw-free interval graphs. On the other hand, 2-LengthINT have infinitely many minimal forbidden induced subgraphs [9] which are interval graphs.

4 Maximal Cliques and MPQ-trees

▶ **Lemma 5** (Fulkerson and Gross [11]). *A graph is an interval graph if and only if there exists a linear ordering $<$ of its maximal cliques such that, for each vertex, the maximal cliques containing this vertex appear consecutively.*

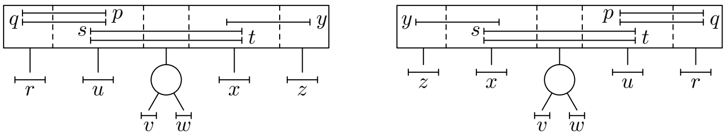


Figure 4 Two equivalent MPQ-trees with denoted sections. In all figures, we denote P-nodes by circles and Q-nodes by rectangles.

An ordering of the maximal cliques satisfying the statement of Lemma 5 is called a *consecutive ordering*. Each interval graph has $\mathcal{O}(n)$ maximal cliques of total size $\mathcal{O}(n+m)$ which can be found in linear time [28].

PQ-trees. A *PQ-tree* T is a rooted tree, introduced by Booth and Lueker [4]. Its leaves are in one-to-one correspondence with the maximal cliques. Its inner nodes are of two types: *P-nodes* and *Q-nodes*. Each P-node has at least two children, each Q-node at least three. The orderings of the children of inner nodes are given. The PQ-tree T represents one consecutive ordering $<_T$ called the *frontier* of T which is the ordering of the leaves from left to right.

The PQ-tree T represents all consecutive orderings of G as frontiers of equivalent PQ-trees which can be constructed from T by sequences of *equivalent transformations* of two types: (i) an arbitrary reordering of the children of a P-node, and (ii) a reversal of the order of the children of a Q-node; see Fig. 4. A subtree of T consists of a node and all its descendants. For a node N, its subtrees are the subtrees which have the children of N as the roots.

MPQ-trees. The *MPQ-tree* [22] is an augmentation of the PQ-tree in which the nodes of T have assigned subsets of $V(G)$ called *sections*. To a leaf representing a clique C, we assign one section $s(C)$. Similarly, to each P-node P, we assign one section $s(P)$. For a Q-node Q with subtrees $T_1, \dots, T_q$, we have q sections $s_1(Q), \dots, s_q(Q)$ ordered from left to right, each corresponding to one subtree. Examples of sections are depicted in Fig. 4.

The section $s(C)$ has all vertices contained in the maximal clique C and no other maximal clique. The section $s(P)$ of a P-node P has all vertices that are contained in all maximal cliques of the subtree rooted at P and in no other maximal clique. Let Q be a Q-node with subtrees $T_1, \dots, T_q$. Let x be a vertex contained only in maximal cliques of the subtree rooted at Q, contained in maximal cliques of at least two subtrees. Then x is contained in every section $s_i(Q)$ such that some maximal clique of T_i contains x.

Every vertex x is in sections of exactly one node of T. In the case of a Q-node, it is placed in consecutive sections of this node. For a Q-node Q, if x is placed in a section $s_i(Q)$, then x is contained in all cliques of T_i. Every section of a Q-node is non-empty, and two consecutive sections have a non-empty intersection.

5 Recognizing k-nested Interval Graphs

Our linear-time algorithm is a dynamic programming on the MPQ-tree. We process from the bottom to the top, and we optimize the minimal nesting.

Two vertices x and y are twins if and only if $N[x] = N[y]$. The standard observation is that twins can be ignored since they can be represented by identical intervals, and notice that this does not increase nesting and the number of lengths. We can locate all twins in time $\mathcal{O}(n+m)$ [28] and we can prune the graph by keeping one vertex per equivalence class of twins. So no two vertices belong to the exactly same sections of the MPQ-tree.

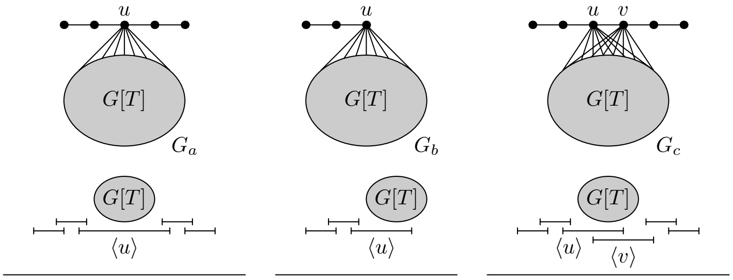


Figure 5 The graphs G_a, G_b and G_c with representations, defining the triple (a, b, c) of T. The vertices of $G[T]$ are adjacent to the added vertices u and v and not to the others.

Triples. We compute triples (a, b, c) for each subtree of the MPQ-tree: a is the optimal nesting of the entire subtree, b is the nesting when the subtree is placed on a side of its parent, and c is used for Q-nodes. We define for a subtree T a triple (a, b, c) as follows. Let $G[T]$ be the interval graph induced by the vertices of the sections of T. Let G_a, G_b and G_c be graphs as in Fig. 5. We define $a = \nu(G_a) - 1 = \nu(G[T])$, $b = \nu(G_b) - 1$, and $c = \nu(G_c) - 1$.

The triple can be interpreted as increase in the nesting, depending how $G[T]$ is represented with respect to the rest of the graph. We compute these triples from the leaves to the root, and output a of the root as $\nu(G)$.

▶ **Lemma 6.** *For any subtree T, its triple (a, b, c) satisfies $a - 1 \le b \le c \le a$.*

Proof. We prove equivalently that $\nu(G_a) - 1 \le \nu(G_b) \le \nu(G_c) \le \nu(G_a)$. We trivially know that $\nu(G_b) \le \nu(G_c)$ since G_b is an induced subgraph of G_c.

Our definition of G_a implies that $\nu(G_a) = \nu(G[T])+1$, since in every interval representation of G_a, both endpoints of $\langle u \rangle$ are covered by attached paths, and there the representation of $G[T]$ is nested in $\langle u \rangle$. Since $\nu(G_b) \ge \nu(G[T])$, the inequality $\nu(G_a) - 1 \le \nu(G_b)$ follows. Alternatively, for a representation of G_b optimizing nesting, we stretch $\langle u \rangle$ (which increases nesting by at most one) and add the second attached path, to get a representation of G_a.

It remains to show the last inequality that $\nu(G_c) \le \nu(G_a)$. Consider the representation of G_a with optimal nesting, in which $G[T]$ is strictly contained inside $\langle u \rangle$. By shifting $r(u)$ to the left and adding $\langle v \rangle$, we get a representation of G_c with the same nesting. ◀

Leaves. We initiate (a, b, c) for every leaf L of the MPQ-tree as follows. If $s(L) = \emptyset$, we put $(0, 0, 0)$; otherwise we put $(1, 0, 0)$. The reason is that the vertices of $s(L)$ form a clique.

P-nodes. Let $T_1, \dots, T_p$ be the children of a P-node P, with $p \ge 2$, with the computed triple (a_i, b_i, c_i) for each subtree T_i. We want to compute (a, b, c) for P. We first assume that $s(P) \ne \emptyset$. Let $w \in s(P)$, then the vertices of every subtree except two outer subtrees T_s and T_t are nested inside $\langle w \rangle$, so their minimal nesting number is increased by one. We can choose optimally two outer subtrees T_s and T_t, and their outer maximal cliques. Only the intervals contained in these two outer maximal cliques are not nested inside $\langle w \rangle$.

We get the following formulas (see Fig. 6):

$$\begin{aligned} a &= \begin{cases} \max\{a_1, \dots, a_p\}, & s(P) = \emptyset, \\ \min_{s \ne t} \max\{b_s, b_t, a_i : i \ne s, t\} + 1, & s(P) \ne \emptyset. \end{cases} \\ b &= \min_s \max\{b_s, a_i : i \ne s\}, \\ c &= \max\{a_1, \dots, a_p\}. \end{aligned}$$

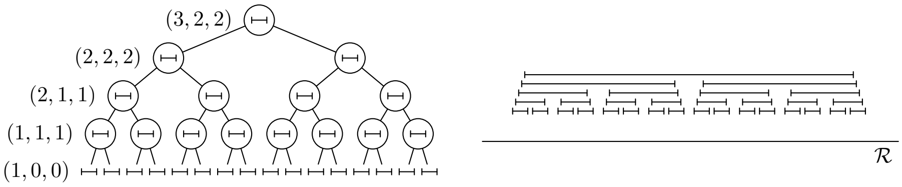


Figure 6 An MPQ-tree representing G with the computed triples (a, b, c) (equal on each level) and a representation minimizing nesting. We have that $\nu(G) = 3$.

▶ **Lemma 7.** *The formulas compute the triple (a, b, c) of a P-node P correctly.*

Proof. Let T be the subtree with P as the root. We know that $a = \nu(G[T])$. If $s(P) = \emptyset$, then $G[T]$ is the disjoint union of $G[T_1], \dots, G[T_p]$ with $a_i = \nu(G[T_i])$, so $a = \max\{a_1, \dots, a_p\}$. Otherwise, let $\{w\} = s(P)$. Vertices of every subtree except for two outer T_s and T_t, which we are free to choose, are completely nested inside; so we minimize over all possible choices of $T_s \neq T_t$. For every $i \neq s, t$, we have the optimal nesting a_i increased by one because of $\langle w \rangle$. Now, T_s can be placed on one side of $\langle w \rangle$, while the other side is blocked by the remaining vertices. Therefore, we can simulate this by G_b of $G[T_s]$, where u corresponds to w, so the optimal nesting of $G[T]$ is at least $b_s + 1$, and similarly for T_t and $b_t + 1$.

Concerning b, we have in every representation $\langle u \rangle$ with one side covered by the attached path. Therefore all vertices of $G[T]$ are either nested in $\langle u \rangle$, or intersect the other side. We can always place all vertices of $s(P)$ in such a way that they are not nested in $\langle u \rangle$, and so we can ignore them. All vertices of $G[T]$ except for one subtree T_s, which we can arbitrarily choose, have to be nested in $\langle u \rangle$. Therefore, their optimal nesting a_i is increased by one by $\langle u \rangle$. For T_s, the nesting again corresponds to the optimal nesting of G_b of $G[T_s]$, which is $b_s + 1$. Therefore we get $\nu(G_b) = \min_s \max\bigl(\{a_i + 1 : i \neq s\} \cup \{b_s + 1\}\bigr)$, which gives the formula for b since $b = \nu(G_b) - 1$. Concerning c, we have also $\langle v \rangle$. We can choose T_s and T_t so that T_s has optimized nesting with respect to $\langle u \rangle$ and T_t with respect to $\langle v \rangle$. But anyway all intervals of $G[T_s]$ are nested in $\langle v \rangle$ and all intervals of $G[T_t]$ are nested in $\langle u \rangle$. Therefore this optimization is useless and we just get $\nu(G_c) = \max\{a_1, \dots, a_p\} + 1$. ◀

▶ **Lemma 8.** *For a P-node with p children, the triple (a, b, c) can be computed in $\mathcal{O}(p)$.*

Proof. By Lemma 6, we always have either $b_i = a_i - 1$, or $b_i = a_i$. Only in the former case, we can optimize the nesting by choosing s or t equal i. We call these T_i *savable*. Concerning c, we just find the maximum a_i which can be done in time $\mathcal{O}(p)$. Concerning a and b, we first locate all T_i which maximize a_i. If at least one of them is not savable, say T_j, then $a = a_j + 1$ and $b = a_j$. Otherwise if all are savable, then the values depend on the number of T_i's maximizing a_i. If there are at most two, then $a = a_i$, otherwise $a = a_i + 1$. If there is at most one, then $b = b_i$, otherwise $b = b_i + 1$. ◀

Q-nodes. To optimize Q-nodes, the values c are also required. Let Q be a Q-node with children $T_1, \dots, T_q$, where $q \geq 3$, each with a triple (a_i, b_i, c_i). We first work with each section $s_i(Q)$ separately. For intervals of sections of Q, we define their *nesting* $\subsetneq$ as follows: $u \subsetneq v$ if v is contained in a section more to the left than all sections of u and in a section more to the right than all sections of u. (In every representation, $\langle u \rangle$ is contained in $\langle v \rangle$.)

For each subtree T_i, we choose whether to optimize its left side, or its right side. For the optimized side, the nesting is increased by b_i, while for the other side it is increased by c_i. We denote the increase in the nesting on the left side by $\bigcirc_i^{\leftarrow}$ and on the right side by $\bigcirc_i^{\rightarrow}$. So we choose either $\bigcirc_i^{\leftarrow} = b_i$ and $\bigcirc_i^{\rightarrow} = c_i$, or $\bigcirc_i^{\leftarrow} = c_i$ and $\bigcirc_i^{\rightarrow} = b_i$.

Suppose that we have some choice of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ for each subtree T_i. Let $\nu(x)$ be the length of the longest chain of nested intervals ending with $\langle x \rangle$, minimized over all possible interval representations of G. We compute the nesting $\nu(x)$ for each x belonging to sections of Q, from bottom to the top of $\subsetneq$. Suppose that we process some x and we know $\nu(y)$ for every direct predecessor of x, and let $s_s(Q)$ be the leftmost section of x and $s_t(Q)$ be the rightmost section. We get the following formula:

$$\nu(x) = \max\{\bigcirc_s^{\leftarrow}, \bigcirc_t^{\rightarrow}, a_i, \nu(y) : s < i < t \text{ and } y \in \mathrm{Pred}(x)\} + 1.$$

Then, we compute (a, b, c) as follows:

$$\begin{aligned} a &= \max\{a_1, \dots, a_q, \nu(y) : y \in \mathrm{Pred}(u)\}, \\ b^{\leftarrow} &= \max\{b_1, a_2, \dots, a_q, \nu(y) : y \in \mathrm{Pred}(u)\}, \\ b^{\rightarrow} &= \max\{a_1, \dots, a_{q-1}, b_q, \nu(y) : y \in \mathrm{Pred}(u)\}, \\ c &= \max\{a_1, \dots, a_q, \nu(y) : y \in \mathrm{Pred}(u) \text{ or } y \in \mathrm{Pred}(v)\}. \end{aligned}$$

We put $b = \min\{b^{\leftarrow}, b^{\rightarrow}\}$. For each of numbers a, b and c, we choose the minimum over all possible choices of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$.

▶ **Lemma 9.** *The formulas compute the triple (a, b, c) of a Q-node Q correctly.*

Proof. Notice that there exists a representation such that two intervals of sections of Q are nested as in the relation $\subsetneq$ defined above. For each subtree with some choice of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$, we have precomputed what is the optimal nesting and what is the length of a longest chain ending with a vertex not contained in the leftmost clique ($\bigcirc_i^{\leftarrow}$), and of a longest chain not ending with a vertex in the rightmost clique ($\bigcirc_i^{\rightarrow}$). The formulas extends these chains according to the vertices of sections of Q and of added vertices u and v in G_a, G_b, G_c. The length of a longest chain ending with u or v is used. ◀

▶ **Lemma 10.** *For each of a, $b^{\leftarrow}$, $b^{\rightarrow}$ and c, the choices of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ are done greedily.*

Proof. We argue for a, the argument is similar for the others. Notice that values $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ describe the starting length of some chains. We can easily compute how much are these chains extended by the intervals of sections of Q and by u. We find out which side needs to save more, for which we choose b_i, and for the other side we choose c_i. These values can be chosen for different subtrees independently since their chains start differently. ◀

▶ **Lemma 11.** *For a Q-node Q with q children, the triple (a, b, c) can be computed in time $\mathcal{O}(q + m_Q)$, where m_Q is the number of edges of $G[Q]$.*

Proof. For every interval u in sections of Q, we know its leftmost section and its rightmost section. We compute the DAG of nesting for all vertices of the sections of Q. This can be done by considering all edges, and testing for each whether the pair is nested.

For each of a, $b^{\leftarrow}$, $b^{\rightarrow}$, and c, we add vertex u (and possibly v) to this DAG, together with correct edges according to their nesting. Then for each vertex of this DAG, we compute the length of the longest chain starting in this vertex. This can be done in linear time by processing the DAG from top to the bottom. Next, we process the subtrees T_i. For each, we compute how much chains of length a_i, $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ are prolonged. Then we greedily choose $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ from b_i and c_i. The total consumed time is $\mathcal{O}(q + m_Q)$. ◀

Proof of Theorem 2. If a graph is an interval graph, we can compute its MPQ-tree in time $\mathcal{O}(n + m)$. Then we process the tree from the bottom to the root and compute triples (a, b, c)

for every node, as described above. We output a of the root which is the minimal nesting number $\nu(G)$. By Lemmas 7 and 9, this value is computed correctly. By Lemmas 8 and 11, the running time of the algorithm is $\mathcal{O}(n+m)$. ◀

6 Extending Partial Representations of Minimal Nesting

In this section, we sketch a polynomial-time algorithm of Theorem 3; see the full version [20].

Partial Ordering $\lhd$. The paper [18] characterizes all extendible partial representations in terms of consecutive orderings. A certain partial ordering $\lhd$ of maximal cliques is derived from the partial representation $\mathcal{R}'$.

▶ **Lemma 12** ([18]). *A partial representation $\mathcal{R}'$ is extendible if and only if there exists a consecutive ordering $<$ of the maximal cliques extending $\lhd$. The consecutive orderings of extending representations are exactly the consecutive ordering extending $\lhd$.*

General Idea. We want to minimize the nesting similarly as in Section 5. A partial representation $\mathcal{R}'$ poses three restrictions: (i) Some pre-drawn intervals can be nested in each other which increases the nesting. (ii) The consecutive ordering has to extend $\lhd$ which restricts the possible shuffling of subtrees. (iii) Some subtrees can be optimized differently depending whether they are placed on the left or on the right of its parent.

Concerning (iii), instead of a triple, we compute for every subtree T a tuple $(a, b^{\leftarrow}, c^{\rightarrow}, b^{\rightarrow}, c^{\leftarrow})$. These number are equal to minimum nestings of extending representations of partial representations of G_a, G_b and G_c, in which the pre-drawn vertices of $G[T]$ are pre-drawn the same and we have pre-drawn $\langle u\rangle'$ (and possibly $\langle v\rangle'$) with their added neighbors. The difference between $b^{\leftarrow}$ and $b^{\rightarrow}$ is whether $\langle u\rangle'$ covers the right side of $G[T]$, or covers the left side of $G[T]$; similarly for $c^{\leftarrow}$ and $c^{\rightarrow}$.

For (i), we compute for pre-drawn intervals $\langle x\rangle'$ their nesting $\nu(x)$ in an optimal extending representation of $\mathcal{R}'$ constructed by the algorithm, from bottom to the top as they are encountered in the MPQ-tree. We show that for each *variable* $\nu(x)$ and each $b^{\leftarrow}$, $c^{\rightarrow}$, $b^{\rightarrow}$ and $c^{\leftarrow}$, their values are determined up to one and we can force some subtrees to optimize them. Unfortunately, some of these values cannot be optimized simultaneously, so we introduce the concept of an optimization graph H. The variables are represented by vertices and we have an edge $xy \in E(H)$ if and only if they cannot be optimized at the same time. The optimizable subsets of variables are the independent sets in H.

P-nodes. We have a P-node P with subtrees $T_1, \dots, T_p$ with the computed tuple $(a_i, b_i^{\leftarrow}, c_i^{\rightarrow}, b_i^{\rightarrow}, c_i^{\leftarrow})$ and the optimization graph H_i for each subtree T_i and $\nu(x)$. Because of (ii), we get that T_s has to be one of the minimal elements $\min(\lhd)$ of $\lhd$ and T_t one of the maximal elements $\max(\lhd)$. Suppose that $T_s \in \min(\lhd)$ and $T_t \in \max(\lhd)$ is fixed.

We first compute $\nu(x)$ for intervals of $s(P)$, according to $\subsetneq$. Assuming that we know $\nu(y)$ for every $y \in \mathrm{Pred}(x)$, we compute $\nu(x)$ as follows:

$$\nu(x) = \max\bigl\{b_s^{\leftarrow}, b_t^{\rightarrow}, a_i, \nu(y) : i \neq s,t \text{ and } y \in \mathrm{Pred}(x)\bigr\} + 1.$$

Then, we compute $(a, b^{\leftarrow}, b^{\rightarrow}, c^{\rightarrow}, c^{\leftarrow})$:

$$\begin{aligned}
a &= \max\{a_1, \dots, a_p, \nu(y) : y \in \mathrm{Pred}(u)\}.\\
b^{\leftarrow} &= \max\{b_s^{\leftarrow}, a_i, \nu(y) : i \neq s \text{ and } y \in \mathrm{Pred}(u)\}.\\
c^{\rightarrow} &= \max\{b_t^{\rightarrow}, a_i, \nu(y) : i \neq t \text{ and } y \in \mathrm{Pred}(v)\}.\\
b^{\rightarrow} &= \max\{b_t^{\rightarrow}, a_i, \nu(y) : i \neq t \text{ and } y \in \mathrm{Pred}(u)\}.\\
c^{\leftarrow} &= \max\{b_s^{\leftarrow}, a_i, \nu(y) : i \neq s \text{ and } y \in \mathrm{Pred}(v)\}.
\end{aligned}$$

The value of a can be always obtained greedily. Since the values given by $T_1, \dots, T_n$ are not fully determined, we create a new optimization subgraph H of T as the disjoint union of $H_1, \dots, H_p$ together with new pre-drawn vertices of $s(P)$ and $b^{\leftarrow}$, $b^{\rightarrow}$, $c^{\leftarrow}$ and $c^{\rightarrow}$, and we add suitable edges.

We run the above computation over all choices of $T_s \in \min(\lhd)$ and $T_t \in \max(\lhd)$. Since $b^{\leftarrow}$ and $c^{\rightarrow}$ might be optimized using different subtrees T_s and T_t than $b^{\rightarrow}$ and $c^{\leftarrow}$, we argue that the situation is not very limited, so we can combine optimizations together in one H.

Q-node. We have a Q-node Q with subtrees $T_1, \dots, T_q$ with the computed tuple $(a_i, b_i^{\leftarrow}, c_i^{\rightarrow}, b_i^{\rightarrow}, c_i^{\leftarrow})$ and the optimization graph H_i for each subtree T_i and $\nu(x)$. Because of (ii), we get that the Q-node might be flippable (which greatly restricts it), or not. For a subtree T_i, we may optimize it with respect to its left side, which is encoded by $b_i^{\leftarrow}$ and $c_i^{\rightarrow}$, or with respect to the right side, which is encoded by $c_i^{\leftarrow}$ and $b_i^{\rightarrow}$. In the former case, we put $\bigcirc_i^{\leftarrow} = b_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow} = c_i^{\rightarrow}$. In the latter case, we put $\bigcirc_i^{\leftarrow} = c_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow} = b_i^{\rightarrow}$. We have trade-offs leading to exponentially many possibilities how $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ can be chosen, which we capture by the optimization graph H.

Suppose that we fix some choices of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$. Unlike for P-nodes, we compute $\nu(x)$ for all intervals of sections of Q, not just for the pre-drawn ones. For two such vertices, we define $\subsetneq$ by nesting of sections, similarly as in Section 5. We process them from the bottom to the top according to $\subsetneq$. Suppose that we process some x and we know $\nu(y)$ for every direct predecessor of x, and let $s_s(Q)$ be the leftmost section of x and $s_t(Q)$ be the rightmost section. We get the following formula:

$$\nu(x) = \max\bigl\{\bigcirc_s^{\leftarrow}, \bigcirc_t^{\rightarrow}, a_i, \nu(y) : s < i < t \text{ and } y \in \mathrm{Pred}(x)\bigr\} + 1.$$

Then, we compute $(a, b^{\leftarrow}, b^{\rightarrow}, c^{\rightarrow}, c^{\leftarrow})$ as follows:

$$\begin{aligned}
a &= \max\{a_1, \dots, a_q, \nu(y) : y \in \mathrm{Pred}(u)\}\\
b^{\leftarrow} &= \max\{b_1^{\leftarrow}, a_i, \nu(y) : i > 1 \text{ and } y \in \mathrm{Pred}(u)\}.\\
c^{\rightarrow} &= \max\{b_n^{\rightarrow}, a_i, \nu(y) : i < n \text{ and } y \in \mathrm{Pred}(v)\}.\\
b^{\rightarrow} &= \max\{b_n^{\rightarrow}, a_i, \nu(y) : i < n \text{ and } y \in \mathrm{Pred}(u)\}.\\
c^{\leftarrow} &= \max\{b_1^{\leftarrow}, a_i, \nu(y) : i > 1 \text{ and } y \in \mathrm{Pred}(v)\}.
\end{aligned}$$

By some involved structural properties, each variable is determined up to one. We encode all possibilities into the optimization graph H, similarly as in the case of P-nodes.

Putting Together. We construct a polynomial-time algorithm for RepExt(k-NestedINT) as follows. We process the MPQ-tree from the bottom to the top. We have to argue that each step can be computed in polynomial time (easy) and that it computes the tuples $(a, b^{\leftarrow}, b^{\rightarrow}, c^{\rightarrow}, c^{\leftarrow})$ and the optimization graph H correctly (involved).

Acknowledgment. We want to thank Takehiro Ito and Hirotaka Ono for discussions.

References

1 P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. *ACM Trans. Algor.*, 11(4):32:1–32:42, 2015.

2 M. Balko, P. Klavík, and Y. Otachi. Bounded representations of interval and proper interval graphs. In *ISAAC*, volume 8283 of *LNCS*, pages 535–546. Springer, 2013.

3 T. Bläsius and I. Rutter. Simultaneous pq-ordering with applications to constrained embedding problems. *ACM Trans. Algor.*, 12(2):16, 2016.

4 K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.

5 M. R. Cerioli, F. de S. Oliveira, and J. L. Szwarcfiter. On counting interval lengths of interval graphs. *Discrete Applied Mathematics*, 159(7):532–543, 2011.

6 S. Chaplick, P. Dorbec, J. Kratochvíl, M. Montassier, and J. Stacho. Contact representations of planar graphs: Extending a partial representation is hard. In *WG'14*, volume 8747 of *LNCS*, pages 139–151. Springer, 2014.

7 S. Chaplick, R. Fulek, and P. Klavík. Extending partial representations of circle graphs. In *Graph Drawing*, volume 8242 of *LNCS*, pages 131–142. Springer, 2013.

8 P. C. Fishburn. Paradoxes of two-length interval orders. *Discrete Math.*, 52(2):165–175, 1984.

9 P. C. Fishburn. Interval graphs and interval orders. *Discrete Math.*, 55(2):135–149, 1985.

10 P. C. Fishburn. *Interval orders and interval graphs: A study of partially ordered sets.* John Wiley & Sons, 1985.

11 D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.

12 J. Gajarský, D. Lokshtanov, J. Obdržálek, S. Ordyniak, M. S. Ramanujan, and S. Saurabh. FO model checking on posets of bounded width. In *FOCS 2015*, pages 963–974, 2015.

13 R. Ganian, P. Hlinený, D. Král, J. Obdržálek, J. Schwartz, and J. Teska. FO model checking of interval graphs. *Logical Methods in Computer Science*, 11(4:11):1–20, 2015.

14 F. Joos, C. Löwenstein, F. de S. Oliveira, D. Rautenbach, and J. L. Szwarcfiter. Graphs of interval count two with a given partition. *Inform. Process. Lett.*, 114(10):542–546, 2014.

15 P. Klavík, J. Kratochvíl, T. Krawczyk, and B. Walczak. Extending partial representations of function graphs and permutation graphs. In *ESA*, volume 7501 of *LNCS*, pages 671–682. Springer, 2012.

16 P. Klavík, J. Kratochvíl, Y. Otachi, I. Rutter, T. Saitoh, M. Saumell, and T. Vyskočil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 2016.

17 P. Klavík, J. Kratochvíl, Y. Otachi, and T. Saitoh. Extending partial representations of subclasses of chordal graphs. *Theoretical Computer Science*, 576:85–101, 2015.

18 P. Klavík, J. Kratochvíl, Y. Otachi, T. Saitoh, and T. Vyskočil. Extending partial representations of interval graphs. *Algorithmica*, 2016.

19 P. Klavík, J. Kratochvíl, and T. Vyskočil. Extending partial representations of interval graphs. In *TAMC*, volume 6648 of *LNCS*, pages 276–285. Springer, 2011.

20 P. Klavík, Y. Otachi, and J. Šejnoha. On the classes of interval graphs of limited nesting and count of lengths. *CoRR*, abs/1510.03998, 2015.

21 P. Klavík and M. Saumell. Minimal obstructions for partial representation extension of interval graphs. In *ISAAC*, volume 8889 of *LNCS*, pages 401–413, 2014.

22 N. Korte and R. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989.

23 R. Leibowitz, S.F. Assmann, and G.W. Peck. The interval count of a graph. *SIAM J. Algebr. Discrete Methods*, 3:485–494, 1982.

24 C. Lekkerkerker and D. Boland. Representation of finite graphs by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962.

25 M. Patrignani. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science*, 17(05):1061–1069, 2006.

26 A. Proskurowski and J. A. Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3(4):167–176, 1999.

27 F. S. Roberts. Indifference graphs. *Proof techniques in graph theory*, pages 139–146, 1969.

28 D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SICOMP*, 5(2):266–283, 1976.

29 D. Skrien. Chronological orderings of interval graphs. *Discrete Appl. Math.*, 8(1):69–83, 1984.

30 F. J. Soulignac. Bounded, minimal, and short representations of unit interval and unit circular-arc graphs. *CoRR*, abs/1408.3443, 2014.

Pattern Matching and Consensus Problems on Weighted Sequences and Profiles*

Tomasz Kociumaka[1], **Solon P. Pissis**[2], **and Jakub Radoszewski**[†3]

1 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
kociumaka@mimuw.edu.pl
2 **Department of Informatics, King's College London, London, UK**
solon.pissis@kcl.ac.uk
3 **Institute of Informatics, University of Warsaw, Warsaw, Poland; and Department of Informatics, King's College London, London, UK**
jrad@mimuw.edu.pl

Abstract

We study pattern matching problems on two major representations of uncertain sequences used in molecular biology: weighted sequences (also known as position weight matrices, PWM) and profiles (i.e., scoring matrices). In the simple version, in which only the pattern or only the text is uncertain, we obtain efficient algorithms with theoretically-provable running times using a variation of the lookahead scoring technique. We also consider a general variant of the pattern matching problems in which both the pattern and the text are uncertain. Central to our solution is a special case where the sequences have equal length, called the consensus problem. We propose algorithms for the consensus problem parameterized by the number of strings that match one of the sequences. As our basic approach, a careful adaptation of the classic meet-in-the-middle algorithm for the knapsack problem is used. On the lower bound side, we prove that our dependence on the parameter is optimal up to lower-order terms conditioned on the optimality of the original algorithm for the knapsack problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases weighted sequence, position weight matrix, profile matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.46

1 Introduction

We study two well-known representations of uncertain texts: *weighted sequences* and *profiles*. A *weighted sequence* (also known as position weight matrix, PWM) for every position and every letter of the alphabet specifies the probability of occurrence of this letter at this position; see Table 1 for an example. A weighted sequence represents many different strings, each with the probability of occurrence equal to the product of probabilities of its letters at subsequent positions of the weighted sequence. Usually a threshold $\frac{1}{z}$ is specified, and one considers only strings that match the weighted sequence with probability at least $\frac{1}{z}$. A *scoring matrix* (or a profile) of length m is an $m \times \sigma$ matrix. The *score* of a string of length m is the sum of scores in the scoring matrix of the subsequent letters of the string at the respective positions. A string is said to match a scoring matrix if its matching score is above a specified threshold Z.

[*] Work supported by the Polish Ministry of Science and Higher Education under the 'Iuventus Plus' program in 2015–2016 grant no 0392/IP3/2015/73.
[†] The author is a Newton Fellow.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 46; pp. 46:1–46:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 A weighted sequence X of length 4 over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$X[1]$	$X[2]$	$X[3]$	$X[4]$
$\pi_1^{(X)}(\mathtt{a}) = 1/2$ $\pi_1^{(X)}(\mathtt{b}) = 1/2$	$\pi_2^{(X)}(\mathtt{a}) = 1$ $\pi_2^{(X)}(\mathtt{b}) = 0$	$\pi_3^{(X)}(\mathtt{a}) = 3/4$ $\pi_3^{(X)}(\mathtt{b}) = 1/4$	$\pi_4^{(X)}(\mathtt{a}) = 0$ $\pi_4^{(X)}(\mathtt{b}) = 1$

Weighted Pattern Matching and Profile Matching. First of all, we study the standard variants of pattern matching problems on weighted sequences and profiles, in which only the pattern or the text is an uncertain sequence. In the best-known formulation of the Weighted Pattern Matching problem, we are given a weighted sequence of length n, called a text, a solid (standard) string of length m, called a pattern, both over an alphabet of size σ, and a *threshold probability* $\frac{1}{z}$. We are asked to find all positions in the text where the fragment of length m represents the pattern with probability at least $\frac{1}{z}$. Each such position is called an *occurrence* of the pattern in the text; we also say that the fragment of the text and the pattern *match*. The Weighted Pattern Matching problem can be solved in $\mathcal{O}(\sigma n \log m)$ time via the Fast Fourier Transform [5]. In a more general, indexing variant of the problem, considered in [1, 12], one can preprocess a weighted text in $\mathcal{O}(nz^2 \log z)$ time to report all *occ* occurrences of a given solid pattern of length m in $\mathcal{O}(m + occ)$ time. (A similar indexing data structure, which assumes $z = \mathcal{O}(1)$, was presented in [4].) Very recently, the index construction time was reduced to $\mathcal{O}(nz)$ for constant-sized alphabets [2].

In the classic Profile Matching problem, the pattern is an $m \times \sigma$ profile, the text is a solid string of length n, and our task is to find all positions in the text where the fragment of length m has a score which is at least Z. A naïve approach to the Profile Matching problem works in $\mathcal{O}(nm + m\sigma)$ time. A broad spectrum of heuristics improving this algorithm in practice is known; for a survey see [16]. One of the principal techniques, coming in different flavours, is *lookahead scoring* that consists in checking if a partial match could possibly be completed by the highest scoring letters in the remaining positions of the scoring matrix and, if not, pruning the naïve search. The Profile Matching problem can also be solved in $\mathcal{O}(\sigma n \log m)$ time via the Fast Fourier Transform [17].

Weighted Consensus and Profile Consensus. As our most involved contribution, we study a general variant of pattern matching on weighted sequences and the consensus problems on uncertain sequences, which are closely related to the Multichoice Knapsack problem. In the Weighted Consensus problem, given two weighted sequences of the same length, we are to check if there is a string that matches each of them with probability at least $\frac{1}{z}$. A routine to compare user-entered weighted sequences with existing weighted sequences in the database is used, e.g., in JASPAR[1], a well-known database of PWMs. In the General Weighted Pattern Matching (GWPM) problem, both the pattern and the text are weighted. In the most common definition of the problem (see [3, 12]), we are to find all fragments of the text that give a positive answer to the Weighted Consensus problem with the pattern. The authors of [3] proposed an algorithm for the GWPM problem based on the weighted prefix table that works in $\mathcal{O}(nz^2 \log z + n\sigma)$ time. Solutions to these problems can be applied in transcriptional regulation: motif and regulatory module finding; and annotation of regulatory genomic regions.

In an analogous way to the Weighted Consensus problem, we define the Profile Consensus problem. Here we are to check for the existence of a string that matches both

[1] `http://jaspar.genereg.net`

the scoring matrices above threshold Z. The Profile Consensus problem is actually a special case of the well-known (especially in practice) Multichoice Knapsack problem (also known as the Multiple Choice Knapsack problem). In this problem, we are given n classes $C_1, \dots, C_n$ of at most λ items each—N items in total—each item c characterized by a value $v(c)$ and a weight $w(c)$. The goal is to select one item from each class so that the sums of values and of weights of the items are below two specified thresholds, V and W. (In the more intuitive formulation of the problem, we require the sum of values to be *above* a specified threshold, but here we consider an equivalent variant in which both parameters are symmetric.) The Multichoice Knapsack problem is widely used in practice, but most research concerns approximation or heuristic solutions; see [14] and references therein. As far as exact solutions are concerned, the classic meet-in-the middle approach by Horowitz and Sahni [11], originally designed for the (binary) Knapsack problem, immediately generalizes to an $\mathcal{O}^*(\lambda^{\lceil \frac{n}{2} \rceil})$-time[2] solution for Multichoice Knapsack.

Several important problems can be expressed as special cases of the Multichoice Knapsack problem using folklore reductions (see [14]). This includes the Subset Sum problem, which, for a set of n integers, asks whether there is a subset summing up to a given integer Q, and the k-Sum problem which, for $k = \mathcal{O}(1)$ classes of λ integers, asks to choose one element from each class so that the selected integers sum up to zero. These reductions give immediate hardness results for the Multichoice Knapsack problem, and they can be adjusted to yield the same consequences for Profile Consensus. For the Subset Sum problem, as shown in [7, 10], the existence for every $\varepsilon > 0$ of an $\mathcal{O}^*(2^{\varepsilon n})$-time solution would violate the Exponential Time Hypothesis (ETH) [13, 15]. Moreover, the $\mathcal{O}^*(2^{n/2})$ running time, achieved in [11], has not been improved yet despite much effort. The 3-Sum conjecture [9] and the more general k-Sum conjecture state that the 3-Sum and k-Sum problems cannot be solved in $\mathcal{O}(\lambda^{2-\varepsilon})$ time and $\mathcal{O}(\lambda^{\lceil \frac{k}{2} \rceil (1-\varepsilon)})$ time, respectively, for any $\varepsilon > 0$.

Our Results. As the first result, we show how the lookahead scoring technique combined with a data structure for answering longest common prefix (LCP) queries in a string can be applied to obtain simple and efficient algorithms for the standard pattern matching problems on uncertain sequences. For a weighted sequence, by R we denote the size of its list representation, and by λ the maximal number of letters with score at least $\frac{1}{z}$ at a single position (thus $\lambda \le \min(\sigma, z)$). In the Profile Matching problem, we set M as the number of strings that match the scoring matrix with score above Z. In general $M \le \sigma^m$, however, we may assume that for practical data this number is actually much smaller. We obtain the following running times:

- $\mathcal{O}(m\sigma + n \log M)$ for Profile Matching;
- $\mathcal{O}(R \log^2 \log \lambda + n \log z)$ deterministic and $\mathcal{O}(R + n \log z)$ randomized (Las Vegas, failure with probability R^{-c} for any given constant c) for Weighted Pattern Matching.

The more complex part of our study is related to the consensus problems and to the GWPM problem. Instead of considering Profile Consensus, we study the more general Multichoice Knapsack. We introduce parameters based on the number of solutions with *feasible* weight or value: $A_V = |\{(c_1, \dots, c_n) : c_i \in C_i \text{ for all } i = 1, \dots, n, \sum_i v(c_i) \le V\}|$, that is, the number of choices of one element from each class that satisfy the value threshold; $A_W = |\{(c_1, \dots, c_n) : c_i \in C_i \text{ for all } i = 1, \dots, n, \sum_i w(c_i) \le W\}|$; $A = \max(A_V, A_W)$, and $a = \min(A_V, A_W)$. We obtain algorithms with the following complexities:

[2] The $\mathcal{O}^*$ notation suppresses factors polynomial with respect to the instance size (encoded in binary).

- $\mathcal{O}(N+\sqrt{a\lambda}\log A)$ for Multichoice Knapsack;
- $\mathcal{O}(R+\sqrt{z\lambda}(\log\log z+\log\lambda))$ for Weighted Consensus and $\mathcal{O}(n\sqrt{z\lambda}(\log\log z+\log\lambda))$ for General Weighted Pattern Matching.

Note that $a \le A \le \lambda^n$ and thus the running time of our algorithm for Multichoice Knapsack is bounded by $\mathcal{O}(N+n\lambda^{(n+1)/2}\log\lambda)$. Up to lower order terms (i.e., the factor $n\log\lambda = (\lambda^{(n+1)/2})^{o(1)}$), this matches the time complexities of the fastest known solutions for both Subset Sum (also binary Knapsack) and 3-Sum. The main novel part of our algorithm for Multichoice Knapsack is an appropriate (yet intuitive) notion of *ranks* of partial solutions. We also provide a simple reduction from Multichoice Knapsack to Weighted Consensus, which lets us transfer the negative results to the GWPM problem.

- The existence, for every $\varepsilon > 0$, of an $\mathcal{O}^*(z^{\varepsilon})$-time solution for Weighted Consensus would violate the Exponential Time Hypothesis.
- For every $\varepsilon > 0$, an $\mathcal{O}^*(z^{0.5-\varepsilon})$-time solution for Weighted Consensus would imply an $\mathcal{O}^*(2^{(0.5-\varepsilon)n})$-time algorithm for Subset Sum.
- For every $\varepsilon > 0$, an $\tilde{\mathcal{O}}(R+z^{0.5}\lambda^{0.5-\varepsilon})$-time[3] solution for Weighted Consensus would imply an $\tilde{\mathcal{O}}(\lambda^{2-\varepsilon})$-time algorithm for 3-Sum.

For the higher-order terms our complexities match the conditional lower bounds; therefore, we put significant effort to keep the lower order terms of the complexities as small as possible.

Model of Computations. For problems on weighted sequences, we assume the word-RAM model with word size $w = \Omega(\log n + \log z)$ and $\sigma = n^{\mathcal{O}(1)}$. We consider the log-probability model of representations of weighted sequences, that is, we assume that probabilities in the weighted sequences and the threshold probability $\frac{1}{z}$ are all of the form $c^{\frac{p}{2^{dw}}}$, where c and d are constants and p is an integer that fits in a constant number of machine words. Additionally, the probability 0 has a special representation. The only operations on probabilities in our algorithms are multiplications and divisions, which can be performed exactly in $\mathcal{O}(1)$ time in this model. Our solutions to the Multichoice Knapsack problem only assume the word-RAM model with word size $w = \Omega(\log S + \log a)$, where S is the sum of integers in the input instance; this does not affect the $\mathcal{O}^*$ running time.

Structure of the Paper. We start with Preliminaries, where we formally introduce the problems and the main notions used throughout the paper. The following three sections describe our algorithms: in Section 3 for Profile Matching and Weighted Pattern Matching; in Section 4 for Profile Consensus; and in Section 5 for Weighted Consensus and General Weighted Pattern Matching. We conclude with a few remarks in Section 6.

2 Preliminaries

Let $\Sigma = \{s_1, s_2, \dots, s_\sigma\}$ be an alphabet of size σ. A *string* S over Σ is a finite sequence of letters from Σ. We denote the length of S by $|S|$ and, for $1 \le i \le |S|$, the i-th letter of S by $S[i]$. By $S[i..j]$ we denote the string $S[i]\dots S[j]$ called a *factor* of S (if $i > j$, then the factor is an empty string). A factor is called a *prefix* if $i = 1$ and a *suffix* if $j = |S|$. For two strings S and T, we denote their concatenation by $S \cdot T$ (ST in short).

[3] The $\tilde{\mathcal{O}}$ notation ignores factors polylogarithmic with respect to the input size.

For a string S of length n, by $lcp(i,j)$ we denote the length of the longest common prefix of factors $S[i..n]$ and $S[j..n]$. The following fact specifies a well-known efficient data structure answering such queries. It consists of the suffix array with its inverse, the LCP table and a data structure for range minimum queries on the LCP table; see [6] for details.

▶ **Fact 1.** *Let S be a string of length n over an alphabet of size $\sigma = n^{\mathcal{O}(1)}$. After $\mathcal{O}(n)$-time preprocessing, given indices i and j ($1 \le i, j \le n$) one can compute $lcp(i,j)$ in $\mathcal{O}(1)$ time.*

The *Hamming distance* between two strings X and Y of the same length, denoted by $d_H(X,Y)$, is the number of positions where the strings have different letters.

2.1 Profiles

In the Profile Matching problem, we consider a *scoring matrix* (a profile) P of size $m \times \sigma$. For $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,\sigma\}$, we denote the integer score of the letter s_j at the position i by $P[i,s_j]$. The *matching score* of a string S of length m with the matrix P is

$$\mathrm{Score}(S,P) = \sum_{i=1}^{m} P[i,S[i]].$$

If $\mathrm{Score}(S,P) \ge Z$ for an integer *threshold* Z, then we say that the string S *matches the matrix* P *above threshold* Z. We denote the number of strings S that match P above threshold Z by $\mathrm{NumStrings}_Z(P)$.

For a string T and a scoring matrix P, we say that P *occurs in* T *at position* i *with threshold* Z if $T[i..i+m-1]$ matches P above threshold Z. Then $Occ_Z(P,T)$ is the set of all positions where P occurs in T. These notions let us define the Profile Matching problem:

Profile Matching Problem
Input: A string T of length n, a scoring matrix P of size $m \times \sigma$, and a threshold Z.
Output: The set $Occ_Z(P,T)$.
Parameters: $M = \mathrm{NumStrings}_Z(P)$.

2.2 Weighted Sequences

A *weighted sequence* $X = X[1]\dots X[n]$ of length $|X| = n$ over alphabet $\Sigma = \{s_1, s_2, \dots, s_\sigma\}$ is a sequence of sets of pairs of the form $X[i] = \{(s_j,\ \pi_i^{(X)}(s_j))\ :\ j \in \{1,2,\dots,\sigma\}\}$. Here, $\pi_i^{(X)}(s_j)$ is the occurrence probability of the letter s_j at the position $i \in \{1,\dots,n\}$. These values are non-negative and sum up to 1 for a given i.

For all our algorithms, it is sufficient that the probabilities sum up to *at most* 1 for each position. Also, the algorithms sometimes produce auxiliary weighted sequences with sum of probabilities being smaller than 1 on some positions.

We denote the maximum number of letters occurring at a single position of the weighted sequence (with non-zero probability) by λ and the total size of the representation of a weighted sequence by R. The standard representation consists of n lists with up to λ elements each, so $R = \mathcal{O}(n\lambda)$. However, the lists can be shorter in general. Also, if the threshold probability $\frac{1}{z}$ is specified, at each position of a weighted sequence it suffices to store letters with probability at least $\frac{1}{z}$, and clearly there are at most z such letters for each position. This reduction can be performed in linear time, so we shall always assume that $\lambda \le z$.

The *probability of matching* of a string S with a weighted sequence X, $|S| = |X| = m$, is

$$\mathcal{P}(S,X) = \prod_{i=1}^{m} \pi_i^{(X)}(S[i]).$$

We say that a string S *matches a weighted sequence X with probability at least $\frac{1}{z}$*, denoted by $S \approx_{\frac{1}{z}} X$, if $\mathcal{P}(S, X) \geq \frac{1}{z}$. Given a weighted sequence T, by $T[i..j]$ we denote weighted sequence, called a *factor* of T, equal to $T[i] \dots T[j]$ (if $i > j$, then the factor is empty). We say that a string P *occurs* in T at position i if P matches the factor $T[i..i+m-1]$. The set of positions where P occurs in T is denoted by $Occ_{\frac{1}{z}}(P, T)$.

WEIGHTED PATTERN MATCHING PROBLEM
Input: A string P of length m and a weighted sequence T of length n with at most λ letters at each position and R in total, and a threshold probability $\frac{1}{z}$.
Output: The set $Occ_{\frac{1}{z}}(P, T)$.

3 Profile Matching and Weighted Pattern Matching

In this section we present a solution to the PROFILE MATCHING problem. Afterwards, we show that it can be applied for WEIGHTED PATTERN MATCHING as well.

For a scoring matrix P, the *heavy string* of P, denoted $\mathcal{H}(P)$, is constructed by choosing at each position the heaviest letter, that is, the letter with the maximum score (breaking ties arbitrarily). Intuitively, $\mathcal{H}(P)$ is a string that matches P with the maximum score.

▶ **Observation 2.** *If we have* $\text{Score}(S, P) \geq Z$ *for a string S of length m and an $m \times \sigma$ scoring matrix P, then $d_H(\mathcal{H}(P), S) \leq \lfloor \log M \rfloor$ where $M =$* $\text{NumStrings}_Z(P)$*.*

Proof. Let $d = d_H(\mathcal{H}(P), S)$. We can construct 2^d strings of length $|S|$ that match P with a score above Z by taking either of the letters $S[j]$ or $\mathcal{H}(P)[j]$ at each position j such that $S[j] \neq \mathcal{H}(P)[j]$. Hence, $2^d \leq M$, which concludes the proof. ◀

Our solution for the PROFILE MATCHING problem works as follows. We first construct $P' = \mathcal{H}(P)$ and the data structure for finding lcp values between suffixes of P' and T. Let the variable s store the matching score of P'. In the p-th step, we calculate the matching score of $T[p..p+m-1]$ by iterating through subsequent mismatches between P' and $T[p..p+m-1]$ and making adequate updates in the matching score s', which starts at $s' = s$. The mismatches are found using lcp-queries. This process terminates when the score s' drops below Z or when all the mismatches have been found. In the end, we include p in $Occ_Z(P, T)$ if $s' \geq Z$. This gives the following result.

▶ **Theorem 3.** PROFILE MATCHING *problem can be solved in $\mathcal{O}(m\sigma + n \log M)$ time.*

Proof. Let us bound the time complexity of the presented algorithm. The heavy string P' can be computed in $\mathcal{O}(m\sigma)$ time. The data structure for *lcp*-queries in $P'T$ can be constructed in $\mathcal{O}(n + m)$ time by Fact 1. Each query for $lcp(P'[i..m], T[j..n])$ can then be answered in constant time by a corresponding *lcp*-query in $P'T$, potentially truncated to the end of P'. Finally, for each position p in the text T we will consider at most $\lfloor \log M \rfloor + 1$ mismatches between P' and T, as afterwards the score s' drops below Z due to Observation 2. ◀

Basically the same approach can be used for WEIGHTED PATTERN MATCHING. In a natural way, we extend the notion of a heavy string to weighted sequences. Now we can restate Observation 2 in the language of probabilities instead of scores:

▶ **Observation 4.** *If a string P matches a weighted sequence X of the same length with probability at least $\frac{1}{z}$, then $d_H(\mathcal{H}(X), P) \leq \lfloor \log z \rfloor$.*

Comparing to the solution to Profile Matching, we compute the heavy string of the text instead of the pattern. An auxiliary variable α stores the matching probability between a factor of $\mathcal{H}(T)$ and the corresponding factor of T; it is updated when we move to the next position of the text. The rest of the algorithm is basically the same as previously. In the implementation, we perform the following operations on a weighted sequence:

- computing the probability of a given letter at a given position,
- finding the letter with the maximum probability at a given position.

In the standard list representation, the latter can be performed on a single weighted sequence in $\mathcal{O}(1)$ time after $\mathcal{O}(R)$-time preprocessing. We can perform the former in constant time if, in addition to the list representation, we store the letter probabilities in a dictionary implemented using perfect hashing [8] (we build a single hash table for all positions). This way, we can implement the algorithm in $\mathcal{O}(n \log z + R)$ time w.h.p. Alternatively, deterministic dictionaries [18, Theorem 3] (one for each position) can be used to obtain a deterministic solution in $\mathcal{O}(R \log^2 \log \lambda + n \log z)$ time. We arrive at the following result.

▶ **Theorem 5.** Weighted Pattern Matching *can be solved in* $\mathcal{O}(R + n \log z)$ *time with high probability by a Las-Vegas algorithm or in* $\mathcal{O}(R \log^2 \log \lambda + n \log z)$ *time deterministically.*

▶ Remark. In the same complexity one can solve the GWPM problem with a solid text.

4 Profile Consensus as Multichoice Knapsack

Let us start with a precise statement of the Multichoice Knapsack problem.

Multichoice Knapsack Problem
Input: A set $\mathcal{C}$ of N items partitioned into n disjoint classes C_i, each of size at most λ, two integers $v(c)$ and $w(c)$ for each item $c \in \mathcal{C}$, and two thresholds V and W.
Question: Does there exist a *choice* S (a set $S \subseteq \mathcal{C}$ such that $|S \cap C_i| = 1$ for each i) satisfying both $\sum_{c\in S} v(c) \le V$ and $\sum_{c\in S} w(c) \le W$?
Parameters: A_V and A_W: the number of choices S satisfying $\sum_{c\in S} v(c) \le V$ and $\sum_{c\in S} w(c) \le W$, respectively; as well as $A = \max(A_V, A_W)$ and $a = \min(A_V, A_W)$.

Indeed, we see that the Profile Consensus problem reduces to the Multichoice Knapsack problem. For two $m \times \sigma$ scoring matrices, we construct $n = m$ classes of $\lambda = \sigma$ items each, with values equal to the negated scores of the letters in the first matrix and weights equal to the negated scores in the second matrix; both thresholds V and W are equal to $-Z$.

For a fixed instance of Multichoice Knapsack, we say that S is a *partial choice* if $|S \cap C_i| \le 1$ for each class. The set $D = \{i : |S \cap C_i| = 1\}$ is called its *domain*. For a partial choice S, we define $v(S) = \sum_{c\in S} v(c)$ and $w(S) = \sum_{c\in S} w(c)$.

The classic $\mathcal{O}(2^{n/2})$-time solution to the Knapsack problem [11] partitions $D = \{1, \dots, n\}$ into two domains D_1, D_2 of size roughly $n/2$, and for each D_i it generates all partial choices S ordered by $v(S)$. Hence, it reduces the problem to an instance of Multichoice Knapsack with two classes. It is solved using the following folklore lemma.

▶ **Lemma 6.** *The* Multichoice Knapsack *problem can be solved in* $\mathcal{O}(N)$ *time if* $n = 2$ *and the elements* c *of* C_1 *and* C_2 *are sorted by* $v(c)$*.*

The same approach generalizes to Multichoice Knapsack. The partition is chosen to balance the number of partial choices in each domain, and the worst-case time complexity is $\mathcal{O}(\sqrt{Q\lambda})$, where $Q = \prod_{i=1}^{n} |C_i|$ is the number of choices.

Our aim in this section is to replace Q with the parameter a (which never exceeds Q). The overall running time is going to be $\mathcal{O}(N + \sqrt{a\lambda}\log A)$.

Two challenges arise when adapting the meet-in-the-middle approach: how to restrict the set of partial choices to be generated so that a feasible solution is not missed, and how to define a partition $D = D_1 \cup D_2$ to balance the number of partial choices generated for D_1 and D_2. A natural idea to deal with the first issue is to consider only partial choices with small values $v(S)$ or $w(S)$. This is close to our actual solution, which is based on the notion of *ranks* of partial choices. Our approach to the second problem is to consider multiple partitions: those of the form $D = \{1, \dots, j\} \cup \{j+1, \dots, n\}$ for $1 \le j \le n$. This results in an extra $\mathcal{O}(n)$ factor in the time complexity. However, preprocessing can assure $n = \mathcal{O}(\frac{\log A}{\log \lambda})$. While dealing with these two issues, a careful implementation is required to avoid several further extra factors in the running time. In case of our algorithm, this is only $\mathcal{O}(\log \lambda)$, which stems from the fact that we need to keep partial solutions ordered by $v(S)$.

For a partial choice S, we define $\operatorname{rank}_v(S)$ as the number of partial choices S' with the same domain for which $v(S') \le v(S)$. We symmetrically define $\operatorname{rank}_w(S)$. Ranks are introduced as an analogue of match probabilities in weighted sequences. Probabilities are multiplicative, while for ranks we have submultiplicativity:

▶ **Fact 7.** *If $S = S_1 \cup S_2$ is a decomposition of a partial choice S into two disjoint subsets, then $\operatorname{rank}_v(S_1)\operatorname{rank}_v(S_2) \le \operatorname{rank}_v(S)$ (and same for rank_w).*

Proof. Let D_1 and D_2 be the domains of S_1 and S_2, respectively. For every partial choices S'_1 over D_1 and S'_2 over D_2 such that $v(S'_1) \le v(S_1)$ and $v(S'_2) \le v(S_2)$, we have $v(S'_1 \cup S'_2) = v(S'_1) + v(S'_2) \le v(S)$. Hence, $S'_1 \cup S'_2$ must be counted while determining $\operatorname{rank}_v(S)$. ◀

For $0 \le j \le n$, let $\mathcal{L}_j$ be the list of partial choices with domain $\{1, \dots, j\}$ ordered by value $v(S)$, and for $\ell > 0$ let $V^{(\ell)}_{\mathcal{L}_j}$ be the value $v(S)$ of ℓ-th element of $\mathcal{L}_j$ (∞ if $\ell > |\mathcal{L}_j|$). Analogously, for $1 \le j \le n+1$, we define $\mathcal{R}_j$ as the list of partial choices over $\{j, \dots, n\}$ ordered by $v(S)$, and for $r > 0$, $V^{(r)}_{\mathcal{R}_j}$ as the value of the r-th element of $\mathcal{R}_j$ (∞ if $r > |\mathcal{R}_j|$).

The following two observations yield a decomposition of each choice into a single item and two partial solutions of a small rank. In particular, we do not need to know A_V in order to check if the ranks are sufficiently large.

▶ **Lemma 8.** *Let ℓ and r be positive integers such that $V^{(\ell)}_{\mathcal{L}_j} + V^{(r)}_{\mathcal{R}_{j+1}} > V$ for each $0 \le j \le n$. For every choice S with $v(S) \le V$, there is an index $j \in \{1, \dots, n\}$ and a decomposition $S = L \cup \{c\} \cup R$ such that $v(L) < V^{(\ell)}_{\mathcal{L}_{j-1}}$, $c \in C_j$, and $v(R) < V^{(r)}_{\mathcal{R}_{j+1}}$.*

Proof. Let $S = \{c_1, \dots, c_n\}$ with $c_i \in C_i$ and, for $0 \le i \le n$, let $S_i = \{c_1, \dots, c_i\}$. If $v(S_{n-1}) < V^{(\ell)}_{\mathcal{L}_{n-1}}$, we set $L = S_{n-1}$, $c = c_n$, and $R = \emptyset$, satisfying the claimed conditions.

Otherwise, we define j as the smallest index i such that $v(S_i) \ge V^{(\ell)}_{\mathcal{L}_i}$, and we set $L = S_{j-1}$, $c = c_j$, and $R = S \setminus S_j$. The definition of j implies $v(L) < V^{(\ell)}_{\mathcal{L}_{j-1}}$ and $v(L \cup \{c\}) \ge V^{(\ell)}_{\mathcal{L}_j}$. Moreover, we have $v(L \cup \{c\}) + v(R) = v(S) \le V < V^{(\ell)}_{\mathcal{L}_j} + V^{(r)}_{\mathcal{R}_{j+1}}$, and thus $v(R) < V^{(r)}_{\mathcal{R}_{j+1}}$. ◀

▶ **Fact 9.** *Let $\ell, r > 0$. If $V^{(\ell)}_{\mathcal{L}_j} + V^{(r)}_{\mathcal{R}_{j+1}} \le V$ for some $j \in \{0, \dots, n\}$, then $\ell \cdot r \le A_V$.*

Proof. Let L and R be the ℓ-th and r-th entry in $\mathcal{L}_j$ and $\mathcal{R}_{j+1}$, respectively. Note that $v(L \cup R) \le V$ implies $\operatorname{rank}_v(L \cup R) \le A_V$ by definition of A_V. Moreover, $\operatorname{rank}_v(L) \ge \ell$ and $\operatorname{rank}_v(R) \ge r$ (the equalities may be sharp due to draws). Now, Fact 7 yields the claimed bound. ◀

Note that $\mathcal{L}_j$ can be obtained by interleaving $|C_j|$ copies of $\mathcal{L}_{j-1}$, where each copy corresponds to extending the choices from $\mathcal{L}_{j-1}$ with a different item. If we were to construct $\mathcal{L}_j$ having access to the whole $\mathcal{L}_{j-1}$, we could proceed as follows. For each $c \in C_j$, we maintain an *iterator* on $\mathcal{L}_{j-1}$ pointing to the first element S on $\mathcal{L}_{j-1}$ for which $S \cup \{c\}$ has not yet been added to $\mathcal{L}_j$. The associated *value* is $v(S \cup \{c\})$. All iterators initially point at the first element of $\mathcal{L}_{j-1}$. Then the next element to append to $\mathcal{L}_j$ is always $S \cup \{c\}$ corresponding to the iterator with minimum value. Having processed this partial choice, we advance the pointer (or remove it, once it has already scanned the whole $\mathcal{L}_{j-1}$). This process can be implemented using a binary heap H_j as a priority queue, so that initialization requires $\mathcal{O}(|C_j|)$ time and outputting a single element takes $\mathcal{O}(\log |C_j|)$ time.

For $r \geq 0$, let $\mathcal{L}_j^{(r)}$ be the prefix of $\mathcal{L}_j$ of length $\min(r, |\mathcal{L}_j|)$ and $\mathcal{R}_j^{(r)}$ be the prefix of $\mathcal{R}_j$ of length $\min(r, |\mathcal{R}_j|)$. A technical transformation of the procedure stated above leads to an online algorithm that constructs the prefixes $\mathcal{L}_j^{(r)}$ and $\mathcal{R}_j^{(r)}$ (details will be provided in the full version). Along with each reported partial choice S, the algorithm also computes $w(S)$.

▶ **Lemma 10.** *After $\mathcal{O}(N)$-time initialization, one can construct $\mathcal{L}_1^{(i)}, \ldots, \mathcal{L}_n^{(i)}$ online for $i = 0, 1, \ldots,$ spending $\mathcal{O}(n \log \lambda)$ time per each step. Symmetrically, one can construct $\mathcal{R}_1^{(i)}, \ldots, \mathcal{R}_n^{(i)}$ in the same time complexity.*

Also the following reduction can be obtained (details are left for the full version).

▶ **Lemma 11.** *Given an instance I of the* Multichoice Knapsack *problem, one can compute in $\mathcal{O}(N + \lambda \log A)$ time an equivalent instance I' with $A'_V \leq A_V$, $A'_W \leq A_W$, $\lambda' \leq \lambda$, and $n' = \mathcal{O}(\frac{\log A}{\log \lambda})$.*

Note that we may always assume that $\lambda \leq a$. Indeed, if we order the items $c \in C_i$ according to $v(c)$, then only the first A_V of them might belong to a choice S with $v(S) \leq V$.

▶ **Theorem 12.** Multichoice Knapsack *can be solved in $\mathcal{O}(N + \sqrt{a\lambda} \log A)$ time.*

Proof. Below, we give an algorithm working in $\mathcal{O}(N + \sqrt{A_V \lambda} \log A)$ time. The final solution runs it in parallel on the original instance and on the instance with v and V swapped with w and W, waiting until at least one of them terminates.

We increment an integer r starting from 1, maintaining $\ell = \lceil \frac{r}{\lambda} \rceil$ and the lists $\mathcal{L}_j^{(\ell)}$ and $\mathcal{R}_{j+1}^{(r)}$ for $0 \leq j \leq n$, as long as $V_{\mathcal{L}_j}^{(\ell)} + V_{\mathcal{R}_{j+1}}^{(r)} \leq V$ for some j (or until all the lists have been completely generated). By Fact 9, we stop at $r = \mathcal{O}(\sqrt{A_V \lambda})$. Lemma 11 lets us assume that $n = \mathcal{O}(\frac{\log A}{\log \lambda})$, so the running time of this phase is $\mathcal{O}(N + \sqrt{A_V \lambda} \log A)$ due to Lemma 10. The preprocessing time of Lemma 11 is dominated by this complexity.

Due to Lemma 8, every feasible solution S admits a decomposition $S = L \cup \{c\} \cup R$ with $L \in \mathcal{L}_{j-1}^{(\ell)}$, $c \in C_j$, and $R \in \mathcal{R}_{j+1}^{(r)}$ for some index j. We consider all possibilities for j. For each of them we will reduce searching for S to an instance of the Multichoice Knapsack problem with $N' = \mathcal{O}(\sqrt{A_V \lambda})$ and $n' = 2$. By Lemma 6, these instances can be solved in $\mathcal{O}(n\sqrt{A_V \lambda}) = \mathcal{O}(\sqrt{A_V \lambda} \frac{\log A}{\log \lambda})$ time in total.

The items of the j-th instance are going to belong to classes $\mathcal{L}_{j-1}^{(\ell)} \odot C_j$ and $\mathcal{R}_{j+1}^{(r)}$, where $\mathcal{L}_{j-1}^{(\ell)} \odot C_j = \{L \cup \{c\} : L \in \mathcal{L}_{j-1}^{(\ell)}, c \in C_j\}$. The set $\mathcal{L}_{j-1}^{(\ell)} \odot C_j$ can be constructed by merging $|C_j| \leq \lambda$ sorted lists, each of size $\ell = \mathcal{O}(\sqrt{A_V/\lambda})$, i.e., in $\mathcal{O}(\sqrt{A_V \lambda} \log \lambda)$ time. Summing up over all indices j, this gives $\mathcal{O}(\sqrt{A_V \lambda} \log \lambda \frac{\log A}{\log \lambda}) = \mathcal{O}(\sqrt{A_V \lambda} \log A)$ time.

Clearly, each feasible solution of the constructed instances represents a feasible solution of the initial instance, and by Lemma 8, every feasible solution of the initial instance has its counterpart in one of the constructed instances. ◀

5 Weighted Consensus and General Weighted Pattern Matching

The Weighted Consensus problem is formally defined as follows.

Weighted Consensus Problem
Input: Two weighted sequences X and Y of length n with at most λ letters at each position and R in total, and a threshold probability $\frac{1}{z}$.
Output: A string S such that $S \approx_{\frac{1}{z}} X$ and $S \approx_{\frac{1}{z}} Y$ or NONE if no such string exists.

If two weighted sequences satisfy the consensus, we write $X \approx_{\frac{1}{z}} Y$ and say that X *matches* Y *with probability at least* $\frac{1}{z}$. With this definition of a match, we extend the notion of an occurrence and the notation $Occ_{\frac{1}{z}}(P,T)$ to arbitrary weighted sequences.

General Weighted Pattern Matching (GWPM) Problem
Input: Two weighted sequences P and T of length m and n, respectively, with at most λ letters at each position and R in total, and a threshold probability $\frac{1}{z}$.
Output: The set $Occ_{\frac{1}{z}}(P,T)$.

In the case of the GWPM problem, it is more useful to provide an *oracle* that finds witness strings that correspond to the respective occurrences of the pattern. Such an oracle, given $i \in Occ_{\frac{1}{z}}(P,T)$, computes a string that matches both P and $T[i..i+m-1]$.

Our algorithms rely on the following simple observation, originally due to Amir et al. [1].

▶ **Fact 13** ([1]). *A weighted sequence has at most z different matching strings.*

The Weighted Consensus problem is actually a special case of Multichoice Knapsack. Namely, given an instance of the former, we can create an instance of the latter with n classes C_i, each containing an item $c_{i,s}$ for every letter s which has non-zero probability at position i in both X and Y. We set $v(c_{i,s}) = -\log \pi_i^{(X)}(s)$ and $w(c_{i,s}) = -\log \pi_i^{(Y)}(s)$ for this item, whereas the thresholds are $V = W = \log z$. It is easy to see that this reduction indeed yields an equivalent instance and that it can be implemented in linear time. By Fact 13, we have $A \le z$ for this instance, so Theorem 12 yields the following result:

▶ **Corollary 14.** Weighted Consensus *problem can be solved in $\mathcal{O}(R + \sqrt{z\lambda}\log z)$ time.*

The GWPM problem can be clearly reduced to $n+m-1$ instances of Weighted Consensus. This leads to a naïve $\mathcal{O}(nR + n\sqrt{z\lambda}\log z)$-time algorithm. Below, we remove the first term in this complexity. Our solution applies the approach used in Section 3 for Weighted Pattern Matching and uses an observation analogous to Observation 4.

▶ **Observation 15.** *If X and Y are weighted sequences that match with threshold $\frac{1}{z}$, then $d_H(\mathcal{H}(X), \mathcal{H}(Y)) \le 2\lfloor \log z \rfloor$. Moreover there exists a consensus string S such that $S[i] = \mathcal{H}(X)[i] = \mathcal{H}(Y)[i]$ unless $\mathcal{H}(X)[i] \ne \mathcal{H}(Y)[i]$.*

Our algorithm starts by computing $P' = \mathcal{H}(P)$ and $T' = \mathcal{H}(T)$ and the data structure for *lcp*-queries in $P'T'$. We try to match P with every factor $T[p..p+m-1]$ of the text. Following Observation 15, we check if $d_H(T'[p..p+m-1], P') \le 2\lfloor \log z \rfloor$. If not, then we know that no match is possible. Otherwise, let D be the set of positions of mismatches between $T'[p..p+m-1]$ and P'. Assume that we store $\alpha = \prod_{j=1}^{m} \pi_{p+j-1}^{(T)}(T'[p+j-1])$ and $\beta = \prod_{j=1}^{m} \pi_j^{(P)}(P'[j])$. Then, in $\mathcal{O}(|D|)$ time we can compute $\alpha' = \prod_{j \notin D} \pi_{p+j-1}^{(T)}(T'[p+j-1])$ and $\beta' = \prod_{j \notin D} \pi_j^{(P)}(P'[j])$. Now, we only need to check what happens at the positions in D.

If $D = \emptyset$, then it suffices to check if $\alpha \geq \frac{1}{z}$ and $\beta \geq \frac{1}{z}$. Otherwise, we construct two weighted sequences X and Y by selecting only the positions from D in $T[p..p+m-1]$ and in P. We multiply the probabilities of all letters at the first position in X by α' and in Y by β'. It is clear that $X \approx_{\frac{1}{z}} Y$ if and only if $T[p..p+m-1] \approx_{\frac{1}{z}} P$.

Thus, we have reduced the GWPM problem to at most $n-m+1$ instances of the Weighted Consensus problem for strings of length $\mathcal{O}(\log z)$. By Corollary 14, solving each instance takes $\mathcal{O}(\lambda \log z + \sqrt{z\lambda}\log z) = \mathcal{O}(\sqrt{z\lambda}\log z)$ time. Our reduction requires $\mathcal{O}(R \log^2 \log \lambda)$ time to preprocess the input (as in Theorem 5), but this is dominated by the $\mathcal{O}(n\sqrt{z\lambda}\log z)$ total time of solving the Weighted Consensus instances. If we memorize the solutions to all those instances together with the underlying sets of mismatches D, we can also implement the oracle for the GWPM problem with $\mathcal{O}(m)$-time queries.

A tailor-made solution can be designed (details will be provided in the full version) to replace the generic algorithm for the Multichoice Knapsack problem, which lets us improve the $\log z$ factor to $\log\log z + \log \lambda$.

▶ **Theorem 16.** *The* GWPM *problem can be solved in* $\mathcal{O}(n\sqrt{z\lambda}(\log\log z + \log\lambda))$ *time. An oracle for the* GWPM *problem using* $\mathcal{O}(n \log z)$ *space and supporting queries in* $\mathcal{O}(m)$ *time can be computed within the same time complexity.*

A reduction from Multichoice Knapsack to Weighted Consensus (proofs will be provided in the full version) immediately yields that any significant improvement in the dependence on z and λ in the running time of our algorithm would lead to breaking long-standing barriers for special cases of Multichoice Knapsack.

▶ **Theorem 17.** Weighted Consensus *problem is NP-hard and cannot be solved in:*

1. $\mathcal{O}^*(z^{\varepsilon})$ *time for every* $\varepsilon > 0$, *unless the Exponential Time Hypothesis (ETH) fails;*
2. $\mathcal{O}^*(z^{0.5-\varepsilon})$ *time for some* $\varepsilon > 0$, *unless there is an* $\mathcal{O}^*(2^{(0.5-\varepsilon)n})$*-time algorithm for the* Subset Sum *problem;*
3. $\tilde{\mathcal{O}}(R + z^{0.5}\lambda^{0.5-\varepsilon})$ *time for some* $\varepsilon > 0$ *and for* $n = \mathcal{O}(1)$, *unless there is an* $\mathcal{O}(\lambda^{2(1-\varepsilon)})$*-time algorithm for 3-*Sum.

Nevertheless, it might still be possible to improve the dependence on n in the GWPM problem. For example, one may hope to achieve $\tilde{\mathcal{O}}(nz^{0.5-\varepsilon} + z^{0.5})$ time for $\lambda = \mathcal{O}(1)$.

6 Final Remarks

In Section 4, we gave an $\mathcal{O}(N + a^{0.5}\lambda^{0.5}\log A)$-time algorithm for the Multichoice Knapsack problem. Improvement of either exponent to $0.5-\varepsilon$ would result in a breakthrough for the Subset Sum and 3-Sum problems, respectively. Nevertheless, this does not refute the existence of faster algorithms for some particular values (a, λ) other than those emerging from instances of Subset Sum or 3-Sum. Indeed, we can show an algorithm that is superior if $\frac{\log a}{\log \lambda}$ is a constant other than an odd integer. We can also prove it to be optimal (up to lower order terms) for every constant $\frac{\log a}{\log \lambda}$ unless the k-Sum conjecture fails. The details will be provided in the full version.

References

1 Amihood Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theor. Comput. Sci.*, 395(2-3):298–310, April 2008. doi:10.1016/j.tcs.2008.01.006.

2 Carl Barton, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Efficient index for weighted sequences. In Roberto Grossi and Moshe Lewenstein, editors, *Combinatorial Pattern Matching, CPM 2016*, volume 54 of *LIPIcs*, pages 4:1–4:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.4.

3 Carl Barton and Solon P. Pissis. Linear-time computation of prefix table for weighted strings. In Florin Manea and Dirk Nowotka, editors, *Combinatorics on Words, WORDS 2015*, volume 9304 of *LNCS*, pages 73–84. Springer, 2015. doi:10.1007/978-3-319-23660-5_7.

4 Sudip Biswas, Manish Patil, Sharma V. Thankachan, and Rahul Shah. Probabilistic threshold indexing for uncertain strings. In Evaggelia Pitoura, Sofian Maabout, Georgia Koutrika, Amélie Marian, Letizia Tanca, Ioana Manolescu, and Kostas Stefanidis, editors, *19th International Conference on Extending Database Technology, EDBT 2016*, pages 401–412. OpenProceedings.org, 2016. doi:10.5441/002/edbt.2016.37.

5 Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, and Kostas Tsichlas. Pattern matching on weighted sequences. In *Algorithms and Computational Methods for Biochemical and Evolutionary Networks, CompBioNets 2004*, KCL publications, 2004.

6 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings.* Cambridge University Press, New York, NY, USA, 2007.

7 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science, MFCS 2015, Part II*, volume 9235 of *LNCS*, pages 287–298. Springer, 2015. doi:10.1007/978-3-662-48054-0_24.

8 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.

9 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

10 Eitan M. Gurari. *Introduction to the theory of computation.* Computer Science Press, 1989.

11 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

12 Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundam. Inform.*, 71(2-3):259–277, 2006. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-07.

13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

14 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems.* Springer, 2004.

15 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/92.

16 Cinzia Pizzi and Esko Ukkonen. Fast profile matching algorithms – A survey. *Theor. Comput. Sci.*, 395(2-3):137–157, 2008. doi:10.1016/j.tcs.2008.01.015.

17 Sanguthevar Rajasekaran, X. Jin, and John L. Spouge. The efficient computation of position-specific match scores with the fast Fourier transform. *J. Comp. Biol.*, 9(1):23–33, 2002. doi:10.1089/10665270252833172.

18 Milan Ružić. Constructing efficient dictionaries in close to sorting time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, ICALP 2008, Part I*, volume 5125 of *LNCS*, pages 84–95. Springer, 2008. doi:10.1007/978-3-540-70575-8_8.

Hierarchical Time-Dependent Oracles*

Spyros Kontogiannis[1], Dorothea Wagner[2], and Christos Zaroliagis[3]

1 Comp. Sci. & Eng. Dept., University of Ioannina, Greece; and
Computer Technology Institute and Press "Diophantus", Greece
kontog@cse.uoi.gr

2 Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

3 Comp. Eng. & Inf. Dept., University of Patras, Greece; and
Computer Technology Institute and Press "Diophantus", Greece
zaro@ceid.upatras.gr

Abstract

We study networks obeying *time-dependent* min-cost path metrics, and present novel oracles for them which *provably* achieve two unique features: (i) *subquadratic* preprocessing time and space, *independent* of the metric's amount of disconcavity; (ii) *sublinear* query time, in either the network size or the actual Dijkstra-Rank of the query at hand.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Time-dependent shortest paths, FIFO property, Distance oracles

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.47

1 Introduction

Concurrent technological infrastructures (e.g., road networks, social networks, e-commerce platforms, energy-management systems) are typically of very large scale and impose as a routine task the computation of min-cost paths in real-time, while their characteristics usually evolve with time. The large-scale and real-time response challenges have been addressed in the last 15 years by means of a new algorithmic trend: the provision of *oracles*. That is, data structures created by appropriately selecting precomputed information (summaries) and which subsequently support query algorithms with real-time responses. The quality of an oracle is assessed by its preprocessing space and time requirements, the time-complexity of the query algorithm and the approximation guarantee (stretch). Numerous oracles have been proposed and analyzed (see e.g., [1, 21, 22, 24, 25, 27, 28, 29] and references therein) for large-scale, mostly undirected networks, accompanied by a *static* arc-cost metric. In tandem with oracles, an equally important effort (with similar characteristics) has also emerged in the last 15 years under the tag of *speedup techniques*, for approaches tailored to work extremely well in real-life instances (see e.g., [3] and references therein).

The temporality of the network characteristics is often depicted by some kind of predetermined dependence of the metric on the actual time that each resource is used (e.g., traversal speed in road networks, packet-loss rate in IT networks, arc availability in social networks, etc). Perhaps the most typical application scenario, motivating also our work, is

* This work is partially supported by the EU FP7/2007-2013, under grant agreements no. 609026 (project MOVESMART) and no. 621133 (project HoPE), and by DFG under grant agreement no. FOR-2083.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 47; pp. 47:1–47:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

route planning in road networks where the travel-time for traversing an arc $a = uv$ (modeling a road segment) depends on the temporal traffic conditions while traversing uv, and thus on the departure-time from its tail u. This gives rise to *time-varying* network models and to computing min-cost (a.k.a. shortest) paths in such networks. Several variants of this problem try to model time-variation of the underlying graph and/or the arc-cost metric (e.g., dynamic shortest paths, parametric shortest paths, stochastic shortest paths, temporal networks, etc). In this work we assume that the cost variation of each arc a is determined by a *continuous, piecewise linear (pwl) and periodic function* $D[a]$ of the time at which a is actually being traversed[1], as in [7, 8, 12, 20]. When providing route plans in time-dependent road networks, arc-costs are considered as *arc-travel-times*, and time-dependent shortest paths as *minimum-travel-time* paths. The goal is then to determine the cost (*minimum-travel-time*) of a shortest path from an origin o to a destination d, as a function of the *departure-time* t_o from o. Due to the time-dependence of the arc-cost metric, the actual arc-cost value of an arc $a = uv$ is unknown until the exact time $t_u \geq t_o$ at which uv starts being traversed.

Problem setting and related work. Two variants of the *time-dependent shortest path* problem have been considered in the literature: $TDSP(o, d, t_o)$ (resp. $TDSP(o, \star, t_o)$) focuses on the one-to-one (resp. one-to-all) determination of the *scalar cost* of a minimum-travel-time path to d (resp. for all d), when departing from the origin o at time t_o. $TDSP(o, d)$ (resp. $TDSP(o, \star)$) focuses on the one-to-one (resp., one-to-all) succinct representation of the time-dependent minimum-travel-time path *function*(s) $D[o, d]$ from o to d (resp. towards all reachable d), and all departure-times from o. $TDSP(o, d, t_o)$ has been studied as early as [5]. The first work on $TDSP(o, d, t_o)$ for continuous time-axis was [11] where it was proved that, if *waiting-at-nodes* is allowed unconditionally, then $TDSP(o, d, t_o)$ is solvable in quasilinear time via a time-dependent variant of Dijkstra's algorithm (we call it `TDD`), which relaxes arcs by computing the arc costs "on the fly", upon settling their tails. A more complete treatment of the continuous-time case, considering various limitations in the waiting-times at nodes of the network, was provided in [13]; an algorithm was also given for $TDSP(o, d, t_o)$, whose complexity cannot be bounded by a function of the network topology. An excellent overview of the problem is provided in [20]. Among other results, it was proved that for *affine arc-cost* functions possessing the FIFO property (according to which all the arc-cost functions have slopes at least -1), in addition to `TDD`, a time-dependent variant of the label-correcting Bellman-Ford algorithm also works. Moreover, if waiting-at-nodes is *forbidden* and the arc-costs do not preserve the FIFO property, then *subpath-optimality* of shortest paths is not necessarily preserved. In that case, many variants of the problem become NP-hard [23]. Additionally, when shortest path costs are well defined and optimal waiting-times at nodes always exist, a non-FIFO arc with *unrestricted-waiting-at-tail* policy is equivalent to a FIFO arc in which waiting at the tail is not beneficial [20]. For these reasons, we focus here on instances for which the FIFO property holds, as indeed is the case with most of past and recent works on $TDSP(o, d, t_o)$. The complexity of $TDSP(o, d)$ was first questioned in [6, 7] and remained open until recently, when it was proved in [12] that, even for FIFO-abiding pwl arc-cost functions and a *single* origin-destination pair (o, d), the number of breakpoints for succinctly representing $D[o, d]$ is $(1 + K) \cdot n^{\Theta(\log n)}$, where n is

[1] Major car navigator vendors provide real-time estimations of travel-time values by periodically sampling the average speed of road segments, using the cars connected to the service as sampling devices. The most customary way to represent this historic traffic data, is to consider the continuous pwl interpolants of the sample points as arc-travel-time functions of the corresponding instance.

the number of vertices and K is the number of breakpoints in all the arc-cost functions. Note that K can be substituted by the number K^* of *concavity-spoiling* breakpoints (at which the slopes increase) of the arc-cost functions. Several *output-sensitive* algorithms for the construction of $D[o,d]$ have appeared [7, 8, 12, 20], the most efficient being the ones in [8, 12]. Due to the hardness of $TDSP(o,d)$, and also since the arc-cost functions are typically only (e.g., pwl) approximations of the actual costs, it is quite natural to seek for succinct representations of approximations to $D[o,d]$, which aim at trading-off accuracy for computational effort. Several *one-to-one* $(1+\varepsilon)$-approximation algorithms for $TDSP(o,d)$ have appeared in the literature [8, 12, 19], the most successful being those provided in [19]. The first *one-to-all* $(1+\varepsilon)$-approximation algorithm for $TDSP(o,\star)$, called *bisection* (BIS), was given in [17]. It is based on bisecting the (common to all functions) axis of departure-times from o and considers slightly stricter assumptions than just the FIFO property for the arc-cost metric. BIS requires $\mathcal{O}\left(\frac{K^*}{\varepsilon}\cdot\log^2\left(\frac{n}{\varepsilon}\right)\right)$ calls to $TDSP(o,\star,t_o)$, assuming that the travel-time diameter is upper-bounded by the period $T=n^\alpha$, for some tuning parameter $\alpha\in(0,1)$. Note that all one-to-one approximation algorithms for $TDSP(o,d)$ [8, 12, 19] demand, in worst-case, a comparable amount of calls to $TDSP(o,\star,t_o)$, just for one *od*-pair.

Minimum-travel-time oracles for time-dependent networks (TD-oracles henceforth) had received no attention until recently [17]. A TD-oracle is an offline-precomputed data structure that allows the *efficient evaluation* of an upper-approximation $\overline{\Delta}[o,d](t_o)$ of $D[o,d](t_o)$, for *any* possible query $(o,d,t_o)\in V\times V\times\mathbb{R}_{\geq 0}$ that may appear in an online fashion. One trivial solution would be to provide a succinct representation of $\overline{\Delta}[o,d]$ for all $(o,d)\in V\times V$, for the sake of rapid evaluations in the future but at the expense of superquadratic space. Another trivial solution would be to execute TDD "on-the-fly" per query (o,d,t_o), at the expense of superlinear query-time. A non-trivial TD-oracle should aim to trade-off preprocessing requirements with query-times and approximation guarantees. In particular, it should precompute a data structure in *subquadratic* time and space, and also provide a query algorithm which evaluates *efficiently* (i.e., faster than TDD) $\overline{\Delta}[o,d](t_o)$, where $\overline{\Delta}[o,d]$ must be a *provably* good approximation of $D[o,d]$. Note that there exists important applied work (*speedup heuristics*) for computing time-dependent shortest paths (e.g., [4, 9, 10, 18]), which however provide mainly empirical evidence on the success of the adopted approaches.

The TD-oracles in [17] require $\mathcal{O}\big(n^{2-\beta}(K^*+1)\big)$ preprocessing space and time, for constant $\beta\in(0,1)$, and can answer queries (under certain conditions) in time $\mathcal{O}\big(n^\delta\big)$, for constant $\delta\in(0,1)$. When $K^*\in\mathrm{o}(n)$, the oracles can be fine-tuned to assure query-time $\mathrm{o}(n)$ and preprocessing requirements $\mathrm{o}\big(n^2\big)$. An extensive experimental evaluation of those oracles on a real-world road network is provided in [14], demonstrating their practicality, at the expense, however, of large memory consumption due to the linear dependence of the preprocessing space on K^* which can be $\Omega(n)$. The main challenge addressed here is to provide TD-oracles that achieve: (i) subquadratic preprocessing requirements, *independently* of K^*; and (ii) query-times sublinear, not just in the network size n, but in the number $\Gamma[o,d](t_o)$ (a.k.a. *Dijkstra-Rank*) of settled vertices when executing $\mathtt{TDD}(o,\star,t_o)$ until d is settled.

Our contributions and roadmap. We address positively the aforementioned challenge by providing: (i) A novel and remarkably simple algorithm (TRAP) (cf. Section 3) for constructing one-to-many $(1+\varepsilon)$-upper-approximations $\overline{\Delta}[o,d]$ (*summaries*) of minimum-travel-time functions $D[o,d]$, for all "sufficiently distant" destinations d from the origin o. TRAP requires $\mathrm{o}(n)$ calls to $TDSP(o,\star,t_o)$, which is *independent* of the degree of concavity K^*. Its novelty is that it does not demand the concavity of the unknown function to approximate. (ii) The

Table 1 Achievements of oracles for TD-instances with period $T = n^{\alpha}$, for $\alpha \in (0,1)$. The stretch of all query algorithms is $1 + \sigma(r) = 1 + \varepsilon \cdot \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$. For all but the first oracle, we assume that $\beta \downarrow 0$.

	preprocessing space/time	query time	recursion budget (depth) r
[17]	$K^* \cdot n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \in \mathcal{O}(1)$
TRAPONLY	$n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \approx \frac{\delta}{\alpha} - 1$
FLAT	$n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \approx \frac{2\delta}{\alpha} - 1$
HORN	$n^{2-\beta+o(1)}$	$\approx \Gamma[o,d](t_o)^{\delta+o(1)}$	$r \approx \frac{2\delta}{\alpha} - 1$

TRAPONLY and FLAT oracles (cf. Section 4) which exploit TRAP and BIS to construct minimum-travel-time summaries from randomly selected landmarks to *all* reachable destinations. The preprocessed data structures require subquadratic space and time, independently of K^*. FLAT uses the query algorithms of [17]. TRAPONLY needs to extend them in order to recover missing summaries for local neighborhoods around each landmark. In both cases sublinear query-times are achieved. (iii) The HORN oracle (cf. Section 5) which organizes a hierarchy of landmarks, from many local landmarks possessing summaries only for small neighborhoods of destinations around them, up to a few global landmarks possessing summaries for all reachable destinations. HORN's preprocessing requirements are again subquadratic. We then devise and analyze a novel query algorithm (HQA) to exploit this hierarchy, with query-time *sublinear* in the Dijkstra-Rank of the query at hand. Except for the choice of landmarks, our algorithms are deterministic. An experimental study [15] demonstrates the excellent performance of our oracles in practice, achieving considerable memory savings and query-times about three orders of magnitude faster than TDD, and more than 70% faster than those in [14]. Table 1 summarizes the achievements of the TD-oracles presented here and their comparison with the oracles in [17]. Due to lack of space, all missing proofs are provided in the full version of the paper [16].

2 Preliminaries

Notation and terminology. For any integer $k \geq 1$, let $[k] = \{1, 2, \ldots, k\}$. A *time-dependent network instance* (TD-instance henceforth) consists of a directed graph $G = (V, A)$ with $|V| = n$ vertices and $|A| = m \in \mathcal{O}(n)$ arcs, where each arc $a \in A$ is accompanied with a continuous, pwl *arc-cost* function $D[a] : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{>0}$. We assume that all these functions are periodic with period $T > 0$ and are defined as follows: $\forall k \in \mathbb{N}, \forall t \in [0,T),\ D[a](kT + t) = d[a](t)$, where $d[a] : [0,T) \to (0, M_a]$ is such that $\lim_{t \uparrow T} d[a](t) = d[a](0)$, for some fixed integer M_a denoting the maximum possible cost ever seen for arc a. Let also $M = \max_{a \in A} M_a$ denote the maximum arc-cost ever seen in the entire network. Since $D[a]$ is periodic, continuous and pwl, it can be represented succinctly by a sequence of K_a breakpoints (i.e., pairs of departure-times and arc-cost values) defining $d[a]$. $K = \sum_{a \in A} K_a$ is the number of breakpoints representing all arc-cost functions, $K_{\max} = \max_{a \in A} K_a$, and K^* is the number of *concavity-spoiling* breakpoints (the ones at which the arc-cost slopes increase). Clearly, $K^* \leq K$, and $K^* = 0$ for *concave* arc-cost functions. To ease the exposition and also for the sake of compliance with terminology in previous works (inspired by the primary application scenario of route planning in time-dependent road networks), we consider arc-costs as *arc-travel-times* and time-dependent shortest paths as *minimum-travel-time* paths. This terminology facilitates the following definitions. The *arc-arrival-time* function of $a \in A$ is $Arr[a](t) = t + D[a](t)$, $\forall t \in [0, \infty)$. The *path-arrival-time* function of a path $p = \langle a_1, \ldots, a_k \rangle$ in G (represented as a sequence of arcs) is the composition $Arr[p](t) = Arr[a_k](Arr[a_{k-1}](\cdots(Arr[a_1](t))\cdots))$ of the arc-arrival-time functions for the constituent arcs. The *path-travel-time* function is then $D[p](t) = Arr[p](t) - t$.

For any $(o,d) \in V \times V$, $\mathcal{P}_{o,d}$ denotes the set of *od*-paths. For any $p \in \mathcal{P}_{o,x}$ and $q \in \mathcal{P}_{x,d}$, $s = p \bullet q \in \mathcal{P}_{o,d}$ is the concatenation of p and q at x. The *earliest-arrival-time* function is $Arr[o,d](t_o) = \min_{p \in \mathcal{P}_{o,d}} \{Arr[p](t_o)\}$, $\forall t_o \geq 0$, while the *minimum-travel-time* function is defined as $D[o,d](t_o) = \min_{p \in \mathcal{P}_{o,d}} \{D[p](t_o)\} = Arr[o,d](t_o) - t_o$. For a given query (o,d,t_o), $SP[o,d](t_o) = \{p \in P_{o,d} : Arr[p](t_o) = Arr[o,d](t_o)\}$ is the set of earliest-arrival-time (equivalently, minimum-travel-time) paths. $ASP[o,d](t_o)$ is the set of *od*-paths whose travel-time values are $(1+\varepsilon)$-approximations of the minimum-travel-time among all *od*-paths.

When we say that we *"grow a* `TDD` *ball from* (o,t_o)*"*, we refer to the execution of `TDD` from $o \in V$ at departure-time $t_o \in [0,T)$ for solving $TDSP(o,\star,t_o)$ (resp. $TDSP(o,d,t_o)$, for a specific destination d). Such a call, denoted as $\texttt{TDD}(o,\star,t_o)$ (resp. $\texttt{TDD}(o,d,t_o)$), takes time $\mathcal{O}(m + n\log(n)[1 + \log\log(1 + K_{\max})]) = \mathcal{O}(n\log(n)\log\log(K_{\max})])$, using predecessor search for evaluating continuous pwl functions [17]. The *Dijkstra-Rank* $\Gamma[o,d](t_o)$ of (o,d,t_o) is the number of settled vertices up to d, when executing $\texttt{TDD}(o,d,t_o)$.

$\forall a = uv \in A$ and $[t_s,t_f) \subseteq [0,T)$, we define upper- and lower-bounding travel-time metrics: the *minimally-congested* travel-time $\underline{D}[uv](t_s,t_f) := \min_{t_u \in [t_s,t_f)}\{D[uv](t_u)\}$ and the *maximally-congested* travel-time $\overline{D}[uv](t_s,t_f) := \max_{t_u \in [t_s,t_f)}\{D[uv](t_u)\}$. If $[t_s,t_f) = [0,T)$, we refer to the static *free-flow* and *full-congestion* metrics $\underline{D},\overline{D} : A \to [1,M]$, respectively. Each arc $a \in A$ is also equipped with scalars $\underline{D}[a]$ and $\overline{D}[a]$ in these static metrics. For any arc-cost metric D, $diam(G,D)$ is the diameter (largest possible vertex-to-vertex distance) of the graph. For example, $diam(G,\underline{D})$ is the free-flow diameter of G.

In our TD-instance, we assume that $T \geq diam(G,\underline{D})$. If not, we take the minimum number c of copies of each $d[a]$ as a single arc-travel-time function $d'[a] : [0,cT) \mapsto \mathbb{R}_{>0}$ and $D'[a](t + kT') = d'[a](t)$, $\forall t \in [0,T')$ such that $T' = cT \geq diam(G,\underline{D'})$. In addition, we can guarantee that $T = n^\alpha$ for a *constant* $\alpha \in (0,1)$ of our control. If $T \neq n^\alpha$, we scale the travel-time metric by setting $D'' = \frac{n^\alpha}{T} \cdot D$ (e.g., we change the unit by which we measure time from milliseconds to seconds) and use the period $T'' = n^\alpha$, without affecting the structure of the instance at all. From now on we assume w.l.o.g. that $T = n^\alpha \geq diam(G,\underline{D})$.

For any $v \in V$, departure-time $t_v \in \mathbb{R}_{\geq 0}$, integer $F \in [n]$ and $R > 0$, $B[v;F](t_v)$ ($B[v;R](t_v)$) is a ball of size F (of radius R) grown by `TDD` from (v,t_v), in the time-dependent metric. Analogously, $\underline{B}[v;F]$ ($\underline{B}[v;R]$) and $\overline{B}[v;F]$ ($\overline{B}[v;R]$) are, respectively, the size-F (radius-R) balls from v in the free-flow and fully-congested travel-time metrics.

A pair of continuous, pwl, periodic functions $\overline{\Delta}[o,d]$ and $\underline{\Delta}[o,d]$), with a (hopefully) small number of breakpoints, are $(1+\varepsilon)$*-upper-approximation* and $(1+\varepsilon)$*-lower-approximation* of $D[o,d]$, if $\forall t_o \geq 0$, $\frac{D[o,d](t_o)}{1+\varepsilon} \leq \underline{\Delta}[o,d](t_o) \leq D[o,d](t_o) \leq \overline{\Delta}[o,d](t_o) \leq (1+\varepsilon) \cdot D[o,d](t_o)$.

Assumptions on the arc-cost metric. The directedness and time-dependence of the TD-instance imply an asymmetric arc-cost metric, which also evolves with time. To achieve a smooth transition from static and undirected graphs towards time-dependent and directed graphs, we need a quantification of the degrees of asymmetry and evolution of our metric over time. These are captured via a set of parameters depicting the steepness of the minimum-travel-time functions, the ratio of minimum-travel-times in opposite directions, and the relation between graph expansion and travel-times. We make some assumptions on the values of these parameters, which seem quite natural for our main application scenario (route planning in road networks). We only present a qualitative interpretation of them. Their exact statements, along with their validation on real-world road networks, are presented in [16]. It is noted that Assumptions 1 and 2 were exploited also in the analyses in [17].

▶ **Assumption 1** (Bounded Travel-Time Slopes). *All the minimum-travel-time slopes are bounded in a given interval* $[-\Lambda_{\min}, \Lambda_{\max}]$*, for given constants* $\Lambda_{\min} \in [0, 1)$ *and* $\Lambda_{\max} \geq 0$*.*

▶ **Assumption 2** (Bounded Opposite Trips). *The ratio of minimum-travel-times in opposite directions between two vertices, for any specific departure-time but not necessarily via the same path, is upper bounded by a given constant* $\zeta \geq 1$*.*

▶ **Assumption 3** (Growth of Free-Flow Dijkstra Balls). $\forall F \in [n]$*, the free-flow ball* $\underline{B}[v; F]$ *blows-up by at most a polylogarithmic factor, when expanding its (free-flow) radius up to the value of the full-congestion radius within* $\underline{B}[v; F]$*.*

Finally, we need to quantify the correlation between the arc-cost metric and the Dijkstra-Rank metric induced by it. For this reason, inspired by the notion of the doubling dimension (e.g., [2] and references therein), we consider some *scalar* $\lambda \geq 1$ and functions $f, g : \mathbb{N} \mapsto [1, \infty)$, such that the following hold: $\forall (o, d, t_o) \in V \times V \times [0, T)$, (i) $\Gamma[o, d](t_o) \leq f(n) \cdot (D[o, d](t_o))^{\lambda}$, and (ii) $D[o, d](t_o) \leq g(n) \cdot (\Gamma[o, d](t_o))^{1/\lambda}$. This property trivially holds, e.g., for $\lambda = 1$, $f(n) = n$, and $g(n) = \max_{a \in A} \left\{ \overline{D}[a] \right\}$. Of course, our interest is for the smallest possible values of λ and at the same time the slowest-growing functions $f(n), g(n)$. Our last assumption quantifies the boundedness of this correlation by restricting λ, $f(n)$ and $g(n)$.

▶ **Assumption 4.** *There exist* $\lambda \in o\left(\frac{\log(n)}{\log\log(n)}\right)$ *and* $f(n), g(n) \in \mathrm{polylog}(n)$ *s.t. the following hold:* (i) $\Gamma[o, d](t_o) \leq f(n) \cdot (D[o, d](t_o))^{\lambda}$*, and* (ii) $D[o, d](t_o) \leq g(n) \cdot (\Gamma[o, d](t_o))^{1/\lambda}$*. Analogous inequalities hold for the free-flow and the full-congestion metrics* $\underline{D}$ *and* $\overline{D}$*.*

Note that static oracles based on the doubling dimension (e.g., [2]) demand a *constant* value for λ. We relax this by allowing λ to be even a (sufficiently slowly) growing function of n. We also introduce some additional slackness, by allowing divergence from the corresponding powers by polylogarithmic factors. In the rest of the paper we consider sparse TD-instances (i.e., $m \in \mathcal{O}(n)$), compliant with Assumptions 1, 2, 3, and 4.

3 The TRAP approximation method

The *trapezoidal* (`TRAP`) method is a novel algorithm for computing one-to-many $(1+\varepsilon)$-upper-approximations $\overline{\Delta}[\ell, v] : [0, T) \mapsto \mathbb{R}_{>0}$ of $D[\ell, v]$, from a (landmark) vertex ℓ towards all sufficiently distant destinations v. `TRAP` is remarkably simple and works as follows. First, $[0, T)$ is split into $\left\lceil \frac{T}{\tau} \right\rceil$ consecutive length-τ subintervals, where τ is a tuning parameter to be fixed later. Then, for each interval $[t_s, t_f = t_s + \tau) \subseteq [0, T)$, a $(1 + \varepsilon)$-upper-approximation of the projection $D[\ell, v] : [t_s, t_f) \mapsto \mathbb{R}_{>0}$ is constructed. Finally, the concatenation of all these $(1+\varepsilon)$-upper-approximations per subinterval constitutes the requested $(1 + \varepsilon)$-upper-approximation $\overline{\Delta}[\ell, v]$ of $D[o, d] : [0, T) \mapsto \mathbb{R}_{>0}$. Note that, contrary to `BIS`, no assumption is made on the shapes of the min-cost functions to approximate within each subinterval; in particular, no assumption is made on them being concave. `TRAP` only exploits the fact that τ is small, along with Assumption 1 on the boundedness of travel-time slopes. We now describe the upper- and lower-approximations of $D[o, d]$ that we construct in a subinterval $I_k = [t_s = (k - 1)\tau, t_f = k\tau) \subset [0, T)$, $k \in \left[\left\lceil \frac{T}{\tau} \right\rceil\right]$, from a vertex $\ell \in V$ towards some destination $v \in V$. The quality of the upper-approximation depends on the value of τ and the delay values at the endpoints of I_k, as we shall explain shortly. `TRAP` computes the following two functions of $D[\ell, v]$ (cf. Fig. 1): $\forall t \in I_k$, $\overline{\delta}_k[\ell, v](t) = \min\left\{ D[\ell, v](t_f) + \Lambda_{\min} t_f - \Lambda_{\min} t \ , \ D[\ell, v](t_s) - \Lambda_{\max} t_s + \Lambda_{\max} t \right\}$

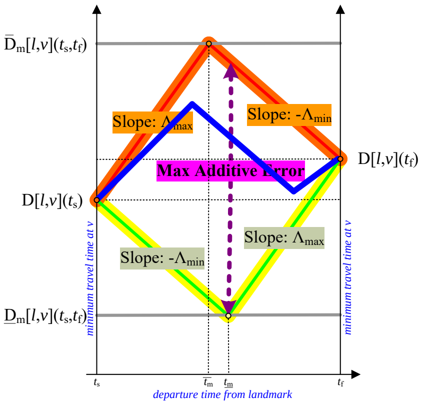


Figure 1 The upper-approximation $\overline{\delta}_k[\ell, v]$ (thick orange, upper pwl line), and lower-approximation $\underline{\delta}_k[\ell, v]$ (thick green, lower pwl line), of the unknown function $D[\ell, v]$ (blue pwl line) within $I_k = [t_s = (k-1)\tau, t_f = k\tau)$.

and $\underline{\delta}_k[\ell, v](t) = \max\left\{\ D[\ell, v](t_f) - \Lambda_{\max} t_f + \Lambda_{\max} t\ ,\ D[\ell, v](t_s) + \Lambda_{\min} t_s - \Lambda_{\min} t\ \right\}$ and considers them as the upper- and lower-approximating functions of $D[\ell, v]$ within I_k. The correctness of this choice is proved in the next lemma, which follows by Assumption 1.

▶ **Lemma 5.** *$\overline{\delta}_k[\ell, v](t)$ and $\underline{\delta}_k[\ell, v](t)$ upper- and lower-approximate $D[\ell, v](t)$ within I_k.*

Let $(\underline{t}_m, \underline{D}_m)$ and $(\overline{t}_m, \overline{D}_m)$ be the intersections of the legs in the definitions of $\underline{\delta}_k[\ell, v]$ and $\overline{\delta}_k[\ell, v]$, respectively. The *maximum additive error* in I_k (c.f. Figure 1) is $MAE(I_k) := \max_{t \in I_k} \left\{\overline{\delta}_k[\ell, v](t) - \underline{\delta}_k[\ell, v](t)\right\} = \overline{\delta}_k[\ell, v](\underline{t}_m) - \underline{\delta}_k[\ell, v](\underline{t}_m)$. The following lemma proves that, for τ sufficiently small, $MAE(I_k)$ cannot be large. It also provides a *sufficient condition* for the value of τ so that $\overline{\delta}_k[\ell, v]$ is a $(1 + \varepsilon)$-upper-approximation of $D[\ell, v]$ in I_k.

▶ **Lemma 6.** *$\forall(\ell, v) \in L \times V$, $\forall k \in \left[\left\lceil \frac{T}{\tau} \right\rceil\right]$ and $I_k = [(k-1)\tau, k\tau)$, the following hold: (1) $MAE[\ell, v](I_k) \leq \Lambda_{\max} \cdot \tau$; (2) $\overline{\delta}_k[\ell, v]$ is a $(1+\varepsilon)$-upper-approximation of $D[\ell, v]$ within I_k, if $\left[\ D[\ell, v](t_s) \geq \left(\Lambda_{\min} + \frac{\Lambda_{\max}}{\varepsilon}\right) \cdot \tau\ \right] \vee \left[\ D[\ell, v](t_f) \geq \left(1 + \frac{1}{\varepsilon}\right) \Lambda_{\max} \cdot \tau\ \right]$*

For given $\tau > 0$ and $\ell \in L$, the set of *faraway destinations* from ℓ is $V[\ell](\tau) = \{v \in V : \tau[\ell, v] > \tau\}$. $\tau[\ell, v] = \frac{\underline{D}[\ell,v]}{(1+1/\varepsilon)\Lambda_{\max}}$ is a sufficient τ-value for $\overline{\delta}_k[\ell, v]$ being $(1 + \varepsilon)$-upper-approximation of $D[\ell, v]$ within $I_k = [(k-1)\tau[\ell, v], k\tau[\ell, v])$ (cf. Lemma 6). The next theorem proves that TRAP provides a $(1 + \varepsilon)$-upper-approximation $\overline{\Delta}[\ell, v]$ for all faraway destinations from ℓ, and also estimates the preprocessing requirements of the algorithm.

▶ **Theorem 7.** *Fix $\ell \in L$, $F > f(n)$, and $\tau \in (0, T)$ s.t. $|V[\ell](\tau)| = n - F$. Let $\tau^* = \min_{v \in V[\ell](\tau)} \left\{\frac{\underline{D}[\ell,v]}{(1+1/\varepsilon)\Lambda_{\max}}\right\}$. $\forall v \in V[\ell](\tau)$, $\overline{\Delta}[\ell, v]$ is the concatenation of all the upper-approximating functions $\overline{\delta}_k[\ell, v]$ that* TRAP *returns per subinterval $I_k = \left[\ t_{s_k} = (k-1)\tau^*\ ,\ t_{f_k} = \min\{k\tau^*, T\}\ \right) : k \in \left[\left\lceil \frac{T}{\tau^*} \right\rceil\right]$. Then, $\forall v \in V[\ell](\tau)$, $\overline{\Delta}[\ell, v]$ is a $(1 + \varepsilon)$-upper-approximation of $D[\ell, v]$ in $[0, T)$, requiring PRE SPACE... at most $2\left\lceil \frac{T}{\tau^*} \right\rceil$ breakpoints. PRE TIME... The number of calls to $TDSP(\ell, \star, t)$ for their construction is $\left\lceil \frac{T}{\tau^*} \right\rceil \leq 1 + \frac{T(1+1/\varepsilon)\Lambda_{\max}}{\min_{v \in V[\ell](\tau)}\{\underline{D}[\ell,v]\}} \in \mathcal{O}(n^{\alpha})$.*

Proof of Theorem 7. τ^* is the appropriate length for the subintervals which assures that TRAP returns $(1 + \varepsilon)$-upper-approximations for all faraway destinations from ℓ. By definition it holds that $\tau^* \geq \tau$. Since $F > f(n)$, it holds that TRAP does not consider destinations at free-flow distance less than 1. To see this, fix $v \in V$ s.t. $\underline{D}[\ell, v] \leq 1$. By Assumption 4,

$\underline{\Gamma}[\ell, v] \leq f(n) \cdot \underline{D}[\ell, v]^{\lambda} \leq f(n) < F$. Thus, we can be sure that $v \notin V[\ell](\tau)$. Since $T = n^{\alpha}$, we conclude that $\frac{T}{\tau^*} = \frac{T(1+1/\varepsilon)\Lambda_{\max}}{\min_{v \in V[\ell](\tau)} \underline{D}[\ell,v]} \in \mathcal{O}(n^{\alpha})$. We proceed now with the analysis of `TRAP`. $[0, T)$ is split into $\left\lceil \frac{T}{\tau^*} \right\rceil$ consecutive length-τ^* subintervals. Lemma 5 assures that for each $I_k = [k\tau^*, (k+1)\tau^*)$ an upper-approximating function $\overline{\delta}_k[\ell, v]$ of $D[\ell, v]$ is determined, for each $v \in V[\ell](\tau)$. The concatenation of all these functions constitutes the upper-approximating function $\overline{\Delta}[\ell, v]$ for $D[\ell, v]$ within $[0, T)$. Since $\tau[\ell, v] \geq \tau^* \Rightarrow \underline{D}[\ell, v] \geq \left(1 + \frac{1}{\varepsilon}\right) \Lambda_{\max}\tau^*$, we deduce (cf. Lemma 6) that, for all $v \in V[\ell](\tau)$, the produced upper-approximations within the consecutive length-τ^* intervals are $(1 + \varepsilon)$-approximations of $D[\ell, v]$. `TRAP` preprocesses $\ell \in L$ by making $\left\lceil \frac{T}{\tau^*} \right\rceil \in \mathcal{O}(n^{\alpha})$ calls to $TDSP(\ell, \star, t)$, to sample the endpoints of all the $\left\lceil \frac{T}{\tau^*} \right\rceil$ length-τ^* subintervals. For storing $\overline{\Delta}[\ell, v]$, it needs $2\left\lceil \frac{T}{\tau^*} \right\rceil$ breakpoints (at most one intermediate breakpoint $(\overline{t}_m, \overline{D}_m)$ per subinterval). ◀

4 Oracles with fully-informed landmarks

We now describe two novel oracles with landmarks possessing summaries for all reachable destinations, excluding possibly a small neighborhood around them. We start with a random landmark set $L \subset_{\mathbf{uar}(\rho)} V$, i.e., we decide independently and uniformly at random whether each vertex is a landmark, with probability $\rho = n^{-\omega}$ for a constant $\omega \in (0, 1)$. We consider as *faraway vertices* from $\ell \in L$, all the vertices at free-flow distance at most $\underline{R} = T^{\theta}$ from it, for a constant $\theta \in (0, 1)$ to be determined later. $F = \max_{\ell \in L} \{|\underline{B}[\ell; \underline{R}]|\}$ is the maximum number of faraway vertices from a landmark. The next lemma shows that the main parameters we should consider w.r.t. a TD-instance are λ (cf. Assumption 4) and $\alpha \in (0, 1)$ s.t. $T = n^{\alpha}$. All the other parameters essentially adjust their values to them.

▶ **Lemma 8.** *For $\nu \in (0, 1)$ s.t. $T = diam(G, \underline{D})^{1/\nu}$, $\theta \in (0, 1)$ s.t. $\frac{\nu}{\theta} \in \mathcal{O}(1)$ and λ, f, g defined as in Assumption 4, the following hold:* (i) $\frac{1}{\lambda\nu} = \alpha \pm \mathrm{o}(1)$, *and* (ii) $F \in n^{[1 \pm \mathrm{o}(1)]\theta/\nu}$.

The `TRAPONLY` oracle. A first attempt towards avoiding the dependency of the preprocessing requirements on K^* is to develop an oracle, called `TRAPONLY`, whose preprocessing is based solely on `TRAP`.TRAPONLY PREPROCESSING... The preprocessing of `TRAPONLY` first considers as subinterval length the value $\tau = \frac{\underline{R}}{(1+1/\varepsilon)\Lambda_{\max}} > 0$. It then calls `TRAP` for each landmark $\ell \in L$, which guarantees $(1 + \varepsilon)$-upper-approximations for all the faraway destinations $v \in V[\ell](\tau)$ (cf. Theorem 7).RQA+... The distances of nearby destinations from ℓ are left to be computed by the query algorithm of `TRAPONLY`, which is an appropriate variant of `RQA` (we call it `RQA`$^+$) which additionally grows a small `TDD` ball of size $F\,\mathrm{polylog}(F)$ (cf. Assumption 3) from each newly settled landmark. TRAPONLY COMPLEXITY... The following theorem analyzes the performance of `TRAPONLY`.

▶ **Theorem 9.** *The expected time of* `RQA`$^+$ *and the preprocessing requirements of* `TRAPONLY` *are:* $\mathbb{E}\left\{Q_{\texttt{RQA}^+}\right\} \in \mathcal{O}\left(n^{\omega r + \max\left\{\omega, \frac{\theta}{\nu}\right\} + \mathrm{o}(1)}\right)$ *and* $S_{\texttt{TRAPONLY}}$, $P_{\texttt{TRAPONLY}} \in \mathcal{O}\left(n^{2+\alpha\cdot(1-\theta)-\omega+\mathrm{o}(1)}\right)$.

Proof of Theorem 9. During the preprocessing, `TRAPONLY` makes $\left\lceil \frac{T}{\tau^*} \right\rceil \leq 1 + \frac{T(1+1/\varepsilon)\Lambda_{\max}}{\underline{R}} = 1 + T^{1-\theta}(1 + 1/\varepsilon)\Lambda_{\max}$ calls of `TDD`(ℓ, t), for departure-times $t \in \left\{0, \tau^*, 2\tau^*, \ldots, \left\lceil \frac{T}{\tau^*} \right\rceil - 1\right\}$ and landmarks $\ell \in L$, where the equality comes from Lemma 8. Therefore, the preprocessing-time is dominated by the aggregate time for all these `TDD` probes. Taking into account that each `TDD` probe takes time $\mathcal{O}(n \log(n) \log\log(K_{\max}))$ and that $|L| = \rho n = n^{1-\omega}$ landmarks, by using Lemma 8 we get the following: $P_{\texttt{TRAPONLY}} = n^{1-\omega} \cdot n^{\frac{1-\theta}{\nu\lambda}[1+\mathrm{o}(1)]} \cdot n \log(n) \log\log(n) \in n^{2-\omega+\frac{1-\theta}{\nu\lambda}[1+\mathrm{o}(1)]+\frac{\log\log(n)+\log\log\log(n)}{\log(n)}} = n^{2-\omega+\alpha\cdot(1-\theta)+\mathrm{o}(1)}$. The calculations are analogous for the required preprocessing space: For all landmarks $\ell \in L$

and all their faraway destinations $v \in V[\ell](\tau)$, the total number of breakpoints to store is at most $S_{\texttt{TRAPONLY}} = 2\left\lceil \frac{T}{\tau^*} \right\rceil \rho n^2 \in n^{2-\omega+\frac{1-\theta}{\nu\lambda}[1+o(1)]+o(1)} = n^{2-\omega+\alpha\cdot(1-\theta)+o(1)}$. As for the query-time complexity of $\texttt{RQA}^+$, recall that the expected number of $\texttt{TDD}$ balls that it grows is $(1/\rho)^r$. Additionally, $\texttt{RQA}^+$ grows $(1/\rho)^r$ $\texttt{TDD}$ balls from the corresponding closest landmarks. Each ball from a new center costs $\mathcal{O}((1/\rho)\log(1/\rho))$. Each ball from a landmark costs $\mathcal{O}(F\,\mathrm{polylog}(F)) \in n^{[1\pm o(1)]\theta/\nu}$. Thus, the expected query-time is upper-bounded as follows: $\mathbb{E}\{Q_{\texttt{RQA}^+}\} \in \mathcal{O}((1/\rho)^r[(1/\rho)\log(1/\rho) + F\,\mathrm{polylog}(F)]\log\log(K_{\max})) = \mathcal{O}\left(n^{\omega r+\max\{\omega,[1+o(1)]\theta/\nu\}}\right)$. ◀

The next corollaries are parameter-tuning examples showcasing the trade-offs among the sublinearity of query-time, the subquadratic preprocessing requirements and the stretch.

▶ **Corollary 10.** *For $\delta \in (\alpha, 1)$, $\beta \in (0, \alpha^2\nu]$, $\omega = \frac{\delta}{r+1}$, $\theta = \frac{\delta\nu}{r+1}$ and $r = \left\lfloor \frac{\delta\cdot(1+\alpha\nu)}{\alpha+\beta} \right\rfloor - 1$, $S_{\texttt{TRAPONLY}}, P_{\texttt{TRAPONLY}} \in n^{2-\beta+o(1)}$, $\mathbb{E}\{Q_{\texttt{RQA}^+}\} \in n^{\delta+o(1)}$ and the stretch is $1+\sigma(r) = 1+\varepsilon\cdot\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$.*

▶ **Corollary 11.** *For any integer $k \geq 2$, let $\eta(k) = \left\lceil \frac{\log(k/(k-1))}{\log(1+\varepsilon/\psi)} \right\rceil - 1$, $\delta \in (0,1)$ and $\beta \in \left(0, \frac{\delta}{\eta(k)+2}\right)$. Then $\texttt{TRAPONLY}$ achieves stretch $1+k\cdot\varepsilon$ with $S_{\texttt{TRAPONLY}}, P_{\texttt{TRAPONLY}} \in n^{2-\beta+o(1)}$ and $\mathbb{E}\{Q_{\texttt{RQA}^+}\} \in n^{\delta+o(1)}$, by scaling the TD-instance so that $T = n^\alpha$, for $\alpha = \frac{\delta-[\eta(k)+2]\cdot\beta}{\eta(k)+2-\delta\nu}$.*

The `FLAT` oracle. Our second attempt, the `FLAT` oracle, provides preprocessed information for all reachable destinations per landmark, and uses the query algorithm `RQA` [17]. PRE: BIS+TRAP... `FLAT` considers again as subinterval length the value $\tau = \frac{R}{(1+1/\varepsilon)\Lambda_{\max}} > 0$. Then, it constructs summaries for all reachable destinations per landmark $\ell \in L$ exploiting both `BIS` and `TRAP`: `BIS` handles all the (at most $F = \max_{\ell\in L}\{|\underline{B}[\ell;\underline{R}]|\}$) nearby destinations in $\underline{B}[\ell;\underline{R}]$, whereas `TRAP` handles all the faraway destinations of $V \setminus \underline{B}[\ell;\underline{R}]$. The space requirements for the summaries created by `TRAP` are exactly the same as in `TRAPONLY`. As for the summaries computed by `BIS`, we avoid the linear dependence of `BIS` on K^* by assuring that F is sufficiently small (but not too small) and exploiting Assumption 3 which guarantees that the involved subgraph $\underline{B}'[\ell;F]$ in the preprocessing phase of `BIS` on behalf of ℓ has size $\mathcal{O}(F\,\mathrm{polylog}(F))$. The next lemma shows that `BIS` is affected only by the concavity-spoiling breakpoints of arc-travel-time functions in $\underline{B}'[\ell;F]$, rather than the entire graph.

▶ **Lemma 12.** $\forall(\ell,v) \in L \times \underline{B}[\ell;F], \forall u \in V \setminus \underline{B}'[\ell;F], \forall t \in [0,T), D[\ell,v](t) < D[\ell,u](t)$.

Proof of Lemma 12. From the definitions of the involved free-flow and full-congestion Dijkstra balls, the following holds: $D[\ell,v](t) \leq \overline{D}[\ell,v] \leq \overline{R}[\ell] < \underline{D}[\ell,u] \leq D[\ell,u](t)$. ◀

The following theorem summarizes the complexities of the `FLAT` oracle.

▶ **Theorem 13.** *The query-time $Q_{\texttt{RQA}}$ and the preprocessing time $P_{\texttt{FLAT}}$ and space $S_{\texttt{FLAT}}$ of* `FLAT` *are: $\mathbb{E}\{Q_{\texttt{RQA}}\} \in \mathcal{O}\left(n^{\omega(r+1)+o(1)}\right)$ and $P_{\texttt{FLAT}}$, $S_{\texttt{FLAT}} \in \mathcal{O}\left(n^{1-\omega+o(1)}\cdot[n^{2\theta/\nu}+n^{1+\alpha\cdot(1-\theta)}]\right)$.*

Proof of Theorem 13. `BIS` requires space at most $F^2\,\mathrm{polylog}(F)$, since by Lemma 12 the involved graph only contains $F\,\mathrm{polylog}(F)$ vertices and concavity-spoiling breakpoints at the arc-travel-time functions. For the faraway vertices of $V \setminus \underline{B}[\ell;F]$, since $\tau = \frac{R}{(1+1/\varepsilon)\Lambda_{\max}}$, `TRAP` provides $(1+\varepsilon)$-approximate summaries for all destinations $v \in V \setminus \underline{B}[\ell;\underline{R}]$, because the sufficient condition of Theorem 7 holds: $\underline{D}[\ell,v] > \underline{R} = (1+1/\varepsilon)\,\Lambda_{\max}\tau$. Thus, we conclude that $S_{\texttt{FLAT}} \in \rho n\left[F^2\,\mathrm{polylog}(F) + \frac{T(1+1/\varepsilon)\Lambda_{\max}n}{\underline{R}}\right] \overset{/*\text{ L.8 }*/}{=} n^{1-\omega}\left[n^{(2\theta/\nu)\cdot[1+o(1)]} + n^{1+\alpha\cdot(1-\theta)[1+o(1)]}\right]$ $= n^{1-\omega+[1+o(1)]\cdot\max\{\ 2\theta/\nu\ ,\ 1+\alpha(1-\theta)\ \}+o(1)}$, since $f(n), g(n) \in \mathrm{polylog}(n)$. ◀

The next corollaries are parameter-tuning examples to showcase the effectiveness of `FLAT`.

▶ **Corollary 14.** *If* $\delta \in (\alpha, 1)$*,* $\beta \in \left(0, \frac{\alpha\cdot(1+\alpha)}{2/\nu+\alpha}\right]$*,* $\omega = \frac{\delta}{r+1}$*,* $r = \left\lfloor \frac{\delta}{\alpha}\cdot\frac{2/\nu+\alpha}{(\beta/\alpha)\cdot(2/\nu+\alpha)+(2/\nu-1)}\right\rfloor - 1$ *and* $\theta = \frac{1+\alpha}{2/\nu+\alpha}$*, then* `FLAT` *has* $P_{\texttt{FLAT}}, S_{\texttt{FLAT}} \in n^{2-\beta+o(1)}$*,* $\mathbb{E}\{Q_{\texttt{RQA}}\} \in n^{\delta+o(1)}$ *and stretch* $1+\sigma(r) = 1+\varepsilon\cdot\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$.

▶ **Corollary 15.** *For integer* $k \geq 2$*, let* $\eta(k) = \left\lceil \frac{\log(k/(k-1))}{\log(1+\varepsilon/\psi)}\right\rceil - 1$ *and* $\delta \in (0,1)$*.* `FLAT` *achieves a target stretch* $1+k\cdot\varepsilon$ *with preprocessing requirements* $n^{2-o(1)}$ *and expected query-time* $n^{\delta+o(1)}$*, by scaling the TD-instance so that* $T = n^{\alpha}$ *for* $\alpha = \frac{2\delta}{[\eta(k)+2]\cdot(2-\nu)-\delta\nu}$*, as* $\beta \downarrow 0$*.*

Comparison of `TRAPONLY` and `FLAT`. Both `TRAPONLY` and `FLAT` depend on the travel-time metric, but are independent of the degree of disconcavity K^*. On one hand, `TRAPONLY` is a simpler oracle, at least w.r.t. its preprocessing phase. On the other hand, `FLAT` achieves a better approximation for the same TD-instance and anticipations for sublinear query-time n^{δ} and subquadratic preprocessing requirements $n^{2-\beta}$. This is because, as $\beta \downarrow 0$, `FLAT` guarantees a recursion budget r of (roughly) $\frac{2\delta}{a} - 1$, whereas `TRAPONLY` achieves about half this value and r has an exponential effect on the stretch that the query algorithms achieve.

5 The `HORN` oracle

We now describe and analyze the *Hierarchical ORacle for time-dependent Networks* (`HORN`), whose query algorithm is highly competitive against `TDD`, not only for long-range queries (i.e., having Dijkstra-Rank proportional to the network size) but also for medium- and short-range queries, while ensuring *subquadratic* preprocessing space and time. The main idea of `HORN` is to preprocess: many landmarks, each possessing summaries for a few destinations around them, so that all short-range queries can be answered using only these landmarks; fewer landmarks possessing summaries for more (but still not all) destinations around them, so that medium-range queries be answered by them; and so on, up to only a few landmarks (those required by `FLAT`) possessing summaries for all reachable destinations. The *area of coverage* $C[\ell] \subset V$ of ℓ is the set of its nearby vertices, for which ℓ possesses summaries. ℓ is called *informed* for each $v \in C[\ell]$, and *uninformed* for each $v \in V \setminus C[\ell]$. The landmarks are organized in a hierarchy, according to the sizes of their areas of coverage. Each level L_i in the hierarchy is accompanied with a *targeted Dijkstra-Rank* $N_i \in [n]$, and the goal of `HORN` is to assure that L_i should suffice for `RQA` to successfully address queries (o, d, t_o) with $\Gamma[o, d](t_o) \leq N_i$, in time $o(N_i)$. The difficulty of this approach lies in the analysis of the query algorithm. We want to execute a variant of `RQA` which, based on a *minimal* subset of landmarks, would guarantee a $(1 + \sigma(r))$-approximate solution for any query (o, d, t_o) (as in `TRAPONLY` and `FLAT`), but also time-complexity sublinear in $\Gamma[o, d](t_o)$. We propose the *Hierarchical Query Algorithm* (`HQA`) which grows an initial `TDD` ball from (o, t_o) that stops only when it settles an informed landmark ℓ w.r.t. d which is at the "right distance" from o, given the density of landmarks belonging to the same level with ℓ. `HQA` essentially "guesses" as appropriate level-i in the hierarchy the level that contains ℓ, and continues with the execution of `RQA` with landmarks having coverage at least equal to that of ℓ (cf. Fig. 2).

Initialization of `HORN`. We use the following parameters for the hierarchical construction: (i) $k \in \mathcal{O}(\log\log(n))$ determines the number of levels (minus one) comprising the hierarchy of landmarks. (ii) $\gamma > 1$ determines the actual values of the targeted Dijkstra-Ranks, one per level of the hierarchy. For example, as γ gets closer to 1, the targeted Dijkstra-Ranks

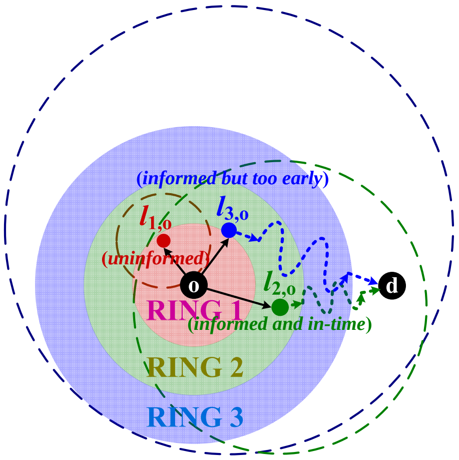


Figure 2 Demonstration of execution of HQA. Dashed circles indicate areas of coverage. Solid circular stripes indicate the rings of the corresponding levels in the hierarchy. Landmark $\ell_{1,o}$ is uninformed and $\ell_{3,o}$, although informed, comes too early. $\ell_{2,o}$ is both informed and within the ring of its own level, leading HQA to deduce that the appropriate level is $i = 2$.

accumulate closer to small- and medium-rank queries. (iii) $\delta \in (0, 1)$ is the parameter that quantifies the sublinearity of the query algorithm (HQA), in each level of the hierarchy, compared to the targeted Dijkstra-Rank of this level. In particular, if N_i is the targeted Dijkstra-Rank corresponding to level-i in the hierarchy, then HQA should be executed in time $\mathcal{O}\big((N_i)^{\delta}\big)$, if only the landmarks in this level (or in higher levels) are allowed to be used.

Preprocessing of HORN. $\forall i \in [k]$, we set the targeted Dijkstra-Rank for level-i to $N_i = n^{(\gamma^i-1)/\gamma^i}$. Then, we construct a randomly chosen level-i landmark set $L_i \subset_{\mathbf{uar}(\rho_i)} V$, where $\rho_i = N_i^{-\delta/(r+1)} = n^{-\delta(\gamma^i-1)/[(r+1)\gamma^i]}$. Each $\ell_i \in L_i$ acquires summaries for all (and only those) $v \in C[\ell_i]$, where $C[\ell_i]$ is the smallest *free-flow* ball centered at ℓ_i containing $c_i = N_i \cdot n^{\xi_i} = n^{(\gamma^i-1)/\gamma^i+\xi_i}$ vertices, for a sufficiently small constant $\xi_i > 0$. The summaries to the $F_i = c_i^{\chi}$ nearby vertices around ℓ_i are constructed with BIS; the summaries to the remaining $c_i - F_i$ faraway vertices of ℓ_i are constructed with TRAP, where $\chi = \frac{\theta}{\nu} = \frac{1+\alpha}{2+\alpha\nu} \in \left[\frac{1}{2}, \frac{2}{2+\nu}\right]$ is an appropriate value determined to assure the correctness of FLAT w.r.t. the level-i of the hierarchy. An ultimate level $L_{k+1} \subset_{\mathbf{uar}(\rho_{k+1})} V$ of landmarks, with $\rho_{k+1} = n^{-\frac{\delta}{r+1}}$, assures that HORN is also competitive against queries with Dijkstra-Rank greater than $n^{(\gamma^k-1)/\gamma^k}$. We choose in this case $c_{k+1} = N_{k+1} = n$, $F_{k+1} = n^{\chi}$ and $C[\ell_{k+1}] = V$, $\forall \ell_{k+1} \in L_{k+1}$.

Description of HQA. A TDD ball from (o, t_o) is grown until d is settled, or the *(ESC)*-criterion or the *(ALH)*-criterion is fulfilled (whichever occurs first):

- **Early Stopping Criterion (ESC):** $\ell_o \in L = \cup_{i\in[k+1]} L_i$ is settled, which is informed ($d \in C[\ell_o]$) and, for $\varphi \geq 1$, $\frac{\overline{\Delta}[\ell_o,d](t_o+D[o,\ell_o](t_o))}{D[o,\ell_o](t_o)} \geq (1+\varepsilon)\cdot\varphi\cdot(r+1)+\psi-1\,.$
- **Appropriate Level of Hierarchy (ALH):** For some level $i \in [k]$ of the hierarchy, the first landmark $\ell_{i,o} \in L_i$ is settled such that: (i) $d \in C[\ell_{i,o}]$ ($\ell_{i,o}$ is "informed"); and (ii) $\frac{N_i^{\delta/(r+1)}}{\ln(n)} \leq \Gamma[o, \ell_{i,o}](t_o) \leq \ln(n)\cdot N_i^{\delta/(r+1)}$ ($\ell_{i,o}$ is at the "right distance"). In that case, HQA concludes that i is the "appropriate level" of the hierarchy to consider. Observe that the level-$(k + 1)$ landmarks are always informed. Thus, if no level-$(\leq k)$ informed landmark is discovered at the right distance, then the first level-$(k + 1)$ landmark that

will be found at distance larger than $\ln(n) \cdot N_k^{\delta/(r+1)}$ will be considered to be at the right distance, and then HQA concludes that the appropriate level is $k+1$.

If d is settled, an exact solution is returned. If (ESC) causes termination of HQA, the value $D[o,\ell_o](t_o) + \overline{\Delta}[\ell_o, d](t_o + D[o,\ell_o](t_o))$ is reported. Otherwise, HQA stops the initial ball due to the (ALH)-criterion, considering $i \in [k+1]$ as the appropriate level, and then continues executing the variant of RQA, call it $\texttt{RQA}_i$, which uses as its landmark set $M_i = \cup_{j=i}^{k+1} L_j$. Observe that $\texttt{RQA}_i$ may fail constructing approximate solutions via certain landmarks in M_i that it settles, since they may not be informed about d. Eventually, HQA returns the best *od*-path (w.r.t. the approximate travel-times) among the ones discovered by $\texttt{RQA}_i$ via *all* settled and informed landmarks ℓ. Theorem 16 summarizes the performance of HORN.

▶ **Theorem 16.** *Consider any TD-instance with* $\lambda \in \mathrm{o}\Big(\sqrt{\frac{\log(n)}{\log\log(n)}}\Big)$ *and* $g(n), f(n) \in \mathrm{polylog}(n)$ *(cf. Assumption 4). For* $\varphi = \frac{\varepsilon\cdot(r+1)}{\psi\cdot(1+\varepsilon/\psi)^{r+1}-1}$ *and* $k \in \mathcal{O}(\log\log(n))$*, let* $\xi_i \in \left(\left[(1+\lambda)\cdot\log\log(n) + \lambda\log\left(1+\frac{\zeta}{1-\Lambda_{\min}}\right)\right] / \log(n)\ ,\ 1-\gamma^{-i}\right)$*, for all* $i \in [k]$*. Then, for any query* (o,d,t_o) *s.t.* $N_{i^*-1} < \Gamma[o,d](t_o) \leq N_{i^*}$ *for some* $i^* \in [k+1]$*, any* $\delta \in (\alpha, 1)$*,* $\beta > 0$*, and* $r = \left\lfloor \frac{\delta}{\alpha} \cdot \frac{(2/\nu+\alpha)(1-\gamma)}{\beta\cdot(2/(\alpha\nu)+1)+2/\nu-1} \right\rfloor - 1$*, HORN achieves* $\mathbb{E}\{Q_{\texttt{HQA}}\} \in (N_{i^*})^{\delta+\mathrm{o}(1)}$*,* $P_{\texttt{HORN}}\ ,\ S_{\texttt{HORN}} \in n^{2-\beta+\mathrm{o}(1)}$ *and stretch* $1+\varepsilon\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$*, with probability at least* $1-\mathcal{O}\big(\frac{1}{n}\big)$.

References

1 R. Agarwal, P. Godfrey. Distance oracles for stretch less than 2. *ACM-SIAM SODA*:526–538 (2013)

2 Y. Bartal, L. A. Gottlieb, T. Kopelowitz, M. Lewenstein, L. Roditty. Fast, precise and dynamic distance queries. *ACM-SIAM SODA*:840-853 (2011)

3 H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, R. Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, http://arxiv.org/abs/1504.05140 (2015)

4 G. V. Batz, R. Geisberger, P. Sanders, C. Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM J. of Exp. Algorithmics*, **18**:1–43 (2013)

5 K. Cooke, E. Halsey. The shortest route through a network with time-dependent intermodal transit times. *Math. Anal. and Appl.*, **14**(3):493-498 (1966)

6 B. C. Dean. Algorithms for minimum-cost paths in time-dependent networks with waiting policies. *Networks*, **44**(1):41–46 (2004)

7 B. C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical Report, MIT (2004)

8 F. Dehne, O. T. Masoud, and J. R. Sack. Shortest paths in time-dependent FIFO networks. *Algorithmica*, **62**(1-2):416–435 (2012)

9 D. Delling. Time-Dependent SHARC-Routing. *Algorithmica*, **60**(1):60–94 (2011)

10 D. Delling, D. Wagner. Time-dependent route planning. *Robust and Online Large-Scale Optimization*, **LNCS 5868**:207–230, Springer (2009)

11 S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, **17**(3):395–412 (1969)

12 L. Foschini, J. Hershberger, S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, **68**(4):1075–1097 (2014)

13 J. Halpern. Shortest route with tme dependent length of edges and limited delay possibilities in nodes. *Zeitschrifl für Operations Research*, **21**:117–124 (1977)

14 S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. Analysis and experimental evaluation of time-dependent distance oracles. *ALENEX*:147–158 (2015)

15 S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. Engineering oracles for time-dependent road networks. *ALENEX*:1–14 (2016)

16 S. Kontogiannis, D. Wagner, C. Zaroliagis. Hierarchical Time-Dependent Oracles. CoRR `http://arxiv.org/abs/1502.05222` (2015)

17 S. Kontogiannis, C. Zaroliagis. Distance oracles for time-dependent networks. *Algorithmica*, **74**(4):1404-1434 (2016).

18 G. Nannicini, D. Delling, L. Liberti, D. Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, **59**:240–251 (2012)

19 M. Omran, J. R. Sack. Improved approximation for time-dependent shortest paths. *COCOON*, **LNCS 8591**:453-464, Springer (2014)

20 A. Orda, R. Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *J. of the ACM*, **37**(3):607–625 (1990)

21 M. Patrascu, L. Roditty. Distance oracles beyond the Thorup–Zwick bound. *IEEE FOCS*:815-823 (2010)

22 E. Porat, L. Roditty. Preprocess, set, query! *ESA*, **LNCS 6942**:603-614, Springer (2011)

23 H. Sherali, K. Ozbay, S. Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, **31**(4):259–272 (1998)

24 C. Sommer. Shortest-path queries in static networks. *ACM Comp. Surv.*, **46** (2014)

25 C. Sommer, E. Verbin, W. Yu. Distance oracles for sparse graphs. *IEEE FOCS*:703–712 (2009)

26 M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM*, **51**(6):993–1024 (2004)

27 M. Thorup, U. Zwick. Approximate distance oracles. *J. of the ACM*, **52**(1):1–24 (2005)

28 C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. *ACM-SIAM SODA*:202–208 (2012)

29 C. Wulff-Nilsen. Approximate distance oracles with improved query time. *ACM-SIAM SODA*:539–549 (2013)

A Refined Definition for Groups of Moving Entities and its Computation

Marc van Kreveld[1], **Maarten Löffler**[*2], **Frank Staals**[†3], and **Lionov Wiratma**[‡4]

1 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands**
`m.j.vankreveld@uu.nl`

2 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands**
`m.loffler@uu.nl`

3 **MADALGO, Aarhus University, Aarhus, Denmark**
`f.staals@cs.au.dk`

4 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands; and**
Dept. of Informatics, Parahyangan Catholic University, Indonesia
`l.wiratma@uu.nl`, `lionov@unpar.ac.id`

Abstract

One of the important tasks in the analysis of spatio-temporal data collected from moving entities is to find a *group*: a set of entities that travel together for a sufficiently long period of time. Buchin et al. [2] introduce a formal definition of groups, analyze its mathematical structure, and present efficient algorithms for computing all maximal groups in a given set of trajectories. In this paper, we refine their definition and argue that our proposed definition corresponds better to human intuition in certain cases, particularly in dense environments.

We present algorithms to compute all maximal groups from a set of moving entities according to the new definition. For a set of n moving entities in $\mathbb{R}^1$, specified by linear interpolation in a sequence of τ time stamps, we show that all maximal groups can be computed in $O(\tau^2 n^4)$ time. A similar approach applies if the time stamps of entities are not the same, at the cost of a small extra factor of $\alpha(n)$ in the running time. In higher dimensions, we can compute all maximal groups in $O(\tau^2 n^5 \log n)$ time (for any constant number of dimensions).

We also show that one τ factor can be traded for a much higher dependence on n by giving a $O(\tau n^4 2^n)$ algorithm for the same problem. Consequently, we give a linear-time algorithm when the number of entities is constant and the input size relates to the number of time stamps of each entity. Finally, we provide a construction to show that it might be difficult to develop an algorithm with polynomial dependence on n *and* linear dependence on τ.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases moving entities, trajectories, grouping, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.48

[*] M. L. is partially supported by the Netherlands Organisation for Scientific Research (NWO), under projects 639.021.123 and 614.001.504.

[†] F. S. is partially supported by the Danish National Research Foundation (grant nr. DNRF84).

[‡] L. W. is supported by the Ministry of Research, Technology and Higher Education of Indonesia (138.41/E4.4/2015).

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 48; pp. 48:1–48:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Nowadays, inexpensive modern devices with advanced tracking technologies make it easy to track movements of an entity. This has led to the availability of movement data for various types of moving entities (human, animals, vehicles, etc.). Since a tracking device typically returns a single location at each time stamp, each moving entity will be represented by a moving point. Data may consist of just one trajectory tracked over a period of time, or a whole collection of trajectories that are all tracked over the same time period. Note that for the latter case, the locations of each trajectory are not necessarily collected at the same time stamps. It is common to denote the number of trajectories (or moving entities) by n and the number of time stamps used for each trajectory by τ. Hence, the input size is $\Theta(\tau n)$. Depending on the application, one of n or τ can be much larger than the other.

To analyze moving object data, a number of methods have been developed in recent times. These methods perform similarity analysis or compute a clustering, outliers, a segmentation, or various patterns that may emerge from the movement of the entities (for surveys see [3, 15]). These methods are often based on geometric algorithms, because the data is essentially spatial.

One particular type of pattern that has been well-studied is flocking [1, 4, 5]. Intuitively, a flock is a subset of the entities moving together (or simply being together) over a period of time. Other names for this and closely related concepts with slightly different definitions are herds [6], convoys [8], moving clusters [9], mobile groups [7], swarms [11], and groups [2]. Buchin et al. [2] introduce a model called the *trajectory grouping structure* which not only defines groups, but also the splitting of a group into subgroups and its opposite, merging. The algorithmic problem of reporting all maximal groups that occur in the trajectories is solved in $O(\tau n^3 + N)$ time, where $N \in O(\tau n^4)$ is the output size (the summed size of all groups reported). The algorithm also considers times in between the τ time stamps where the locations are recorded as relevant. In between these time stamps, locations are inferred by linear interpolation over time.

In this paper we continue the study of such groups, but we propose a refined definition to the one by Buchin et al. [2]. We motivate why it captures our intuition better and present algorithms to compute all maximal groups.

Previous definition of a group. The definition of a group by Buchin et al. [2] relies on three parameters: one for the distance between entities, one for the duration of a group, and one for the size of a group. We review their definitions next.

For a set of moving entities $\mathcal{X}$, two entities x and y are *directly ε-connected* at time t if the Euclidean distance between x and y is at most ε at time t, for some given $\varepsilon \geq 0$. Two entities x and y are *ε-connected in $\mathcal{X}$* at time t if there is a sequence $x = x_0, ..., x_k = y$, with $\{x_0, ..., x_k\} \subseteq \mathcal{X}$ and for all i, x_i and x_{i+1} are directly ε-connected at time t.

In [2], a *group* for an entity inter-distance ε, a minimum required duration δ, and a minimum required size m, is defined as a subset $G \subseteq \mathcal{X}$ and corresponding time interval I for which three conditions hold:

(i) G contains at least m entities.
(ii) I has a duration at least δ.
(iii) Every two entities $x, y \in G$ are ε-connected in $\mathcal{X}$ at all times in I.

Furthermore, a group G with time interval I is *maximal* if there is no time interval $I' \supset I$ for which G is also a group, and there is no proper superset $G' \supset G$ that is also a group during I [2].

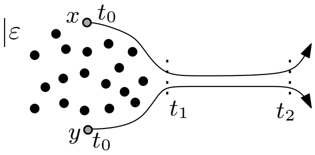


Figure 1 In the definition by [2], x and y are ε-connected during $[t_0, t_2]$.

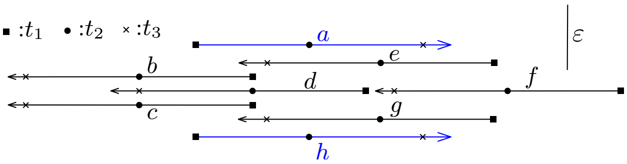


Figure 2 Entities in $G = \{a, h\}$ are ε-connected using entities not in G.

Refined definition of a group. One issue with the previous definition is that it does not correspond fully to our intuition. Two entities x and y may form a (rather small) maximal group in an interval I even if they are always far apart, as long as there are always entities of $\mathcal{X}$ in between them to make x and y ε-connected in $\mathcal{X}$. These entities in between are not part of the maximal group, but they do cause x and y to be ε-connected by the previous definition. This can have counter-intuitive effects especially in dense crowds. To avoid such issues, we refine the definition of a group. In particular, we replace condition (iii) above by: **(iii')** Every two entities $x, y \in G$ are ε-connected in G during I.

We define maximal groups in the same way as before.

We give two examples that show the difference in these definitions.

First, consider a number of stationary entities S and two entities x and y, see Figure 1. Entity x starts (at time t_0) to the North of S and moves around its perimeter to the East. Entity y starts (at t_0) to the South and also moves around the perimeter to the East. After encountering (at t_1) each other at the East side, both continue together eastward, away from the stationary entities in S (ending at t_2). By the definition in [2], x and y form a maximal group in the interval $[t_0, t_2]$. By our refined definition, they form a maximal group during $[t_1, t_2]$, starting when x and y are at distance ε and actually encounter each other.

Second, the previous definition can even see groups of entities that were never close, see Figure 2. Here, $\{a, h\}$ is a maximal group in the interval $I = [t_1, t_3]$ using the definition in [2]. At each time, a and h are ε-connected, but through different subsets of entities. By choosing the coordinates carefully, we can ensure that no supergroup of $\{a, h\}$ is also a group in the same time interval, and hence $\{a, h\}$ will be maximal. Although a and h move in the same direction with the same speed, intuitively they do not form a group because they are too far apart and separated by other entities that move in the opposite direction. With our refined definition, we do not consider $\{a, h\}$ a group in the interval I, and hence also not a maximal group.

Results and Organization. We have refined the previous definition for a group of moving entities by Buchin et al. [2] and gave two examples and argue why our refined definition can give an intuitively plausible group. From now on, we will use the term "group" to denote a group of entities that comply with our refined definition.

In the following section, we show that for a set $\mathcal{X}$ of n moving entities in $\mathbb{R}^1$ with τ time stamps each, the number of maximal groups by the refined definition is $O(\tau n^3)$, which is tight in the worst case.

In Section 3, we present algorithms to compute all maximal groups in $\mathbb{R}^1$. First we consider the case where all trajectories have their vertices at the same time and begin with a basic algorithm for that runs in $O(\tau^3 n^6)$ time. Subsequent improvements lead to a running time of $O(\tau^2 n^4)$. When the time stamps of trajectories are not the same, we show that our algorithm runs in $O(\tau^2 n^4 \alpha(n))$ time.

Next, for moving entities in $\mathbb{R}^d$ $(d > 1)$, we model entities and their inter-distance into graphs and show that all maximal groups can be computed in $O(\tau^2 n^5 \log n)$ time, regardless the uniformity of the time stamps in the trajectories. We show how to achieve this bound in Section 4.

In Section 5, we consider situations where the value of n is significantly smaller than τ, which is typical in real-life moving entity datasets. We give an $O(\tau 2^n n^4)$ time algorithm for entities that move in any constant dimension.

Finally, we show an exponential bound on the number of maximal groups that can contain any given time t in the last section.

2 Preliminaries

Let $\mathcal{X}$ be a set of n entities moving in $\mathbb{R}^1$, given by locations at τ time stamps. A trajectory of an entity in $\mathcal{X}$ can be expressed by a piecewise-linear function which maps time to a point in $\mathbb{R}^1$. If $\mathbb{R}^1$ is associated with the vertical axis and time with the horizontal axis of a 2-dimensional plane, the trajectories of entities in $\mathcal{X}$ are polylines with τ vertices each. We will use the same notation to denote an entity and its trajectory. We assume that there are no two parallel edges of trajectories.

Let $\mathrm{d}_{ij}(t)$ be the Euclidean distance between $i \in \mathcal{X}$ and $j \in \mathcal{X}$ at time t. When $\mathrm{d}_{ij}(t) = \varepsilon$, we say that an ε-*event* occurs. For any ε-event v, we denote by t_v the time when v occurs and $\omega(v)$ the function that returns the two entities that create v. We assume that no two or more ε-events occur at the same time.

Consider an ε-event v; let $\omega(v) = \{i, j\}$. If i and j are further than ε immediately before t_v, then v is a *start* ε-*event*; if they are further immediately after t_v it is an *end* ε-*event*. If there is no entity $k \in \mathcal{X}$ located strictly in between i and j at t_v (so $\mathrm{d}_{ik}(t_v) + \mathrm{d}_{jk}(t_v) = \varepsilon$), then we say that v is a *free* ε-*event*.

▶ **Observation 1.** *The number of* ε*-events is* $O(\tau n^2)$.

Let G be a group of entities in time interval I that is maximal in size. All entities in G are pairwise ε-connected in the interval I, and hence, there are no free ε-events in G during I. In the arrangement of trajectories from G, we define the height of a face as the length of the longest vertical line segment inside the face. Thus, no face has height greater than ε.

It is also clear that G can begin only at a start ε-event and end only at an end ε-event. Furthermore, we observe that if a start ε-event (or end ε-event) of G is not a free ε-event with respect to the entities in G, then before (or after) the interval I, entities in G are still pairwise ε-connected and we can extend the interval of G. Therefore, G can be a maximal group only if both the start ε-event and end ε-event are free ε-events (but this is not a sufficient condition).

▶ **Observation 2.** *There can be at most one maximal group that starts and ends at a particular pair of start* ε*-event and end* ε*-event.*

▶ **Theorem 3.** *For a set* $\mathcal{X}$ *of* n *entities, each entity moving along a piecewise-linear trajectory of* τ *edges, the maximum number of maximal group is* $\Theta(\tau n^3)$.

Proof. Any group G that starts at a start ε-event contains at most n entities. When a free end ε-event involving G occurs, only group G ends but a subgroup of G with fewer entities may continue longer. This can happen at most $n-1$ times. Therefore, the maximum number of maximal groups is $O(\tau n^3)$. Furthermore, there can be $\Omega(\tau n^3)$ maximal groups because the lower bound construction by van Goethem et al. [14] also works for our definition of a group. ◀

The approach to compute all maximal groups is to work on the arrangement $\mathcal{A}$ of line segments that are the trajectories. For a subset $G \subseteq \mathcal{X}$ and interval I, we can remove entities from G that are separated at a face with height larger than ε in I (corresponding to a free ε-event). Only if there are no such faces, the remaining entities in G can be a group. Note that removing entities in G involves removing the corresponding trajectories from the arrangement $\mathcal{A}$, which can cause new faces that are free ε-events.

3 Algorithms for Entities in $\mathbb{R}^1$

In this section, first we consider the case where the trajectories have the same time stamps. We present a basic algorithm that computes all maximal groups in $O(\tau^3 n^6)$ time for entities moving in $\mathbb{R}^1$. Then we present a more efficient algorithm that runs in $O(\tau^2 n^4)$ time. Furthermore, we present an $O(\tau^2 n^4 \alpha(n))$ time algorithm if the vertices of the trajectories have different time stamps.

3.1 Basic Algorithm

We describe a simple algorithm to compute all maximal groups. Let $\mathcal{V}_s$ and $\mathcal{V}_e$ be the sets of all start ε-events and all end ε-events respectively. Fix one event of each type: $\alpha \in \mathcal{V}_s$ and $\beta \in \mathcal{V}_e$. By Observation 2, there is only one maximal group G that starts at α and ends at β. Furthermore, observe that G necessarily contains the entities $\omega(\alpha) = \{a, b\}$ and $\omega(\beta) = \{c, d\}$, and that if G is a maximal group on $I = [t_\alpha, t_\beta]$, then all entities in G are on the same side at time $t_\gamma \in (t_\alpha, t_\beta)$ when a free ε-event γ occurs. We then use the following approach to find G (if it exists):

1. Initialize a set G containing all entities in $\mathcal{X}$.
2. Build an arrangement $\mathcal{A}$ induced by the trajectories of the entities in G on I.
3. A face f in $\mathcal{A}$ contains a free ε-event γ if (and only if) the height of f is more than ε. If f has height larger than ε, test if (the trajectories of) a, b, c, and d, all lie on the same side of f. If not, there is no maximal group G that starts at α and ends at β. If they do pass on the same side, let S denote the set of entities whose trajectories lie on the other side of f. Remove these entities of S from G, and remove their trajectories from $\mathcal{A}$. Observe that new free ε-events may appear because removal of a trajectory from $\mathcal{A}$ merges two faces of $\mathcal{A}$ into a larger one. See Figure 3. Repeat this step until there is no more free ε-event γ with $t_\gamma \in (t_\alpha, t_\beta)$.
4. Check that α and β are now free. If so, G is a maximal group on I, and hence we can report it. If not, G is actually a group during a time interval $I' \supset I$. Hence, G may be maximal in size, but not in duration. We do not report G in this case.

▶ **Theorem 4.** *Given a set $\mathcal{X}$ of n entities in which each entity moves in $\mathbb{R}^1$ along a trajectory of τ edges, all maximal groups can be computed in $O(\tau^3 n^6)$ time using the Basic Algorithm.*

Proof. The number of combination of a pair of start and end ε-events is $O(\tau^2 n^4)$. Building an arrangement from trajectories of entities takes $O(\tau n^2)$ time. Removing a trajectory e

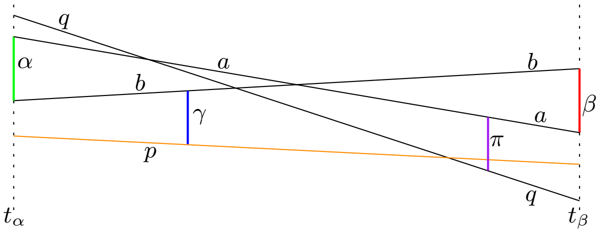


Figure 3 Removing trajectory p (due to the free ε-event γ) causes the ε-event π to become a free ε-event.

and checking new faces in $\mathcal{A}$ takes time proportional to the zone complexity of e: $O(\tau n)$. Since there are at most n trajectories to be removed, the whole process to remove entities for each interval I takes $O(\tau n^2)$ time. Therefore, the running time of the algorithm is $O(\tau^3 n^6)$ time. ◀

3.2 Improved Algorithm

The previous algorithm checks every pair of possible start and end ε-events α and β to potentially find one maximal group. To improve the running time, we fix a start ε-event α and consider the $O(\tau n^2)$ end ε-events β in increasing order. We show that we can check for a maximal group on $[t_\alpha, t_\beta]$ in amortized $O(1)$ time.

We build the arrangement $\mathcal{A}$ for all trajectories, starting from time t_α, and sort the end ε-events β, with $t_\beta > t_\alpha$ on increasing time. We then consider the end ε-events β in this order, while maintaining a maximal set G that is ε-connected in G throughout the time interval $[t_\alpha, t_\beta]$.

Let $\omega(\alpha) = \{a, b\}$ be the entities defining the start ε-event α, and let $G \supseteq \{a, b\}$ be the largest ε-connected set on $[t_\alpha, t_\beta]$. We compute the largest ε-connected set on $[t_\alpha, t_{\beta'}]$ for the next ending event β' as follows. Note that this set will be a subset of G.

Let S be the set of entities that separate from a and b at β. We remove all trajectories from the entities in S from $\mathcal{A}$. As before, this may introduce faces of height larger than ε. For every such face f, we check if a and b still pass f on the same side. If not, there can be no maximal groups that contain a and b, start at t_α, and end after t_β. If a and b lie on the same side of f, we add all entities that lie on the other side of f to S and remove their trajectories from $\mathcal{A}$. We repeat this until all faces in $\mathcal{A}$ that have non-empty intersection with the vertical strip defined by $[t_\alpha, t_{\beta'}]$ have height at most ε (or until we have found a face that splits a and b). It follows that the set $G' = G \setminus S$ is the largest set containing a and b that is ε-connected throughout $[t_\alpha, t_{\beta'}]$. If α and β' are free with respect to G' then we report G' as a maximal group.

Building the arrangement $\mathcal{A}$ takes $O(\tau n^2)$ time, and sorting the ending-events takes $O(\tau n^2 \log(\tau n))$ time. By the Zone Theorem, we can remove each trajectory in $O(\tau n)$ time. Checking the height of the new faces can be done in the same time bound. It follows that the total running time is $O(\tau n^2(\tau n^2 + \tau n^2 \log(\tau n) + R))$ where R is the total time for removing trajectories from the arrangement. Clearly, R is bounded by the complexity of the arrangement: $O(\tau n^2)$. So, the total running time is $O(\tau^2 n^4 \log(\tau n))$.

Further Improvement We can avoid repeated sorting of end ε-events by pre-sorting them in a list, and for each start ε-event, use this list. The list will contain events that do not concern the entities involved in the start ε-event, but this can be tested easily in constant time. Thus, we conclude:

▶ **Theorem 5.** *Given a set $\mathcal{X}$ of n entities in which each entity moves in $\mathbb{R}^1$ along a trajectory of τ edges, all maximal groups can be computed in $O(\tau^2 n^4)$ time.*

Next, we consider finding all maximal groups when the vertices of different trajectories do not have the same time stamps. We use the same idea as in the above algorithm: take one start ε-event α at a time and remove trajectories to find all maximal groups containing $\omega(\alpha)$.

We use a similar strategy to split trajectories vertically into τ cells as in [10], where each cell now contains $O(n)$ segments of trajectories. It follows that the complexity of each cell is bounded by the number of possible intersections between segments: $O(n^2)$. Thus, building the arrangement A still takes $O(\tau n^2)$ time. However, by the Zone Theorem for an arrangement of line segments, removing a trajectory in each cell now takes $O(n\alpha(n))$ time [13], where $\alpha(n)$ is the inverse Ackermann Function. Therefore, the total time to remove trajectories in A is $O(\tau n^2 \alpha(n))$ time and we obtain:

▶ **Theorem 6.** *Given a set $\mathcal{X}$ of n entities in which each entity moves in $\mathbb{R}^1$ along a trajectory of τ edges under the condition that their vertices have different time stamps, all maximal groups can be computed in $O(\tau^2 n^4 \alpha(n))$ time.*

4 Algorithms for Entities in $\mathbb{R}^d$

In $\mathbb{R}^d$ $(d > 1)$, it is harder to test whether an ε-event really connects or disconnects because the two entities may be ε-connected through other entities in the group. This observation immediately gives the condition for an ε-event to be *free*. We model our moving entities as a graph where vertices represent entities and an edge exists if two entities are directly ε-connected. As in Parsa [12], we can maintain the graph under edge updates, while allowing same component queries, in $O(\log n)$ time per operation.

To compute maximal groups, we start at a start ε-event α and maintain the connected component $\mathcal{C}$ throughout the sequence of sorted ε-events. At each ε-event β, we remove any vertices that are disconnected from $\mathcal{C}$ and start again from α in case we remove anything. We stop if a and b are disconnected. If α is a free ε-event when we reach β again, we report $\mathcal{C}$ as a maximal group and continue.

We start at $O(\tau n^2)$ ε-events and for each, we process $O(\tau n^2)$ ε-events. We may need to restart this process up to $n - 1$ times. In $\mathbb{R}^d$, our approach only examine the ε-events of entities and does not affected by whether the vertices of trajectories have the same time or not, therefore we obtain the same result for both cases:

▶ **Theorem 7.** *Given a set $\mathcal{X}$ of n entities moving in $\mathbb{R}^d$ along a trajectory of τ edges, all maximal groups can be computed in $O(\tau^2 n^5 \log n)$ time.*

5 Algorithms with Linear Dependence on τ

In many real-life situations, the number of vertices in each trajectory is much larger than the number of moving entities. Therefore, the dependence of the algorithm on τ is more important than the dependence on n. Next, we show a simple algorithm that is linear in τ, at the cost of an exponential dependence on n. In particular, our algorithm will compute all maximal groups in $O(\tau n^4 2^n)$ time.

We consider all 2^n subsets of $\mathcal{X}$ in order of decreasing size, while maintaining the set of maximal groups found so far (ordered by increasing starting time). For each subset G we determine the maximal time intervals during which G is ε-connected, and for each such an interval I we check if G is dominated by a maximal group $H \supset G$ on I. If such a set does not exist, G is a maximal group on I. Notice that we only need to know when the start ε-event and end ε-event of a particular group occured. Therefore, this algorithm applies to both cases where the time stamps of the entities are not the same.

For each subset G, we consider the ε-events generated by the entities in G. We can compute all these $O(\tau n^2)$ ε-events in $O(\tau n^2 \log n)$ time, by sorting the groups of $O(n^2)$ ε-events between two consecutive time stamps separately, and concatenating the resulting τ lists. We then go through the ε-events in order, and check if G is ε-connected at every ε-event. We can easily handle every event in $O(n^2)$ time, by naively checking if the entities in G are ε connected (we can easily improve on this, but the total running time will be dominated by the number of sets anyway). It follows that we can compute the sequence $\mathcal{S}_G$ of maximal time intervals on which G is ε-connected in $O(\tau n^4)$ time. Note that $\mathcal{S}_G$ contains at most $O(\tau)$ such time intervals.

For each interval I in $\mathcal{S}_G$ we now have to check if G is a maximal group during I. The set G is a maximal group on I if and only if there is no maximal group $H \supset G$ on a time interval that contains I. Since we maintain the maximal groups larger than G (and the time interval on which they are a maximal group), ordered by increasing starting time, we can iterate through them once, and extract the maximal groups that are a superset of G. Since, by Theorem 3 there are at most $O(\tau n^3)$ maximal groups, this takes at most $O(\tau n^4)$ time. Let $\mathcal{I}$ denote the set of time-intervals corresponding to those groups, ordered by increasing starting time. We now simply scan through $\mathcal{S}_G$ and $\mathcal{I}$ simultaneously, while maintaining the time interval in $\mathcal{I}$ that started earliest and has not ended yet. For every interval I in $\mathcal{S}_G$ we can then check if G is a maximal group on I in constant time. In total this takes $O(\tau n^3)$ time. Using a similar simultaneous scan we can add the intervals on which G is maximal to our set of maximal groups found so far.

It follows that we can compute all time intervals on which G is maximal in $O(\tau n^4)$ time. Since we do this for all subsets $G \subseteq \mathcal{X}$ we obtain the following result.

▶ **Theorem 8.** *Given a set $\mathcal{X}$ of n entities in which each entity moves in $\mathbb{R}^d$ along a trajectory of τ edges, we can compute all maximal groups in $O(\tau n^4 2^n)$ time, using $O(\tau n^3)$ space.*

6 A Lower Bound on the Maximum Number of Maximal Groups at some Time t

The result in the previous section shows that, when τ is large but n is small, we can improve the dependence on τ from quadratic to linear. However, we pay for this by having an exponential dependence on n. This naturally raises the question whether an algorithm with linear dependence on τ, but polynomial dependence on n, is possible. While we do not know the answer to this question, we present a construction which may indicate that such a result is hard to obtain, if possible at all.

We show that the number of maximal groups that contain a given time t can be exponential in n, provided that τ is sufficiently large. Without the requirement that the maximal groups must span a single moment in time, it is easy to make a construction of trajectories that has a number of maximal groups that is linear in τ, even with just two entities, so it is unbounded in n. Similarly, we can easily construct trajectories that give rise to $2^n - n - 1$ maximal groups (with a group size of at least $m = 2$) that are different in composition using roughly 2^n time stamps by making these groups consecutive. The construction that we present, where many different maximal groups occur simultaneously, is more involved, and shows that there may be $\Omega(\sqrt{2}^n)$ maximal groups simultaneously when there are $\Omega(\sqrt{2}^n)$ time stamps. While the result does not imply any lower bound for the problem of computing all maximal groups,

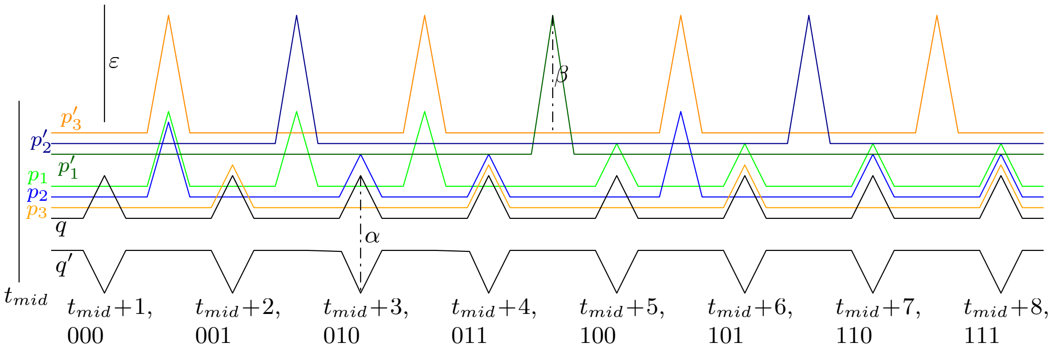


Figure 4 Right half of the lower bound construction for $k = 3$. The times very near the start and end ε-events of q and q' are shown together with the bitstring of the k-max group that ends there. At t_{mid}, all trajectories are in a single point (the trajectories are not shown near t_{mid}).

it suggests that it may be difficult to obtain an algorithm that is linear in τ and polynomial in n. Several natural approaches to the problem (based on, for instance, divide-and-conquer) appear not to work due to this construction and the result on simultaneous maximal groups.

▶ **Theorem 9.** *There exists a set $\mathcal{X}$ of n entities in $\mathbb{R}^1$ whose trajectories are defined by $\Theta(\sqrt{2}^n)$ time stamps, for which the number of maximal groups at some time t is $\Omega(\sqrt{2}^n)$.*

Proof. We use a set of $n = 2k + 2$ entities, denoted $p_1, \dots, p_k$, $p'_1, \dots, p'_k$, and q and q'. We are interested in counting the groups that contain q, q', and for each i, exactly one of p_i and p'_i. We call any such group *k-max* and will show that they are all maximal. A k-max group G is encoded by a length-k bitstring where the i-th bit is 1 if $p_i \in G$ and it is 0 if $p'_i \in G$.

We make a construction with the following properties; the half after t_{mid} is illustrated for $k = 3$ in Figure 4:

1. The trajectory of p_i is the reverse of p'_i, with respect to t_{mid} (that is, mirrored in t_{mid}), and vice versa.
2. A k-max group starts and ends at free ε_q-events of q and q'.
3. A k-max group encoded by bitstring B starts a fraction after time $1 + B$ and ends a fraction before time $t_{mid} + 1 + B$, where B is interpreted as a binary number.
4. There are only $O(1)$ trajectory vertices of each trajectory within one time unit.
5. Each k-max group is maximal.

At t_{mid}, all trajectories pass through a single point to ensure they are continuous when mirroring, and they are pairwise directly ε-connected. It is the moment in time for which $\Omega(\sqrt{2}^n)$ maximal groups exist, as we will show. After t_{mid}, the entities q and q' will have 2^k pairs of ε-events: an end ε-event directly followed by a start ε-event. We call these events ε_q-events. Whether these ε_q-events are free for a k-max group G depends on the time and the bitstring, or equivalently, which entities from $p_1, \dots, p_k$ are in G.

The ε_q-event at a time t is free for a k-max group G if and only if the bitstring corresponding to time t is the same as the bitstring of G. Hence, (assuming that no earlier ε-event ends G) G will end at the time of its bitstring, so a fraction before $t_{mid} + 1 + B$. By symmetry of p_i and p'_i, G will start a fraction after time $1 + B$. In Figure 4, for example, at time $t_{mid} + 4$, the k-max group $\{p'_1, p_2, p_3, q, q'\}$ ends.

The other ε-events of the trajectories are between two consecutive ε_q-events. These ε-events involve the entities of $p'_1, \dots, p'_k$ and the trajectories need three vertices between ε_q-events. Their presence ensures that only one of p_i or p'_i is in a particular k-max group.

Notice that these ε-events will also be the start or end ε-events of maximal groups that are supersets of a k-max group.

Suppose p'_i is an entity that creates a free ε-event α just before a k-max group G containing p_i ends at $t_{mid} + B$. Obviously, p'_i only needs to create such a free ε-event once and it follows this is only necessary if the previous k-max group G' that ends at $t_{mid} + B - 1$ is not containing p_i. However, other k-max groups that will end after G might contain p'_i. Therefore, to prevent this ε-event becomes free in the duration of those k-max groups, we make entities of $p_1, \ldots, p_k$ that are not in G to keep p'_i ε-connected to all other entities. Still, not all of them are needed to prevent α to become free, but only for each entity p'_h where $h < i$, because by the ordering of the bitstrings, k-max groups contain p'_i and those entities might end after α. See Figure 4: before ε-event α, only p_1 prevents p'_2 from creating a free ε-event (but not p_3). Two k-max groups contain p_1,p'_2 and one of p_3 or p'_3 end after α while k-max group of $\{p'_1, p'_2, p_3\}$ ends before α.

▶ **Claim 10.** *If a maximal group containing time t_{mid} contains at least p_i or p'_i for all indices i, and both p_i and p'_i for at least one index i, then its time interval cannot contain both time h and $t_{mid} + h$ for any integer h.*

Proof. Suppose for contradiction G is a maximal group which contains both p_i and p'_i, and its time interval fully contains an interval $[h, t_{mid} + h]$ for some integer h; suppose further that i is the smallest index for which this is the case. Let B be the bitstring that encodes the entities with indices $1 \ldots i - 1$; let $B^- = B0111 \ldots 1$ be obtained from B by appending a single 0 and $k - i$ 1s and let $B^+ = B1000 \ldots 0$ be obtained from B by appending a single 1 and $k - i$ 0s. Then G starts not earlier than some time between B^- and B^+, and ends not later than some time between $t_{mid} + B^-$ and $t_{mid} + B^+$. Refer to Figure 4. Hence, there is no integer h such that both h and $t_{mid} + h$ are contained in the time interval of G. ◀

The claim directly implies that all k-max groups are maximal, because by Property 3 they start and end at some time h and $t_{mid} + h$, but adding any other trajectories will cause both p_i and p'_i to be in the group for some i.

Moreover, the ε-events created by entities of $p'_1, \ldots, p'_k$ are also the end ε-events of the $2^k - 1$ groups that have more entities than a k-max group. Let the maximal group contains all entities end at free ε-event β at time $t_\beta = t_{mid} + 2^{k-1} + T$ $(0 < T < 1)$ created by p'_i. By the simmetry of the construction and the ordering of the bitstrings, two groups of $n - 1$ entities not containing either p'_i or p_i will end at time $t_\beta - 2^{k-2}$ and $t_\beta + 2^{k-2}$, respectively. Then, continuing the same process with the two groups recursively will results on other maximal groups with different entities. Since the start and end ε-events of these groups are always start later or end earlier than k-max groups, then these groups are maximal because their interval will not contain interval of other maximal groups. Clearly, the number of these maximal groups is fewer than k-max groups because their ε-events only occur between two consecutive ε_q events. In Figure 4, p'_1 defines β, the end ε-event for a maximal group containing all entities. Then, maximal group that are not contain p_1 or p'_1 will end before or after β, respectively.

To build the construction, all trajectories must have a constant times 2^k vertices for the ε-events of q and q' and a constant number of vertices in between those ε-events. Each trajectory in the construction has $\Theta(\sqrt{2}^n)$ vertices. We conclude that the number of maximal groups that contain time t_{mid} in this construction is at least $2^k = 2^{n/2-1} = \Omega(\sqrt{2}^n)$. ◀

7 Conclusions and Future Work

In this paper we introduced a variation on the grouping structure definition [2] and argued that it corresponds better to human intuition. The number of maximal groups that can arise in a set of n moving entities is $\Theta(\tau n^3)$ in the worst case. We have given an algorithm for trajectories moving in $\mathbb{R}^1$ that computes all maximal groups and runs in $O(\tau^2 n^4)$ time. In $\mathbb{R}^d$, our algorithm runs in $O(\tau^2 n^5 \log n)$ time. For the more general case where the input trajectories do not have time-aligned vertices, the algorithm for trajectories in $\mathbb{R}^1$ can be extended at the cost of an extra factor of $\alpha(n)$, while the same result still holds for trajectories in $\mathbb{R}^d$.

Furthermore, we presented an algorithm that has only linear dependence in τ, at the expense of exponential dependence in n. Since collections of trajectories are often very large in the number of time stamps and not necessarily in the number of trajectories, this algorithm or a practical variation on it may still be useful. This algorithm is not affected by whether or not the vertices of the trajectories are aligned in time.

The trade-off in the dependence on n and τ gives rise to interesting open problems. Most importantly, is it possible to develop an algorithm whose running time is linear in τ and polynomial in n? Similarly, can we realize subquadratic dependence on τ without having exponential dependence on n? In general, what trade-offs are possible?

Future work includes implementing our algorithms and experimentally showing the differences between the definition of groups in [2] and our refined definition, both qualitatively and quantitatively. It would also be interesting to develop an output-sensitive algorithm that uses considerably less time if the output is small, or under realistic input assumptions. Finally, it would be interesting to investigate whether one can develop algorithms that take geodesic distance into account to define direct ε-connectedness instead of the straight-line distance, as was done for the previous definition of a group [10].

References

1 Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111–125, 2008.

2 Kevin Buchin, Maike Buchin, Marc van Kreveld, Bettina Speckmann, and Frank Staals. Trajectory grouping structure. *Journal of Computational Geometry*, 6(1):75–98, 2015.

3 Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Computational movement analysis. In *Handbook of Geographic Information*, pages 423–438. Springer, 2012.

4 Joachim Gudmundsson and Marc van Kreveld. Computing longest duration flocks in trajectory data. In *Proc. 14th ACM International Symposium on Advances in Geographic Information Systems*, GIS'06, pages 35–42, 2006.

5 Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica*, 11:195–215, 2007.

6 Yan Huang, Cai Chen, and Pinliang Dong. Modeling herds and their evolvements from trajectory data. In *Geographic Information Science*, volume 5266 of *LNCS*, pages 90–105. Springer, 2008.

7 San Hwang, Ying Liu, Jeng Chiu, and Ee Lim. Mining mobile group patterns: A trajectory-based approach. In *Advances in Knowledge Discovery and Data Mining*, volume 3518 of *LNCS*, pages 145–146. Springer, 2005.

8 Hoyoung Jeung, Man Yiu, Xiaofang Zhou, Christian Jensen, and Heng Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1:1068–1080, 2008.

9 Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in Spatial and Temporal Databases*, volume 3633 of *LNCS*, pages 364–381. Springer, 2005.

10 Irina Kostitsyna, Marc van Kreveld, Maarten Löffler, Bettina Speckmann, and Frank Staals. Trajectory grouping structure under geodesic distance. In *Proc. 31st International Symposium on Computational Geometry, SoCG 2015*, pages 674–688, 2015.

11 Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.

12 Salman Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. In *Proc. 28th Annual Symposium on Computational Geometry*, SoCG'12, pages 269–276, 2012.

13 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

14 Arthur van Goethem, Marc van Kreveld, Maarten Löffler, Bettina Speckmann, and Frank Staals. Grouping time-varying data for interactive exploration. In *Proc. 32th Annual Symposium on Computational Geometry*, SoCG'16, pages 61:1–61:16, 2016.

15 Yu Zheng and Xiaofang Zhou. *Computing with Spatial Trajectories*. Springer, 2011.

A Sidetrack-Based Algorithm for Finding the k Shortest Simple Paths in a Directed Graph[*]

Denis Kurz[1] **and Petra Mutzel**[2]

1 **Department of Computer Science, TU Dortmund, Germany**
denis.kurz@tu-dortmund.de

2 **Department of Computer Science, TU Dortmund, Germany**
petra.mutzel@tu-dortmund.de

Abstract

We present an algorithm for the k shortest simple path problem on weighted directed graphs (kSSP) that is based on Eppstein's algorithm for a similar problem in which paths are allowed to contain cycles. In contrast to most other algorithms for kSSP, ours is not based on Yen's algorithm [19] and does not solve replacement path problems. Its worst-case running time is on par with state-of-the-art algorithms for kSSP. Using our algorithm, one may find $\mathcal{O}(m)$ simple paths with a single shortest path tree computation and $\mathcal{O}(n+m)$ additional time per path in well-behaved cases, where n is the number of nodes and m is the number of edges. Our computational results show that on random graphs and large road networks, these well-behaved cases are quite common and our algorithm is faster than existing algorithms by an order of magnitude.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases directed graph, k-best, shortest path, simple path, weighted graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.49

1 Introduction

The k *shortest path problem* in weighted, directed graphs (kSP) asks for a set of k paths from a *source* s to a target *target* t in a graph with n nodes and m edges. Every path that is not output by an algorithm should be at least as long as any path in the output. Algorithms for this problem can be useful tools when it is hard to specify constraints that a solution should satisfy. Instead of computing only one shortest path, kSP algorithms generate k paths, and the user can then pick the one that suits their needs best. The best known algorithm for this problem runs in time $\mathcal{O}(m + n \log n + k \log k)$ and is due to Eppstein [4]. In the initialization phase, the algorithm builds a data structure that contains information about all s-t paths and how they interrelate with each other, in time $\mathcal{O}(m + n \log n)$. This can be reduced to $\mathcal{O}(m + n)$ if the shortest path tree (SP tree) can be computed in time $\mathcal{O}(m + n)$. In the enumeration phase, a *path graph* is constructed. The path graph is a quaternary min-heap where every path starting in the root correlates to an s-t path in the original graph. We require $\mathcal{O}(k \log k)$ time for the enumeration phase if we want the output paths to be ordered by length. If the order in which the paths are output does not matter, Frederickson's heap selection algorithm [7] can be used to enumerate the paths after the initialization phase in time $\mathcal{O}(k)$.

The k *shortest simple path problem* (kSSP), introduced in 1963 by Clarke, Krikorian and Schwartz [2], seems to be more expensive, computationally. In contrast to kSP, the computed

[*] This work was partially supported by the German Research Foundation, RTG 1855.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 49; pp. 49:1–49:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

paths are required to be simple, i.e., they must not contain a cycle. The extra effort may be well-invested if many of the k shortest paths are non-simple and we are only interested in simple paths. The algorithm by Yen [19] used to have the best theoretical worst-case running time of $\mathcal{O}(kn(m + n \log n))$ for quite some time. Gotthilf and Lewenstein [9] improved upon this bound recently. They observed that kSSP can be solved by solving $\mathcal{O}(k)$ all pairs shortest path (APSP) instances. Using the APSP algorithm by Pettie [13], they obtain a new upper bound of $\mathcal{O}(kn(m + n \log \log n))$. Vassilevska Williams and Williams [18] showed that, for constant k, an algorithm for kSSP with running time $\mathcal{O}(n^{3-\varepsilon})$ for some positive ε (*truly subcubic*) would also yield algorithms with truly subcubic running times for some other problems, including APSP. A recent survey of the field is due to Eppstein [5]. The kSSP on undirected graphs seems to be significantly easier. Katoh et al. [11] proposed an algorithm that solves kSSP on undirected graphs in time $\mathcal{O}(k(m + n \log n))$.

A subproblem occurring in Yen's algorithm is the (restricted) *replacement path problem*. Given a shortest s-t path p in a graph, it asks for a set of paths as follows. For each $i < |p|$, the set has to include a shortest simple path that uses the first $i - 1$ edges of p, but not the ith. This problem has to be solved $\mathcal{O}(k)$ times to find the k shortest simple paths using Yen's algorithm. In the original version of Yen's algorithm, the replacement paths are found using $\mathcal{O}(|p|)$ shortest path computations, resulting in time $\mathcal{O}(n(m + n \log n))$. Hershberger et al. [10] compute one SP tree rooted in s and one reversed SP tree rooted in t, respectively. They use these two trees to find a replacement path in constant time per edge on p, cutting down the time required to find all replacement paths to $\mathcal{O}(m + n \log n)$ when Dijkstra's algorithm is used. However, the paths generated this way are not guaranteed to be simple. Such non-simple paths can be detected in constant time and repaired by falling back to Yen's replacement path computation for the path edge in question. Since they do not provide an upper bound for the number of non-simple paths that may occur using this method, the worst-case running time is again $\mathcal{O}(n(m + n \log n))$.

Some approaches reuse one fixed reversed SP tree T_0 rooted in t and computed during the initialization of their kSSP algorithm, in contrast to $\mathcal{O}(1)$ SP trees per replacement path instance. Pascoal [12] noticed that the replacement path that deviates from p at node v might be one that uses an edge (v, w) to an unused successor w of v and then follows the path from w to t in T_0. Therefore, they test whether the shortest such path is simple, and fall back to a full shortest path computation if it is not. Although they do not describe in detail how this check is done, it can be done in time $\mathcal{O}(m + n)$ per replacement path instance by partitioning the nodes into blocks as described by Hershberger et al. [10]. Feng [6] uses the reversed SP tree to partition V into three classes. For each edge (u, v) on p for which we want to compute a replacement path, *red* nodes have already been used to reach v via p. A *yellow* node v is a non-red upstream node of some red node in T_0, i.e., the path from v to t in T_0 contains a red node. All other nodes are *green*. They then do shortest path computations from v using Dijkstra's algorithm like Yen. However, they are able to restrict the search to yellow nodes, resulting in a significantly smaller search space. Feng does not provide upper bounds on the size of this search space, resulting again in a worst-case running time of $\mathcal{O}(n(m + n \log n))$ for each replacement path instance.

The fact that the upper time bound for exact kSSP algorithms has not been improved for a long time inspired research on inexact approaches. This line of research so far spans three publications that all use the notion of a *detour* of a path, which is the (connected) subpath of a replacement path that diverts from a reference path. It was started by Roditty and Zwick [15] who proposed a Monte Carlo $\mathcal{O}(m\sqrt{n} \log n)$ algorithm for the replacement path problem on directed graphs with small integer weights. They also proposed a framework that solves

kSSP by $\mathcal{O}(k)$ computations of second shortest simple paths in appropriate subgraphs, which in turn is solved by their replacement path algorithms. Roditty [14] enhanced this framework to allow for approximate kSSP algorithms when an approximation algorithm for the second shortest path subproblem is used. They also provided such an $\frac{3}{2}$-approximation algorithm, leading to a $\frac{3}{2}$-approximation algorithm for kSSP with running time in $\mathcal{O}(k\sqrt{n}(m + n\log n))$. An α-approximation algorithm guarantees that the i-th output path is at most α times as long as an actual i-th shortest path. The algorithm for approximate second shortest simple paths distinguishes between short and long detours. This approach was extended by Bernstein [1] from two to $\mathcal{O}(\log n)$ classes of detours, which are handled in increasing order. This way, they were able to obtain an algorithm that gives $(1 + \varepsilon)$ approximations in $\mathcal{O}(\varepsilon^{-1}\log^2 n \log(nC/c)(m + n\log n)$ time, where c, C are the minimum and maximum edge cost, respectively, giving the first (approximation) algorithm that breaks the $\mathcal{O}(m\sqrt{n})$ barrier. See Frieder and Roditty [8] for an experimental study of Bernstein's algorithm.

Our contribution. We propose an algorithm that was derived from Eppstein's notion of a path graph [4]. Our algorithm achieves the same worst-case running time as Yen's algorithm. Like Yen, we rely on shortest path (tree) computations. In contrast to Yen-based algorithms, however, our algorithm may draw $\mathcal{O}(m)$ simple paths from one SP tree computation. If the underlying graph is acyclic, the revised algorithm at the end of this paper requires $\mathcal{O}(n\log n + k(m + n))$ without further modifications. Alternatively, one could test whether the graph is acyclic and then use Eppstein's algorithm. However, this method fails if the graph has just a single cycle, in which case our algorithm appears to be a good choice. As most other kSSP algorithms, our algorithm works on multigraphs without modification. After some definitions in Section 2, we propose a simplified version of our algorithm with running time $\mathcal{O}(km(m + n\log n))$ in Section 3. In Section 4, we show how this running time can be reduced to $\mathcal{O}(kn(m + n\log n))$, and how to improve the running time in practice even further. Finally, we present the results of our computational studies in Section 5 to prove the efficiency of our algorithm.

2 Definitions

Let $G = (V, E)$ be a directed graph with *node* set V and *edge* set E. Let $s, t \in V$ be *source* and *target* nodes, respectively. We assume an implicit edge weight function $c : E \to \mathbb{R}_0^+$ throughout this paper. We denote the number of nodes $|V|$ by n and the number of edges $|E|$ by m. A *path* connecting v to w in G, or v-w path, is an edge sequence $p = (e_1, e_2, \ldots, e_r)$, $e_i = (v_i, w_i)$, with $v = v_1$, $w = w_r$ and $w_i = v_{i+1}$ for $1 \leq i < r$. For the sake of simplicity, we only consider combinations of G, s and t such that there exists an s-v path and a v-t path in G for every $v \in V$. A node u is said to be on the path p, denoted by $u \in p$, if $u = w$ or $u = v_i$ for some i. If $v_i \neq v_j \neq w$ for $1 \leq i, j \leq r$, p is a *simple* path. The *prefix* $(e_1, \ldots, e_i)$ is a v-w_i path and denoted by p^i. The *length* $c(p)$ of the path p is the sum of edge weights of its edges. If every v-w path is at least as long as p, it is called a *shortest v-w path*. We write $G - p$ to denote the induced subgraph $G[\{v \in V \mid v \notin p\}]$.

The *k shortest simple path problem* (kSSP) is an enumeration problem. Given a directed graph $G = (V, E)$ with source node $s \in V$, target node $t \in V$, edge weights c, and some $k \in \mathbb{N}$, we want to compute a set P comprising k simple paths from s to t in G such that $c(p) \leq c(p')$ for every pair $p \in P$, $p' \notin P$ of simple paths. We obtain the *k shortest path problem* (kSP) if we do not require the computed paths to be simple.

A *shortest path tree* (SP tree) T of G is a subtree of G with node set V such that each $v \in V$ has exactly one outgoing edge, which lies on a shortest v-t path, or no outgoing edges if no such edge exists. We denote the latter case by $v \notin T$. Our algorithm will compute several SP trees, the first of which we call *initial SP tree* T_0. An edge $e \notin T$ is called *sidetrack* w.r.t. T; we will omit T in most cases. For a sidetrack $e = (v, w)$, the *sidetrack cost* $\delta_T(e)$ is defined as $(c(e) + d(w)) - d(v)$, where $d(u)$ is the length of the unique u-t path in T. The sidetrack cost is therefore the difference between the length of a shortest v-t path and the length of a shortest v-t path that starts with e. The sidetrack set $D_T(v)$ of a node $v \in V$ is the set of all sidetracks w.r.t. T with tails on the unique v-t path in T. When sidetracks are organized in heaps, we use sidetrack costs for comparison.

Let $p = (e_1, \ldots, e_k)$, $p' = (f_1, \ldots, f_l)$ be two s-t paths, and $i^* = \max\{i \mid e_j = f_j \text{ for } 1 \leq j < i\}$. Then, with respect to p, i^* is the *deviation index*, the tail of e_{i^*} is the *deviation node* $\operatorname{dev}(p')$, and e_{i^*} is the *deviation edge* of p'. As is usual for kSSP algorithms, we will discover paths in a hierarchical fashion. We manage a *candidate set*, i.e., a set of *candidate paths* that have been found, but have not been determined to be one of the k shortest simple paths. Any path p that is *extracted* from the candidate set will be part of the solution; candidate paths that are found not to be part of the solution are *discarded*. A range of new candidate paths is derived from p. Any derived path p' is added to the candidate set. We call p the *parent path* of p'. When p is omitted, the terms deviation node and edge are w.r.t. the parent path of p'. By removing the deviation edge of p from p, p is split into its *prefix path* $\operatorname{pref}(p) := p^{i^*}$ starting in s, and its *suffix path* $\operatorname{suff}(p)$ ending in t. The initial s-t path p_0 in T_0 has no parent path and thus no deviation edge. We define its suffix path to be p_0 itself.

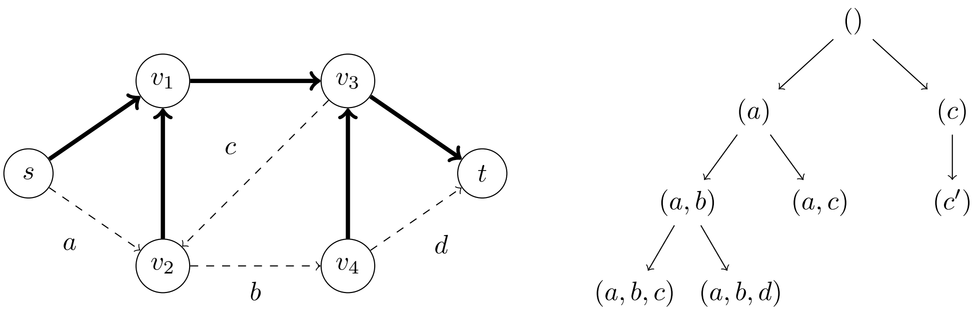
We generalize Eppstein's representation [4] for paths. Eppstein represents paths as sequences of sidetracks, all w.r.t. the same SP tree. In our representation, every sidetrack e in a sidetrack sequence may be associated with a different SP tree T_e. The path represented by a sidetrack sequence $(e_1, \ldots, e_r)$ can then be reconstructed as follows. Starting in s, we follow the initial SP tree T_0 until we reach the tail of e_1. After reaching the tail of e_i, we traverse e_i and follow T_{e_i} until we reach the tail of e_{i+1}, or, in case $i = r$, until we reach t. Note that Eppstein's representation is the special case where $T_e = T_0$ for each e in a sidetrack sequence, and that both Eppstein's sidetrack sequences and our generalized ones may represent non-simple paths. The distance from a node v to t in a SP tree T_e associated with a sidetrack e is denoted by $d_e(v)$.

3 Basic Algorithm

In this section, we propose a simplified way to enumerate the k shortest simple paths. We describe in Section 4 some modifications to achieve our proclaimed running time guarantee.

We initialize an empty priority queue Q that is going to manage candidate paths. The key of a path in Q is its length. We compute the initial SP tree T_0 and push its unique s-t path, represented by an empty sidetrack sequence, to Q. We now process the paths in Q in order of increasing length until we found k simple paths. Let $(e_1, \ldots, e_r)$ be a sidetrack sequence extracted from Q, and p the path that is represented by this sequence. Although the first path that is pushed to Q is always simple, we will eventually push non-simple paths to Q, too. Therefore, we first have to determine whether p is simple in a *pivot step*. This check can be done by simply walking p and marking every visited node.

We first describe how to handle the simple case. We start by outputting p. Let u be the head of e_r, and $T = T_{e_r}$. For every sidetrack $e = (v, w)$ with $v \in \operatorname{suff}(p)$, we discover a new path p' represented by the sequence $(e_1, \ldots, e_r, e)$. We set $\operatorname{dev}(p') = v$, and push p' to Q. By choosing $T_e = T$, we simply reuse the SP tree that is also associated with the last sidetrack in the sequence representing p. The length of p' can easily be computed as $c(p) + \delta_T(e)$. If



(a) Example graph

(b) Sidetrack sequences

Figure 1 Example for the basic algorithm. In Figure 1a, the thick, solid edges belong to T_0. In Figure 1b, every sidetrack is associated with T_0 except for c', which is associated with the SP tree T_1 comprising the edges b and d. An arrow from sequence p to sequence p' indicates that p is the parent path of p'.

$d_{e_r}(w)$ is undefined because T does not contain a w-t path, we simply ignore e. Apart from these dead ends, we add one path for each sidetrack in $D_T(u)$ to Q. Note that sidetracks emanating from t can safely be ignored.

Consider the example in Figure 1. The sidetrack sequence (a) with $T_a = T_0$ represents a simple path p that passes the nodes s, v_2, v_1, v_3, t in this order. The suffix of this path is its v_2-t subpath, and the sidetracks b, c have tails on this suffix. Therefore, when (a) is extracted from Q, p is output and the sequences (a, b) and (a, c) with $T_a = T_b = T_c$ are pushed to Q.

Now assume we extracted a non-simple path p represented by the sidetrack sequence $(e_1, \ldots, e_r)$. We try to extend the concatenation of $\text{pref}(p)$ and e_r to a simple s-t path. Let $e_r = (v, w)$. Any valid extension avoids the nodes of $\text{pref}(p)$ after v. We are only interested in shortest extensions. Therefore, we compute a new SP tree T and distances d, but in $G - \text{pref}(p)$ instead of G to make sure that nodes of the prefix path of p are not used again. If $w \notin T$, $\text{pref}(p)$ cannot be extended to a simple s-t path, and we discard p. Otherwise, we push the sequence $(e_1, \ldots, e_r)$ to Q again. In this new sequence, we associate T with e_r instead of T_{e_r} from the old sequence. The sequence represents a path p' obtained by concatenating the simple prefix path of p, the edge e_r, and the w-t path in T that, by construction, avoids all nodes of $\text{pref}(p)$. The suffix itself is simple because it is a shortest path in a subgraph of G. Hence, p' is simple. The length of p' is $c(\text{pref}(p)) + c(e_r) + d(w)$.

Consider again the example in Figure 1. The sidetrack sequence (a, c) with $T_a = T_c = T_0$ represents a non-simple path p that visits the nodes s, v_2, v_1, v_3, v_2, v_1, v_3, t in this order. The deviation node of p is v_3, its deviation edge c, and its prefix path is $(a, (v_2, v_1), (v_1, v_3))$. We compute a new SP tree T in $G - \text{pref}(p)$, which only consists of the edge d. Therefore, T does not contain a v_2-t path, and p is discarded. In contrast, assume the sequence (c) with $T_c = T_0$ was just extracted from Q. It represents almost the same path as the sequence above, but it skips the first visit of v_2. Again, v_3 is the deviation node and c the deviation edge. The prefix path comprises the nodes s, v_1 and v_3. After removing them temporarily, a new SP tree T_1 is computed, consisting only of the edges b and d. The sequence (c) with $T_c = T_1$ is pushed to Q. This new sequence represents the simple path $((s, v_1), (v_1, v_3), c, b, d)$, where c is the last sidetrack in the extracted sequence, and (b, d) is the unique v_2-t path in T_1.

Finally, when (c) with $T_c = T_1$ is extracted, the represented path is output. The sidetracks emanating from its prefix are (v_2, v_1) and (v_4, v_3). Since $v_1, v_3 \notin T_1$, these sidetracks are ignored and no new path is pushed to Q.

▶ **Lemma 1.** *The above algorithm computes the k shortest simple s-t paths of a weighted, directed graph $G = (V, E)$.*

Proof. The algorithm uses the same idea of shortest deviations as existing kSSP algorithms or Eppstein's kSP algorithm. We only have to show that a non-simple path p is processed before its simple enhancement p', resulting from the suffix repair in the non-simple case, is actually needed. The set of nodes that are forbidden when the SP tree for p is computed is a proper subset of the node set that the SP tree for p' may not use. The suffix of p is therefore not longer than that of p', and p is extracted from Q (and subsequently, p' is pushed) before we need to extract p'. ◀

This basic form of our algorithm requires too many computations of SP trees:

▶ **Lemma 2.** *The running time of the above algorithm is $\mathcal{O}(km(m + n \log n))$.*

Proof. While processing a non-simple path, at most one new path is pushed to Q, which is always simple. Thus, the parent of a non-simple path is always simple. We have to process at most k simple paths, each of which requires $\mathcal{O}(m + n)$ time. Every simple path may have $\mathcal{O}(m)$ sidetracks extensions. In the worst case, all of them represent non-simple paths, yielding $\mathcal{O}(km)$ SP tree computations with a total running time of $\mathcal{O}(km(m+n \log n))$. The running time for the non-simple cases clearly dominates. For every subset of E, there is at most one permutation of this subset that represents a simple s-t path. The maximum number of paths enumerated by the algorithm is therefore $k' := \min\{k, 2^m\}$. We can limit the size of Q efficiently to k' using a double-ended priority queue [16]. We push $\mathcal{O}(k'm)$ paths to Q and extract $\mathcal{O}(k'm)$ paths from it; both operations require $\mathcal{O}(\log k')$ time on interval heaps. The total time spent on processing Q is $\mathcal{O}(k'm \log k') \subset \mathcal{O}(km^2)$. The pivot step requires $\mathcal{O}(n)$ time for each of the $\mathcal{O}(k'm)$ extracted paths. ◀

Finally, we turn our attention to the space requirements of the above algorithm. We need $\mathcal{O}(n)$ space for each SP tree that we compute. Since SP trees are never discarded and we compute one for each non-simple extracted path, the total space for all SP trees is $\mathcal{O}(kmn)$. For each simple extracted path p, we push a path to Q for each edge that has its tail on p. These new paths are represented by an edge and a pointer to some SP tree, and therefore require constant space. We extract up to k simple paths with $\mathcal{O}(m)$ sidetracks each, and therefore require $\mathcal{O}(km)$ space for Q itself.

4 Improvements

We show how the number of SP tree computations can be reduced to $\mathcal{O}(kn)$ in the worst case. Further, the space requirements are reduced by a factor of n.

So far, we were only able to bound the number of SP tree computations by $\mathcal{O}(m)$ for each extracted simple path. This stems from the fact that there can be $\mathcal{O}(m)$ sidetracks with tails on such a path, each of them requiring a subsequent SP tree computation in the worst case. Consider two sidetrack sequences $(e_1, \ldots, e_r, f_1 = (u, v))$, $(e_1, \ldots, e_r, f_2 = (u, w))$ that were added when a path p represented by $(e_1, \ldots, e_r)$ was processed. Let p_1, p_2 be the paths represented by these sequences, respectively. Assume that both sequences represent non-simple paths, and therefore both require a new SP tree. We assume w.l.o.g. that p_1 is extracted from Q before p_2. When p_1 is extracted from Q, we discover that it contains a cycle. We then have to compute an SP tree T for the graph $G - p'$, where p' is the shortest s-u subpath of p. We push $(e_1, \ldots, e_r, f_1)$ back to Q, and update $T_{f_1} = T$. When p_2 is extracted, the basic algorithm computes an SP tree for the exact same graph. This computation can therefore be skipped. We check if an SP tree for this graph has already been computed, and reuse it if it exists. In our case, we simply push $(e_1, \ldots, e_r, f_2)$ with $T_{f_2} = T$ to Q. We obtain the following result.

▶ **Lemma 3.** *Excluding the time spent on Q, the algorithm proposed in Section 3 in conjunction with SP tree reuse requires $\mathcal{O}(kn(m + n \log n))$ time to process non-simple paths.*

Proof. There are still $\mathcal{O}(km)$ many sequences in Q that represent non-simple paths, but only $\mathcal{O}(kn)$ of them trigger an SP tree computation. Let p be a non-simple path extracted from Q. The initial pivot step requires time $\mathcal{O}(n)$. We store in Q along with each path a pointer to its parent path, as well as a pointer to the SP tree for $G - p'$ for every prefix path p'. We can then check if an SP tree for some prefix path has already been computed, and access it if it has, both in constant time. ◀

The total running time of $\mathcal{O}(km^2)$ spent on Q is no longer dominated. Instead of using a priority queue for the candidate paths, we organize all computed paths in a min-heap in the following way. The shortest path is the root of the min-heap. Whenever a path p' is computed while a path p is processed, we insert p' into the min-heap as a child of p. Figure 1b shows an example of such a min-heap. We use Frederickson's heap selection algorithm [7] to extract the km smallest elements from this heap. The heap described above has maximum degree m, again yielding a running time of $\mathcal{O}(km^2)$. Let P_p be the set of paths found during the processing of p. Instead of inserting every $p' \in P_p$ as a heap child of p, we heapify P_p to obtain the heap H_p, using the lengths of the paths for keys again. The root of H_p is then inserted into the global min-heap as a child of p. Note that the parent path of every path in H_p is not its heap parent in H_p, but still p itself. Every simple path p in the min-heap now has at most two heap successors with the same parent path as p, and at most one heap successor whose parent is p itself. Every non-simple path has at most one simple path as heap processor. The maximum degree of the global min-heap is therefore bounded by three and Frederickson's heap selection can be done in time $\mathcal{O}(km)$.

▶ **Corollary 4.** *The algorithm proposed in Section 3 in conjunction with SP tree reuse and Frederickson's heap selection algorithm computes the k shortest simple s-t paths of a weighted, directed graph $G = (V, E)$, $s, t \in V$, in $\mathcal{O}(kn(m + n \log n))$ time.*

The first improvement above reduced the space required by the basic algorithm from $\mathcal{O}(kmn)$ to $\mathcal{O}(kn^2)$. We are not able to reduce the number of SP tree computations to $o(kn)$. However, it is not necessary to permanently store all these SP trees at the same time. Only up to k of them can contain a simple path that eventually gets extracted from Q. We propose to store the computed SP trees in a max priority queue $\mathcal{S}$. The priority of an SP tree T in $\mathcal{S}$ is $\max\{c(p') + c(e) + d_T(w) \mid e = (v, e) \in E\}$, where $p' = (e_1, \ldots, e_r, f)$ is the path T was computed for, and $f = (u, v)$ for some $u \in V$. Whenever $|\mathcal{S}|$ exceeds k, $\mathcal{S}$ contains an SP tree that will not contribute to the k shortest simple paths. This is always the SP tree with the highest priority. It can be extracted from $\mathcal{S}$ in $\mathcal{O}(\log k)$ time and therefore does not have an impact on the worst-case running time. The space that was used to store the extracted tree can later be used to store new SP trees. The number of SP trees stored at any point in time never exceeds $k + 1$. The total space requirements are then dominated by the D_T's, and bounded by $\mathcal{O}(km)$.

Our final improvement does not change the worst-case running time or space consumption. Instead, we will speed up an important part of the algorithm by a factor of n by using a rather cheap test. Consider one of the k simple s-t paths p represented by sidetracks $(e_1, \ldots, e_r)$ with $e_i = (v_{i-1}, v_i)$, $s = v_0$ and $t = v_r$. When p is processed, we push the set P_p of paths to Q, with $|P_p| \in \mathcal{O}(m)$. The basic algorithm tests for each $p' \in P_p$ if p' is simple in time $\mathcal{O}(n)$, leading to a total time of $\mathcal{O}(kmn)$ for these tests. Let $T = T_{e_r}$. By removing all e_i from T, the SP tree decomposes into a set of trees T_i such that T_i is rooted in v_i. The *block* i is the node set of T_i. Observe that the path p' represented by a sequence $(e_1, \dots, e_r, e)$, $e = (v_i, w)$, with v_i, w in block i, j, respectively, is simple iff $i < j$. If $i \geq j$, we follow p until we reach v_i, traverse e and follow T to reach v_i again. Otherwise, the first node on p we hit after deviating from it via e is v_j. Since $i < j$, the v_j-t subpath of p does not contain v_i, so p' is simple. The partition of V into blocks is $\mathcal{O}(n)$-time computable. We can then collect all sidetracks deviating from p and check for each of them if their heads belong to a smaller block than their tails in $\mathcal{O}(m)$ total time. We store this information along with the corresponding sidetrack sequences in Q. The pivot turn is replaced by a constant time lookup. All tests for simplicity then require time $\mathcal{O}(k(m+n))$ instead of $\mathcal{O}(kmn)$.

5 Experiments

To demonstrate the effectiveness of our algorithm, we conducted a series of experiments. Feng [6] showed recently that their algorithm is the most efficient one in practice. We therefore compare our sidetrack-based algorithm to Feng's node classification algorithm. For reference, we also include results for the most promising third contender, an algorithm proposed by Sedeño-Noda [17]. We used all graph classes that Feng had used in their experiments, including road graphs that are especially relevant in practice.

Sedeño-Noda kindly provided us with the implementation KCM of their algorithm. We conducted our experiments on a desktop computer very similar to that of Feng. On our computer, KCM was consistently slower than what is reported for KCM on Feng's computer [6]. On average, we required 10.4 seconds on NY and 15.94 seconds on BAY (described in Section 5.2) using KCM; Feng reported 8.81 and 11.23 seconds, respectively. In contrast, our implementation NC of Feng's algorithm (*without* express edges) consistently gives lower running times than those reported by Feng. We also implemented Feng's algorithm *with* express edges, which was always slower than NC. Note that Feng did not specify whether express edges were used in their experiments. All improvements proposed in Section 4 were used in the implementation SB of our sidetrack-based algorithm with the following exception: Frederickson's heap selection algorithm was used neither for NC nor for SB. This results in an additional running time of $\mathcal{O}(km \log k)$ for SB, but not for NC.

Shortest paths (NC) and SP trees (SB) are computed using a common implementation of Dijkstra's algorithm; tentative labels are managed by a pairing heap. Our implementation of Dijkstra's algorithm stops as soon as the label of the target node is made permanent if only a single pair shortest path is needed, which is essential for NC. SP trees are computed lazily. A tree is initialized without any edges, and the source node is pushed to a priority queue that is permanently associated to the tree. Whenever a part of the tree is queried (i.e. the distance or predecessor of some node) that has not yet been computed, we simulate the Dijkstra algorithm using the associated priority queue until the queried part is settled. The queue of candidate paths Q is implemented as an interval heap, a form of double-headed priority queues, which allows us to limit its size efficiently to the number of simple paths that have yet to be output. For SB, we use separate priority queues Q_s and Q_n for simple and non-simple paths, respectively. Whenever a path has to be extracted, SB extracts from Q_n iff the shortest path in Q_n is cheaper than the shortest path in Q_s.

We implemented NC and SB in C++, using forward and reverse star representation for directed graphs. The experiments ran on an Intel Core i7-3770 @ 3.40GHz with 16GB of

RAM on a GNU/Gentoo Linux with kernel version `4.4.6` and TurboBoost turned off. Source code was compiled using the GNU C++ compiler `g++-4.9.3` and `-O3` optimization.

5.1 Random Graphs

We first considered random graphs generated by the `sprand` generator provided on the website of the Ninth DIMACS Implementation Challenge [3]. The generator draws at random a fixed amount of edges, possibly resulting in a multigraph. For each combination of graph size $n \in \{2000, 4000, 6000, 8000, 10\,000\}$ and *linear density* $m/n \in \{2, 3, 4, 7, 10, 20, 30, 40, 50\}$, we generated 20 random graphs, and enumerated $k \in \{200, 500, 1000, 2000\}$ simple paths. Edge weights were selected uniformly from $\{1, \ldots, 10\,000\}$. In Table 1, the median and 90% quantile $Q_{.9}$ of execution times for some densities and $k = 2000$ are summarized. Our results confirm Feng's claim that NC is usually faster than KCM on random graphs. NC seems to struggle with very low densities of $m = 2n$ and gets faster for graphs with densities up to about $m = 30n$. On the other hand, KCM and SB display a more consistent growth. SB is always the fastest of the three, with speedup factors ranging from 8 to 15 for lower densities, and 7.5 to 25 for higher densities when compared to the second-fastest algorithm.

We now consider the dispersion of the three algorithms. For SB, 90% of the instances finish within 160% of the corresponding median running times (the fastest 50%) for most combinations of n and m. For KCM, this ratio stays below 106% for all but two combinations of n and m. We could not find any correlation between n, m and k on the one side, and the dispersion of running times on the other side, for KCM and SB. In contrast, NC regularly requires more than thrice the median running time to answer 90% of the queries. Running times are therefore much harder to predict when using NC instead of SB or KCM.

Table 2 shows the median number of Dijkstra calls. The numbers are relatively stable across the various densities, but the Dijkstra counts for the SB algorithms is orders of magnitudes smaller than the count for the NC algorithm. Note, however, that SB needs to compute the complete SP tree every time. In contrast, NC only solves single pair shortest path problems on rather small subgraphs. We also provide the number of polls, i.e., the total number of nodes that were extracted from Dijkstra's priority queue, for comparability. The ratio of the number of polls of NC and SB ranges from 4.6 to 50, and suggests that saving SP tree computations is much more beneficial than reducing the number of nodes visited to answer single-pair shortest path queries.

Finally, the number of SP tree computations actually declines as n grows. Recall that, in the worst case, we have to compute one SP tree for each node of each output simple path. Table 2 shows results for $k = 2000$ and $n \geq 2000$. Nevertheless, the median number of SP tree computations does not exceed 65. Most simple paths therefore correspond to those well-behaved cases where paths represented by sidetracks in already computed SP tree areas are themselves simple most of the time.

5.2 Road Graphs

We consider road graphs of various areas in the USA called TIGER graphs, again provided by the DIMACS website [3]. In particular, we use the road networks of New York (NY), the San Francisco Bay Area (BAY), Colorado (COL), and Florida (FLA). For each of the four areas, we drew 20 s-t pairs at random and enumerated $k \in \{100, 200, 300\}$ paths. The resulting running times are summarized in Table 3, along with the median number of polls. With respect to the median running times, KCM is clearly dominated by NC on any input class, which in turn is dominated by SB. SB achieves a minimum speedup of around 8 on

Table 1 Median and 90% quantile $Q_{.9}$ of running times in seconds for random graphs, $k = 2000$.

		$m = 2n$		$m = 4n$		$m = 10n$		$m = 30n$		$m = 50n$	
n		Med.	$Q_{.9}$	Med.	$Q_{.9}$	Med.	$Q_{.9}$	Med.	$Q_{.9}$	Med.	$Q_{.9}$
2000	KCM	0.67	0.69	0.85	0.88	1.29	1.32	3.58	4.04	9.77	14.11
	NC	0.99	2.51	0.48	1.25	0.39	1.25	0.47	1.50	2.15	3.84
	SB	**0.09**	**0.13**	**0.08**	**0.10**	**0.10**	**0.15**	**0.17**	**0.20**	**0.23**	**0.31**
4000	KCM	1.39	1.46	1.79	1.87	2.96	3.06	15.64	16.03	32.88	33.17
	NC	1.01	2.88	0.84	2.71	0.82	1.97	1.33	5.06	2.07	5.49
	SB	**0.11**	**0.12**	**0.09**	**0.13**	**0.12**	**0.19**	**0.19**	**0.22**	**0.26**	**0.36**
6000	KCM	2.19	2.25	2.86	2.90	5.50	5.78	30.13	30.61	53.33	54.51
	NC	3.28	6.44	0.53	1.67	0.71	3.36	2.05	7.92	2.22	9.07
	SB	**0.13**	**0.20**	**0.11**	**0.13**	**0.13**	**0.21**	**0.22**	**0.32**	**0.30**	**0.45**
8000	KCM	3.04	3.06	4.08	4.14	12.37	12.60	43.36	45.31	73.11	75.00
	NC	1.79	7.84	0.68	2.66	1.92	4.60	3.49	9.67	2.55	9.66
	SB	**0.12**	**0.28**	**0.10**	**0.14**	**0.16**	**0.21**	**0.24**	**0.34**	**0.32**	**0.38**
10 000	KCM	3.96	3.98	5.48	5.53	15.71	15.84	55.95	57.76	92.81	94.79
	NC	1.86	11.91	1.17	5.44	2.61	9.02	6.31	13.56	9.68	26.23
	SB	**0.13**	**0.18**	**0.14**	**0.24**	**0.17**	**0.21**	**0.25**	**0.30**	**0.36**	**0.48**

Table 2 Median number of Dijkstra calls and polls (in millions) of NC and SB for random graphs, $k = 2000$.

		$m = 4n$		$m = 10n$		$m = 30n$		$m = 50n$	
n		Dijkstras	Polls	Dijkstras	Polls	Dijkstras	Polls	Dijkstras	Polls
2000	NC	16 272	1.08	14 533	0.60	13 939	0.43	14 510	2.09
	SB	46	**0.09**	65	**0.13**	38	**0.08**	44	**0.09**
4000	NC	17 292	1.93	14 581	1.45	15 805	1.20	15 605	1.39
	SB	25	**0.10**	20	**0.08**	21	**0.08**	29	**0.11**
6000	NC	17 499	0.86	16 652	0.70	16 544	1.52	16 444	1.13
	SB	23	**0.14**	19	**0.11**	24	**0.14**	21	**0.13**
8000	NC	18 300	1.01	17 316	2.40	17 127	2.72	17 034	1.09
	SB	16	**0.12**	17	**0.13**	17	**0.14**	17	**0.14**
10 000	NC	19 074	1.94	17 824	3.53	18 125	5.07	18 187	6.11
	SB	16	**0.15**	15	**0.15**	10	**0.10**	14	**0.13**

Table 3 Median and 90% quantile $Q_{.9}$ of running times in seconds, and median number of polls for large TIGER road graphs. KCM does not provide the number of polls and was not able to compute the 300 shortest simple paths on FLA.

		$k = 100$			$k = 200$			$k = 300$		
Area		Med.	$Q_{.9}$	Polls	Med.	$Q_{.9}$	Polls	Med.	$Q_{.9}$	Polls
NY	KCM	9.72	11.57	-	19.54	23.48	-	29.76	35.55	-
	NC	2.09	12.38	3.90	3.80	24.30	6.91	5.43	36.18	9.76
	SB	**0.18**	**1.57**	**0.53**	**0.26**	**3.92**	**0.69**	**0.43**	**5.99**	**1.06**
BAY	KCM	12.72	25.90	-	25.16	53.00	-	38.17	83.36	-
	NC	5.32	17.88	15.21	9.49	34.57	28.77	14.14	51.11	38.72
	SB	**0.30**	**4.28**	**0.96**	**0.55**	**9.67**	**1.58**	**0.71**	**14.17**	**1.90**
COL	KCM	17.13	32.99	-	34.56	71.66	-	56.77	117.12	-
	NC	6.83	27.71	16.65	12.04	49.23	30.30	16.73	65.92	44.23
	SB	**0.17**	**11.80**	**0.44**	**0.22**	**17.43**	**0.44**	**0.29**	**26.64**	**0.44**
FLA	KCM	48.51	99.06	-	95.69	215.48	-	-	-	-
	NC	29.86	70.10	54.21	58.07	132.73	106.02	85.70	193.30	157.40
	SB	**0.47**	**4.31**	**1.07**	**0.58**	**20.84**	**1.07**	**0.71**	**26.99**	**1.07**

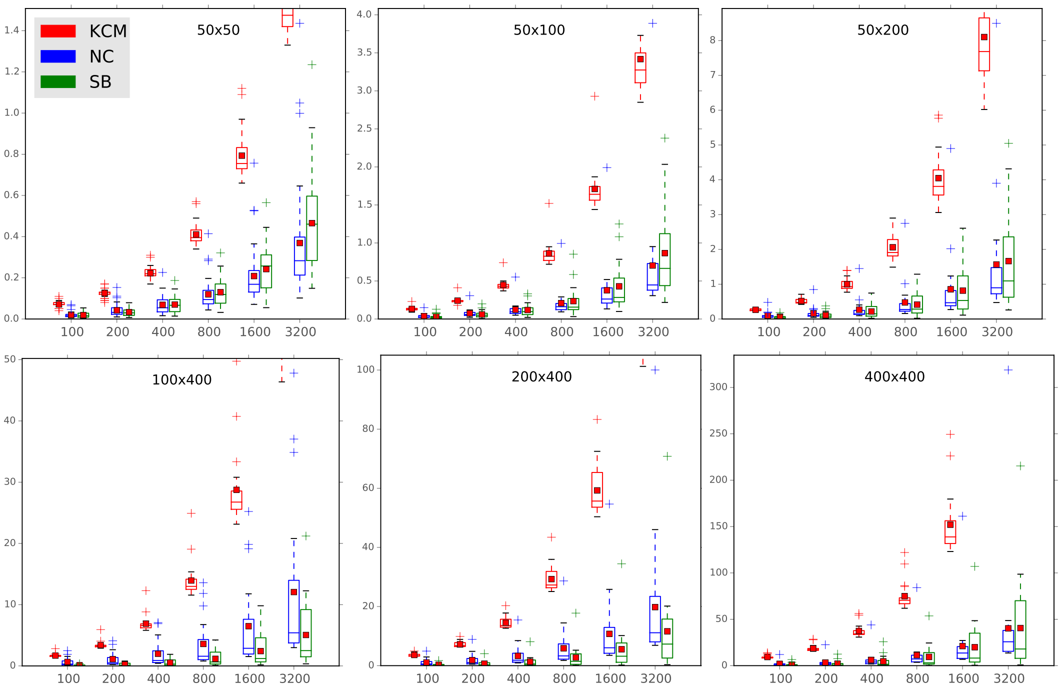


Figure 2 Boxplots of running times in seconds for grid graphs. Plus signs represent outliers. A red square marks the mean. The x-axis corresponds to different values of k and uses a logarithmic scale.

NY for $k = 100$ in comparison to NC. In 90% of the input classes, however, SB achieves a speedup factor of more than 62, and even peaks at a speedup factor of 120. Running times of KCM and SB are much more dispersed than in the random graph case. Still, SB answers 90% of the queries in about the same time the 50% fastest query times of NC, and sometimes much faster. KCM was not able to finish all computations on all inputs.

5.3 Grid Graphs

We repeated Feng's experiments on grid graphs generated by the `spgrid` generator provided on the DIMACS website [3]. The grids have sidelengths $l, w \in \{50, 100, 200, 400\}$ with $l \leq w$, resulting in 10 different grids. For each grid, we generated 20 weight functions by selecting uniformly from $\{1, \ldots, 10\,000\}$ for each edge, and then enumerated $k \in \{100, 200, 400, 800, 1600, 3200\}$ paths. The results of our experiments on grid graphs are summarized in Figure 2. KCM is again the slowest algorithm, but NC is not slower than SB any more. Although NC and SB differ on some classes, there does not seem to be any correlation to the shape or size of the grid. For example, NC is slightly faster on 50×100 grids, but slower on 200×400 grids although these two configurations share the same shape, characterized by an aspect ratio of 2. On the other hand, NC loses its advantage over SB when the grid grows from 50×50 to 50×200 (upper row of plots), but SB loses its advantage over NC as it grows from 100×400 to 400×400 (lower row). In summary, none of the two algorithms is clearly better than the other on grid graphs.

The algorithm proposed in this paper is not slower on any of the considered graph classes, and even faster than state-of-the-art algorithms on random and TIGER road graphs by an order of magnitude. Our algorithm shines on graphs where the length of shortest paths is small in relation to the graph size because sidetrack heaps tend to be smaller.

References

1 Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *SODA 2010*, pages 742–755. SIAM, 2010.

2 S. Clarke, A. Krikorian, and J. Rausen. Computing the N best loopless paths in a network. *J. SIAM*, 11(4):1096–1102, 1963.

3 The Ninth DIMACS Implementation Challenge: 2005-2006. `http://www.dis.uniroma1.it/challenge9/`. Accessed: 2015-11-12.

4 David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

5 David Eppstein. K-best enumeration. arXiv: 1412.5075v1 [cs.DS], 2014.

6 Gang Feng. Finding k shortest simple paths in directed graphs: A node classification algorithm. *Networks*, 64(1):6–17, 2014.

7 Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Inf. Comput.*, 104(2):197–214, 1993.

8 Asaf Frieder and Liam Roditty. An experimental study on approximating k shortest simple paths. *ACM J. Exp. Algorithmics*, 19(1), 2014.

9 Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inf. Process. Lett.*, 109(7):352–355, 2009.

10 John Hershberger, Matthew Maxel, and Subhash Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Trans. Algorithms*, 3(4), 2007.

11 Naoki Katoh, Toshihide Ibaraki, and H. Mine. An efficient algorithm for K shortest simple paths. *Networks*, 12(4):411–427, 1982.

12 Marta M. B. Pascoal. Implementations and empirical comparison of K shortest loopless path algorithms, 2006. 9th DIMACS Implementation Challenge Workshop: Shortest Paths.

13 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

14 Liam Roditty. On the k shortest simple paths problem in weighted directed graphs. *SIAM J. Comput.*, 39(6):2363–2376, 2010.

15 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33, 2012.

16 Sartaj Sahni. *Data Structures, Algorithms, and Applications in C++*. McGraw-Hill Pub. Co., 1st edition, 1999.

17 Antonio Sedeño-Noda. An efficient time and space K point-to-point shortest simple paths algorithm. *Appl. Math. and Comput.*, 218(20):10244–10257, 2012.

18 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS 2010*, pages 645–654, 2010.

19 Jin Y. Yen. Finding the k shortest loopless paths in a network. *Networks*, 17(11):712–716, 1971.

On the Complexity of Matching Cut in Graphs of Fixed Diameter

Hoang-Oanh Le[1] **and Van Bang Le**[2]

1 **Berlin, Germany**
2 **Universität Rostock, Institut für Informatik, Rostock, Germany**
van-bang.le@uni-rostock.de

Abstract

In a graph, a matching cut is an edge cut that is a matching. Matching Cut is the problem of deciding whether or not a given graph has a matching cut, which is known to be NP-complete even when restricted to bipartite graphs. It has been proved that Matching Cut is polynomially solvable for graphs of diameter two. In this paper, we show that, for any fixed integer $d \geq 4$, Matching Cut is NP-complete in the class of graphs of diameter d. This almost resolves an open problem posed by Borowiecki and Jesse-Józefczyk in [Matching cutsets in graphs of diameter 2, Theoretical Computer Science 407 (2008) 574–582].

We then show that, for any fixed integer $d \geq 5$, Matching Cut is NP-complete even when restricted to the class of bipartite graphs of diameter d. Complementing the hardness results, we show that Matching Cut is in polynomial-time solvable in the class of bipartite graphs of diameter at most three, and point out a new and simple polynomial-time algorithm solving Matching Cut in graphs of diameter 2.

1998 ACM Subject Classification F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems, G.2.2 [Discrete Mathematics] Graph Theory

Keywords and phrases matching cut, NP-hardness, graph algorithm, computational complexity, decomposable graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.50

1 Introduction

In a graph $G = (V, E)$, a *cut* is a partition $V = X \dot{\cup} Y$ of the vertex set into disjoint, nonempty sets X and Y, written (X, Y). The set of all edges in G having an endvertex in X and the other endvertex in Y, also written (X, Y), is called the *edge cut* of the cut (X, Y). A *matching cut* is an edge cut that is a (possibly empty) matching. Note that, by our definition, a matching whose removal disconnects the graph need not be a matching cut.

Another way to define matching cuts is as follows ([13, 7]). A partition $V = X \dot{\cup} Y$ of the vertex set of the graph $G = (V, E)$ into disjoint, nonempty sets X and Y, is a matching cut if and only if each vertex in X has at most one neighbor in Y and each vertex in Y has at most one neighbor in X.

Graham [13] studied matching cuts in graphs in connection to a number theory problem called cube-numbering. In [12], Farley and Proskurowski studied matching cuts in the context of network applications. Patrignani and Pizzonia [17] pointed out an application of matching cuts in graph drawing. Matching cuts have been used by Araújo et al. [1] in studying good edge-labellings in the context of WDM (Wavelength Division Multiplexing) networks.

Not every graph has a matching cut; the Matching Cut problem is the problem of deciding whether or not a given graph has a matching cut:

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 50; pp. 50:1–50:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Matching Cut	
Instance:	A graph $G = (V, E)$.
Question:	Does G have a matching cut?

This paper considers the computational complexity of the Matching Cut problem in graphs of fixed diameter.

Previous results. Graphs admitting a matching cut were first discussed by Graham in [13] under the name *decomposable graphs*. The first complexity and algorithmic results for Matching Cut have been obtained by Chvátal, who proved in [7] that Matching Cut is NP-complete, even when restricted to graphs of maximum degree four and polynomially solvable for graphs of maximum degree at most three. These results triggered a lot of research on the computational complexity of Matching Cut in graphs with additional structural assumptions; see [5, 6, 14, 15, 16, 17]. In particular, the NP-hardness of Matching Cut has been further strengthened for planar graphs of maximum degree four ([5]) and bipartite graphs of maximum degree four ([15]).

On the positive side, among others, an important polynomially solvable case has been established by Borowiecki and Jesse-Józefczyk, who proved in [6] that Matching Cut is polynomially solvable for graphs of diameter 2. They also posed the problem of determining the largest integer d such that Matching Cut is solvable in polynomial time for graphs of diameter d.

Our contributions. We prove that Matching Cut is NP-complete, even when restricted to graphs of diameter d, for any fixed integer $d \geq 4$. Thus, unless NP = P, Matching Cut cannot be solved in polynomial time for graphs of diameter d, for any fixed $d \geq 4$. This mostly resolves the open problem posed by Borowiecki and Jesse-Józefczyk mentioned above. Actually, we show a little more: Matching Cut is NP-complete in graphs of diameter 4 and remains NP-complete in bipartite graphs of fixed diameter $d \geq 5$. Complementing our hardness results, we show that Matching Cut can be solved in polynomial time in bipartite graphs of diameter at most 3. We also point out a new and simple approach solving Matching Cut in diameter-2 graphs in polynomial time.

Notation and terminology. Let $G = (V, E)$ be a graph with vertex set $V(G) = V$ and edge set $E(G) = E$. An *independent set* (a *clique*) in G is a set of pairwise non-adjacent (adjacent) vertices. The neighborhood of a vertex v in G, denoted by $N_G(v)$, is the set of all vertices in G adjacent to v; if the context is clear, we simply write $N(v)$. For a subset $W \subseteq V$, $G[W]$ is the subgraph of G induced by W, and $G - W$ stands for $G[V \setminus W]$. The complete graph and the path on n vertices is denoted by K_n and P_n, respectively; K_3 is also called a *triangle*. The complete bipartite graph with one color class of size p and the other of size q is denoted by $K_{p,q}$. Observe that, for any matching cut (X, Y) of G, any K_n with $n \geq 3$, and any $K_{p,q}$ with $p \geq 2, q \geq 3$, in G is contained in $G[X]$ or else in $G[Y]$.

Given a graph $G = (V, E)$ and a partition $V = X \,\dot\cup\, Y$, it can be decided in linear time if (X, Y) is a matching cut of G. This is because (X, Y) is a matching cut of G if and only if the bipartite subgraph $B_G(X, Y)$ of G with the color classes X and Y and edge set (X, Y) is P_3-free. That is, (X, Y) is a matching cut of G if and only if the non-trivial connected components of the bipartite graph $B_G(X, Y)$ are edges. A path P_3 in $B_G(X, Y)$, if any, is called a *bad* P_3.

A *bridge* in a graph is an edge whose deletion increases the number of the connected components. Since disconnected graphs and graphs having a bridge have a matching cut, we

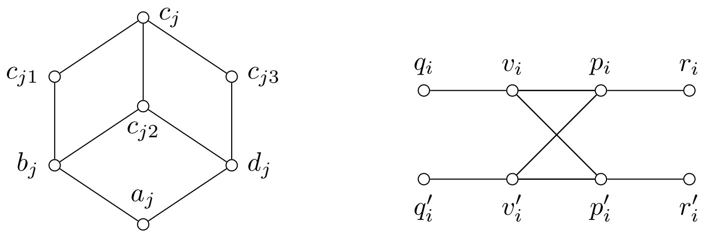


Figure 1 The clause gadget $G(C_j)$ (left) and the variable gadget $G(v_i)$ (right).

may assume that all graphs considered are connected and 2-edge connected. The *distance* between two vertices u, v in a (connected) graph G, denoted $\text{dist}(u, v)$, is the length of a shortest path connecting u and v. The *diameter* of G, denoted $\text{diam}(G)$, is the maximum distance between all pairs of vertices in G.

The paper is organized as follows. In Section 2 we show that Matching Cut is NP-complete when restricted to bipartite graphs of diameter d, for any fixed integer $d \geq 5$. In Section 3 we show that Matching Cut is NP-complete when restricted to graphs of diameter 4. In section 4 we point out a new and simple polynomial time algorithm solving Matching Cut in diameter 2 graphs, and show that Matching Cut can be solved in polynomial time for bipartite graphs of diameter at most 3. We conclude the paper with Section 5.

2 Matching Cut in bipartite graphs of diameter $d \geq 5$

In this section we show that, for each fixed $d \geq 5$, Matching Cut is NP-complete when restricted to bipartite graphs of diameter d. We first prove this for diameter-5 bipartite graphs by giving a polynomial time reduction from 1-IN-3 3SAT to our problem. The NP-completeness of 1-IN-3 3SAT is proved in [18].

1-IN-3 3SAT (without negated literals)

Instance: m clauses $C_1, \ldots, C_m$ over a set of n Boolean variables $v_1, \ldots, v_n$ such that each clause has exactly three variables.

Question: Is there a truth assignment satisfying all clauses such that each clause has exactly one true variable?

Suppose we are given a formula F with clauses $C_j = (c_{j1}, c_{j2}, c_{j3})$, where $c_{j\ell}$, $1 \leq j \leq m$, $1 \leq \ell \leq 3$, are taken from the set of Boolean variables $\{v_1, \ldots, v_n\}$. We will construct, in polynomial time, a bipartite graph $G(F)$ of diameter 5 such that $F \in$ 1-IN-3 3SAT if and only if $G(F)$ has a matching cut.

For each clause $C_j = (c_{j1}, c_{j2}, c_{j3})$ we create the 7-vertex graph $G(C_j)$ as depicted on the left-hand side of Figure 1.

The following property of the clause gadget $G(C_j)$ will be used in the reduction.

▶ **Observation 1.** *In any matching cut (X, Y) of $G(C_j)$, $c_j \in X$ and $a_j \in Y$ or vice versa. Moreover, $G(C_j)$ has exactly three matching cuts (X, Y) with $c_j \in X$ as depicted in Figure 2.*

Proof. By inspection. ◀

For each Boolean variable v_i, $1 \leq i \leq n$, let $G(v_i)$ be the 8-vertex graph depicted in Figure 1, on the right-hand side.

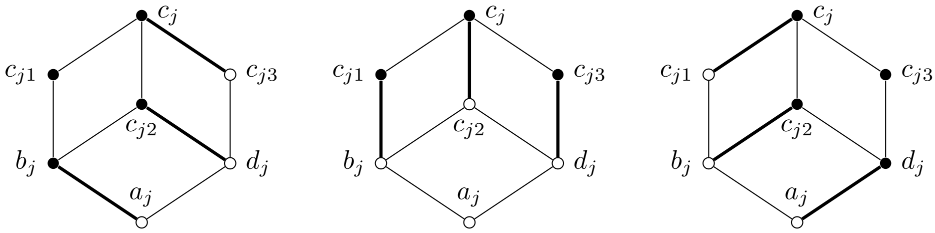


Figure 2 The three matching cuts of the clause gadget: black vertices in X, white vertices in Y.

Finally, the graph $G(F)$ is obtained by taking all $G(C_j)$, $1 \le j \le m$, all $G(v_i)$, $1 \le i \le n$, and four new vertices g_1, g_2, h_1, h_2, and by adding edges

- between $c_{j\ell}$ and $\{v_i, v_i'\}$, whenever in clause C_j, $c_{j\ell}$ is the variable v_i, $1 \le j \le m, 1 \le \ell \le 3$ and $1 \le i \le n$,
- between $\{g_1, g_2\}$ and $\{c_j \mid 1 \le j \le m\} \cup \{r_i, r_i' \mid 1 \le i \le n\}$ (thus, $\{g_1, g_2\}$ and $\{c_j \mid 1 \le j \le m\} \cup \{r_i, r_i' \mid 1 \le i \le n\}$ induce a complete bipartite graph $K_{2,m+2n}$ in $G(F)$),
- between $\{h_1, h_2\}$ and $\{a_j \mid 1 \le j \le m\} \cup \{q_i, q_i' \mid 1 \le i \le n\}$ (thus, $\{h_1, h_2\}$ and $\{a_j \mid 1 \le j \le m\} \cup \{q_i, q_i' \mid 1 \le i \le n\}$ induce a complete bipartite graph $K_{2,m+2n}$ in $G(F)$),
- g_1h_1 and g_2h_2.

▶ **Lemma 2.** *$G(F)$ is a bipartite graph and has diameter* 5.

Proof. By construction, the vertices of $G(F)$ are partitioned into two disjoint independent sets I_1, I_2 as follows.

$$I_1 = \{g_1, g_2\} \cup \{a_j, c_{j1}, c_{j2}, c_{j3} \mid 1 \le j \le m\} \cup \{p_i, p_i', q_i, q_i' \mid 1 \le i \le n\},$$
$$I_2 = \{h_1, h_2\} \cup \{b_j, c_j, d_j \mid 1 \le j \le m\} \cup \{v_i, v_i', r_i, r_i' \mid 1 \le i \le n\}.$$

That is, $G(F)$ is bipartite.

We now show that $\mathrm{diam}(G) = 5$. Observe first that any vertex x is contained in a 6-vertex cycle $Z(x)$ containing the edge g_1h_1. Indeed,

- $x \in \{g_2, h_2\}$: $(g_1, c_1, g_2, h_2, a_1, h_1)$ is such a cycle $Z(x)$;
- $x \in G(C_j)$: g_1, h_1 and any 4-vertex path in $G(C_j)$ connecting c_j and a_j containing x form such a cycle $Z(x)$;
- $x \in \{v_i, r_i, p_i, q_i\}$: $(g_1, r_i, p_i, v_i, q_i, h_1)$ is such a cycle $Z(x)$;
- $x \in \{v_i', r_i', p_i', q_i'\}$: $(g_1, r_i', p_i', v_i', q_i', h_1)$ is such a cycle $Z(x)$.

Thus, any two vertices x, y of $G(F)$ with $Z(x) \ne Z(y)$ belong to a cycle in $Z(x) \cup Z(y) \setminus \{g_1h_1\}$ of length at most 10, hence $\mathrm{dist}(x, y) \le 5$. Finally, observe that $\mathrm{dist}(p_i, v_k) = 5$ for any $i \ne k$. ◀

▶ **Lemma 3.** *Suppose $F \in$ 1-IN-3 3SAT. Then $G(F)$ has a matching cut.*

Proof. Let b be a truth assignment satisfying all $C_j = (c_{j1}, c_{j2}, c_{j3})$ such that exactly one of $\mathrm{b}(c_{j1}), \mathrm{b}(c_{j2}), \mathrm{b}(c_{j3})$ is true. Partition the vertex set of $G(F)$ as follows. Initially, set

$$X = \{g_1, g_2\} \cup \{r_i, r_i' \mid 1 \le i \le n\} \cup \{c_j \mid 1 \le j \le m\} \cup \{c_{j\ell} \mid 1 \le j \le m, 1 \le \ell \le 3, \mathrm{b}(c_{j\ell}) = \text{false}\}.$$

Next, extend X to other vertices of $G(C_j)$ as indicated in Figure 2. That is, for each $1 \le j \le m$,

- if $\mathrm{b}(c_{j1}) = \mathrm{b}(c_{j2}) =$ false, then $X := X \cup \{b_j\}$,
- if $\mathrm{b}(c_{j2}) = \mathrm{b}(c_{j3}) =$ false, then $X := X \cup \{d_j\}$.

Finally, extend X to other vertices of $G(v_i)$ as follows. For each $1 \le i \le n$,

- if $\mathrm{b}(v_i) =$ false, then $X := X \cup \{v_i, v'_i, p_i, p'_i\}$.

Set $Y := V(G(F)) \setminus X$, or explicitly in closed form,

$$\begin{aligned} Y = {} & \{h_1, h_2\} \cup \{q_i, q'_i \mid 1 \le i \le n\} \cup \{a_j \mid 1 \le j \le m\} \cup \\ & \{c_{j\ell} \mid 1 \le j \le m, 1 \le \ell \le 3, \mathrm{b}(c_{j\ell}) = \text{true}\} \cup \\ & \{b_j \mid 1 \le j \le m, \mathrm{b}(c_{j1}) = \text{true}\} \cup \{b_j, d_j \mid 1 \le j \le m, \mathrm{b}(c_{j2}) = \text{true}\} \cup \\ & \{d_j \mid 1 \le j \le m, \mathrm{b}(c_{j3}) = \text{true}\} \cup \{v_i, v'_i, p_i, p'_i \mid 1 \le i \le n, \mathrm{b}(v_i) = \text{true}\}. \end{aligned}$$

By construction of $G(F)$ and by definition of X and Y, it is not difficult to see that (X, Y) is a matching cut of $G(F)$.

▶ **Claim 4.** *(X, Y) is a matching cut of $G(F)$.*

Due to the space limitation, the proof of Claim 4 is omitted. ◀

▶ **Lemma 5.** *Suppose $G(F)$ has a matching cut. Then $F \in$ 1-IN-3 3SAT.*

Proof. Let (X, Y) be a matching cut of $G(F)$. Recall that each of

$$\begin{aligned} C &= \{g_1, g_2\} \cup \{c_j \mid 1 \le j \le m\} \cup \{r_i, r'_i \mid 1 \le i \le n\}, \\ A &= \{h_1, h_2\} \cup \{a_j \mid 1 \le j \le m\} \cup \{q_i, q'_i \mid 1 \le i \le n\} \end{aligned}$$

induces a $K_{2,m+2n}$ in $G(F)$, hence

$$C \subset X \text{ or } C \subset Y, \text{ and } A \subset X \text{ or } A \subset Y.$$

We claim that all c_j belong to X and all a_j belong to Y, or vice versa. Suppose for a contrary $c_j, a_j \in X$ for some j. Then, by the facts above, $C \cup A \subset X$. (The case $c_j, a_j \in Y$ is completely similar.) Then, for all j, $V(G(C_j)) \subset X$ (otherwise, $V(G(C_j)) \cap Y \ne \emptyset$ and the restriction of (X, Y) to $G(C_j)$ would be a matching cut of $G(C_j)$ with $c_j, a_j \in X$, contradicting Observation 1.) This implies that all v_i, v'_i, p_i, p'_i, $1 \le i \le n$, belong to X, too. (This is because v_i, v'_i, p_i, p'_i and the (common) neighbor $c_{j\ell}$ of v_i, v'_i in $G(C_j)$ for suitable j, ℓ induce a $K_{2,3}$.) Therefore all vertices of $G(F)$ belong to X and, hence, $Y = \emptyset$, a contradiction. Thus, all c_j belong to X and all a_j belong to Y, or vice versa, as claimed.

By symmetry we may assume that all $c_j \in X$ and all $a_j \in Y$. Define a truth assignment b as follows. For each $1 \le i \le n$, $\mathrm{b}(v_i) =$ false if $v_i \in X$, and $\mathrm{b}(v_i) =$ true if $v_i \in Y$. We claim that b satisfies all clauses $C_j = (c_{j1}, c_{j2}, c_{j3})$ in such a way that exactly one of $\mathrm{b}(c_{j1}), \mathrm{b}(c_{j2}), \mathrm{b}(c_{j3})$ is true: Consider an arbitrary clause C_j. By our assumption, in $G(C_j)$, $c_j \in X, a_j \in Y$, and thus the restriction of (X, Y) to $G(C_j)$ is a matching cut of $G(C_j)$. By Observation 1, exactly one of c_{j1}, c_{j2}, c_{j3} is in Y; see also Figure 2. Thus, $F \in$ 1-IN-3 3SAT. ◀

By Lemmas 3, 5 and 2, we conclude

▶ **Lemma 6.** Matching Cut *is NP-complete, even when restricted to bipartite graphs of diameter* 5.

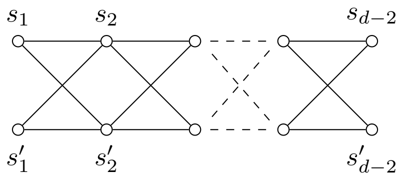


Figure 3 The bipartite chain B_d of $d-3$ $K_{2,2}$s.

Finally, let $d \geq 6$ be a fixed integer, and let B_d be a bipartite chain of $d-3$ complete bipartite graphs $K_{2,2}$ as depicted in Figure 3.

Let G_d be obtained by taking $G(F)$ in Lemma 2 and B_d by identifying the two vertices g_1 and s_1, and the two vertices g_2 and s'_1.

Clearly, G_d is bipartite. Recall that $\text{dist}(x, g_1) \leq 3$ and $\text{dist}(x, g_2) \leq 3$ for all vertices x of $G(F)$ (cf. the proof of Lemma 2). Hence $\text{dist}(x, y) \leq d$ for all vertices x, y of G_d, and $\text{dist}(x, y) = d$ for $x = p_i$ and $y = s_{d-2}$. Thus, G_d has diameter d. Observe that each vertex of B_d is contained in a $K_{2,3}$, hence in any matching cut (X, Y) of G_d, B_d is contained in $G_d[X]$ or else in $G_d[Y]$. Thus, $G(F)$ has a matching cut if and only if G_d has a matching cut. Hence, Lemma 6 implies

▶ **Theorem 7.** *For each fixed $d \geq 5$,* Matching Cut *is NP-complete, even when restricted to bipartite graphs of diameter d.*

3 Matching Cut in graphs of diameter 4

In this section we show that Matching Cut is NP-complete when restricted to graphs of diameter 4. The reduction is again from 1-IN-3 3SAT and is quite similar to that in Section 2. In particular we use the same clause gadget.

Suppose we are given a formula F with clauses $C_j = (c_{j1}, c_{j2}, c_{j3})$, where $c_{j\ell} \in \{v_1, \ldots, v_n\}$, $1 \leq j \leq m$, $1 \leq \ell \leq 3$. We will construct, in polynomial time, a graph $G(F)$ of diameter 4 such that $F \in$ 1-IN-3 3SAT if and only if $G(F)$ has a matching cut.

For each clause $C_j = (c_{j1}, c_{j2}, c_{j3})$ we create the graph $G(C_j)$ as depicted in Figure 1, on the left-hand side.

For each Boolean variable $v_i \in \{v_1, \ldots, v_n\}$, let $G(v_i)$ be the 4-cycle with vertices v_i, v'_i, q_i, q'_i and edges $v_i v'_i, v'_i q'_i, q'_i q_i, q_i v_i$.

Finally, the graph $G(F)$ is obtained by taking all $G(C_j)$, $1 \leq j \leq m$, all $G(v_i)$, $1 \leq i \leq n$, and by adding edges

- between $c_{j\ell}$ and v_i, v'_i, whenever in clause C_j, $c_{j\ell}$ is the variable v_i, $1 \leq j \leq m, 1 \leq \ell \leq 3$ and $1 \leq i \leq n$,
- between the vertices $c_1, \ldots, c_m$ (thus, $\{c_1, \ldots, c_m\}$ is a clique in $G(F)$),
- between the vertices $a_1, \ldots, a_m$, $q_1, \ldots, q_n$, $q'_1, \ldots, q'_n$ (thus, $\{a_j \mid 1 \leq j \leq m\} \cup \{q_i \mid 1 \leq i \leq n\} \cup \{q'_i \mid 1 \leq i \leq n\}$ is a clique in $G(F)$).

Informally, $G(F)$ is obtained from the bipartite graph constructed in Section 2 by deleting the vertices $g_1, g_2, h_1, h_2, p_i, p'_i, r_i, r'_i$ $(1 \leq i \leq n)$ and adding an edge between v_i and v'_i $(1 \leq i \leq n)$, and making $\{c_j \mid 1 \leq j \leq m\}$ to a clique, and $\{a_j \mid 1 \leq j \leq m\} \cup \{q_i, q'_i \mid 1 \leq i \leq n\}$ to a clique.

▶ **Lemma 8.** *$G(F)$ has diameter 4.*

Proof. Let x, y be two distinct vertices of $G(F)$. We will see that the distance between x and y is at most 4. First, note that $G(C_j)$ has diameter 3, $G(v_i)$ has diameter 2, and recall that $\{c_1, \ldots, c_m\}$, as well as $\{a_1, \ldots, a_m\} \cup \{q_i, q'_i \mid 1 \le i \le n\}$ are two cliques in $G(F)$, and that each vertex of $\{v_i, v'_i\}$ is adjacent to a vertex in $\{q_i, q'_i\}$. Hence we need only to consider three cases:

Case 1. $x \in G(C_j)$ and $y \in G(C_k)$ with $j \ne k$: In this case, x and y belong to a cycle of length 8: a 4-vertex path in $G(C_j)$ connecting c_j and a_j containing x and a 4-vertex path in $G(C_k)$ connecting c_k and a_k containing y form, with the edges c_jc_k and a_ja_k, a cycle of length 8. Hence $\text{dist}(x, y) \le 4$.

Case 2. $x \in G(v_i)$ and $y \in G(v_t)$ with $i \ne t$: In this case, $\text{dist}(x, y) \le 3$ because q_i, q'_i, q_t, q'_t are pairwise adjacent.

Case 3. $x \in G(C_j)$ and $y \in G(v_i)$: If $x \ne c_j$, then $\text{dist}(x, a_j) \le 2$ and, as a_j is adjacent to q_i, q'_i, $\text{dist}(a_j, y) \le 2$, hence $\text{dist}(x, y) \le \text{dist}(x, a_j) + \text{dist}(a_j, y) \le 4$. So, let $x = c_j$. If $y \in \{q_i, q'_i\}$, then, as y is adjacent to a_j, $\text{dist}(x, y) \le \text{dist}(c_j, a_j) + 1 = 4$. So, it remains the case $x = c_j, y \in \{v_i, v'_i\}$. Let k and ℓ be such that $v_i = c_{k\ell}$. Then $(y, c_{k\ell}, c_k, x)$ is a 4-vertex path, hence $\text{dist}(x, y) \le 3$.

Finally, note that $\text{dist}(b_1, b_2) = 4$. ◀

▶ **Lemma 9.** *Suppose $F \in$ 1-IN-3 3SAT. Then $G(F)$ has a matching cut.*

Proof. Let b be a truth assignment satisfying all $C_j = (c_{j1}, c_{j2}, c_{j3})$ such that exactly one of $\text{b}(c_{j1}), \text{b}(c_{j2}), \text{b}(c_{j3})$ is true. Partition the vertex set of $G(F)$ as follows. Initially, set

$$X = \{c_1, \ldots, c_m\} \cup \{c_{j\ell} \mid 1 \le j \le m, 1 \le \ell \le 3, \text{b}(c_{j\ell}) = \text{false}\}.$$

Next, extend X to other vertices of $G(C_j)$ as indicated in Figure 2. That is, for each $1 \le j \le m$,

- if $\text{b}(c_{j1}) = \text{b}(c_{j2}) = \text{false}$, then $X := X \cup \{b_j\}$,
- if $\text{b}(c_{j2}) = \text{b}(c_{j3}) = \text{false}$, then $X := X \cup \{d_j\}$.

Finally, extend X to other vertices of $G(v_i)$ as follows. For each $1 \le i \le n$,

- if $\text{b}(v_i) = \text{false}$, then $X := X \cup \{v_i, v'_i\}$.

Then (X, Y) is a matching cut of $G(F)$, where $Y = V(G(F)) \setminus X$. We omit the proof since it is similar to that of Lemma 3. ◀

▶ **Lemma 10.** *Suppose $G(F)$ has a matching cut. Then $F \in$ 1-IN-3 3SAT.*

Proof. Let (X, Y) be a matching cut of $G(F)$. Note that, as $C = \{c_j \mid 1 \le j \le m\}$ and $A = \{a_j, q_j, q'_j \mid 1 \le j \le m\}$ are cliques in $G(F)$,

$$C \subset X \text{ or } C \subset Y, \text{ and } A \subset X \text{ or } A \subset Y.$$

Similar to the proof of Lemma 5, we can show that all c_j belong to X and all a_j belong to Y, or vice versa. Let $C \subset X$ and $A \subset Y$, say. Then, the assignment b with $\text{b}(v_i) = \text{false}$ if $v_i \in X$, and $\text{b}(v_i) = \text{true}$ if $v_i \in Y$ satisfies all clauses $C_j = (c_{j1}, c_{j2}, c_{j3})$ in such a way that exactly one of $\text{b}(c_{j1}), \text{b}(c_{j2}), \text{b}(c_{j3})$ is true. Hence, $F \in$ 1-IN-3 3SAT. ◀

By Lemmas 9, 10 and 8, we conclude

▶ **Theorem 11.** Matching Cut *is NP-complete when restricted to graphs of diameter* 4.

With Theorem 7, Matching Cut is NP-complete when restricted to graphs of diameter d for any fixed $d \ge 4$.

4 Matching Cut in bipartite graphs of diameter at most 3

In this section we prove that MATCHING CUT can be solved in polynomial time when restricted to bipartite graphs of diameter at most 3. To do this, we first prove a lemma that will be useful in many cases. In particular, we will drive from this lemma a new and simple polynomial-time algorithm solving MATCHING CUT in graphs of diameter 2.

4.1 A useful lemma

Given a graph $G = (V, E)$ and two disjoint, non-empty vertex sets $A, B \subset V$. We say a matching cut of G is an A-B-*matching cut* if A is contained in one side and B is contained in the other side of the matching cut.

In general, unless NP $\neq$ P, we cannot decide in polynomial time if G admits an A-B-matching cut for a given pair A, B. However, there are some rules that force certain vertices some of which together with A must belong to one side and the other together with B muss belong to the other side of such a matching cut (if any). We are going to describe such forcing rules. Now assume that A, B are disjoint, non-empty subsets of $V(G)$ such that each vertex in A is adjacent to exactly one vertex of B and each vertex in B is adjacent to exactly one vertex of A. Initially, set $X := A$, $Y := B$ and write $R = V(G) \setminus (X \cup Y)$.

(R1) Let $v \in R$ be adjacent to a vertex in A. If v is
- adjacent to a vertex in B, or
- adjacent to (at least) two vertices in $Y \setminus B$,

then G has no A-B-matching cut.

(R2) Let $v \in R$ be adjacent to a vertex in B. If v is
- adjacent to a vertex in A, or
- adjacent to (at least) two vertices in $X \setminus A$,

then G has no A-B-matching cut.

(R3) If $v \in R$ is adjacent to (at least) two vertices in $X \setminus A$ and to (at least) two vertices in $Y \setminus B$, then G has no A-B-matching cut.

(R4) Let $v \in R$ be adjacent to a vertex in A or to (at least) two vertices in $X \setminus A$, then $X := X \cup \{v\}$, $R := R \setminus \{v\}$. If v has a unique neighbor $w \in Y \setminus B$ then $A := A \cup \{v\}$, $B := B \cup \{w\}$.

(R5) Let $v \in R$ be adjacent to a vertex in B or to (at least) two vertices in $Y \setminus B$, then $Y := Y \cup \{v\}$, $R := R \setminus \{v\}$. If v has a unique neighbor $w \in X \setminus A$ then $B := B \cup \{v\}$, $A := A \cup \{w\}$.

It is obvious that the rules (R1)–(R5) are correct. If none of (R1), (R2) and (R3) is applicable, then each vertex $v \in R$ has no neighbor in A or has no neighbor in B, and v has at most one neighbor in $X \setminus A$ or has at most one neighbor in $Y \setminus B$. If (R4) is not applicable, then each vertex $v \in R$ has no neighbor in A and at most one neighbor in $X \setminus A$. If (R5) is not applicable, then each vertex $v \in R$ has no neighbor in B and at most one neighbor in $Y \setminus B$. Thus, the following fact holds:

▶ **Fact 12.** *Suppose none of (R1)–(R5) is applicable. Then*
- *(X, Y) is an A-B-matching cut of $G[X \cup Y]$, and any A-B-matching cut of G must contain X in one side and Y in other side;*
- *for any vertex $v \in R$, $N(v) \cap A = N(v) \cap B = \emptyset$ and $|N(v) \cap X| \leq 1$, $|N(v) \cap Y| \leq 1$.*

We say that a subset $S \subseteq R$ is *monochromatic* if, for any A-B-matching cut of G, all vertices of S belong to the same side.

▶ **Lemma 13.** *Suppose none of the rules (R1)–(R5) is applicable. Assuming each connected component of $G - (X \cup Y)$ is monochromatic, it can be decided in time $O(|V|^2|E|)$ if G admits an A-B-matching cut.*

Proof. Let Z be a connected component of $G-(X \cup Y)$, and let (X', Y') be an A-B-matching cut of G with $A \subseteq X'$ and $B \subseteq Y'$. Note, by Fact 12, then $X \subseteq X'$ and $Y \subseteq Y'$. Since Z is monochromatic, we have

- $Z \subseteq X'$ whenever some vertex in $X \setminus A$ has at least two neighbors in Z.
- Similarly, $Z \subseteq Y'$ whenever some vertex in $Y \setminus B$ has at least two neighbors in Z.
- If a vertex in $X \setminus A$ has neighbors in two connected components of $G - (X \cup Y)$, then at least one of these components is contained in X'.
- Similarly, if a vertex in $Y \setminus B$ has neighbors in two connected components of $G - (X \cup Y)$, then at least one of these components is contained in Y'.

Thus, we can decide if G admits a matching cut (X', Y') such that $X \subseteq X', Y \subseteq Y'$, by solving the following instance $F(G)$ of the 2-SAT problem.

- For each connected component C of $G - (X \cup Y)$, create two Boolean variables x_C, y_C. The intention is that x_C is set to true if C must go to X and y_C is set to true if C muss go to Y. Then $(x_C \vee y_C)$ and $(\neg x_C \vee \neg y_C)$ are two clauses of the formula $F(G)$.
- For each connected component C of $G-(X \cup Y)$ with $|N(v) \cap C| \geq 2$ for some $v \in X \setminus A$, (x_C) is a clause of the formula $F(G)$. This clause ensures that in this case, C must go to X.
- For each connected component C of $G-(X \cup Y)$ with $|N(w) \cap C| \geq 2$ for some $w \in Y \setminus B$, (y_C) is a clause of the formula $F(G)$. This clause ensures that in this case, C must go to Y.
- For each two connected components $C \neq D$ of $G - (X \cup Y)$ having a common neighbor in $X \setminus A$, $(x_C \vee x_D)$ is a clause of the formula $F(G)$. This clause ensures that in this case, at least one of C and D must go to X.
- For each two connected components $C \neq D$ of $G - (X \cup Y)$ having a common neighbor in $Y \setminus B$, $(y_C \vee y_D)$ is a clause of the formula $F(G)$. This clause ensures that in this case, at least one of C and D must go to Y.

▶ **Claim 14.** *G admits a matching cut (X', Y') such that $X \subseteq X', Y \subseteq Y'$ if and only if $F(G)$ is satisfiable.*

Due to the space limitation, the proof of Claim 14 is omitted.

Note that $F(G)$ has $O(|V|)$ variables and $O(|V|^2)$ clauses and can be constructed in time $O(|V|^2|E|)$. Since 2-SAT can be solved in linear time (cf. [3, 8, 11]), by Claim 14 we can decide in time $O(|V|^2|E|)$ if G admits an A-B-matching cut. ◀

4.2 Diameter 2 graphs: A new, simple and faster polynomial-time algorithm

Let $G = (V, E)$ be a graph of diameter 2. Guess an edge ab of G, and apply rules (R1)–(R5) for $A := \{a\}$, $B := \{b\}$ as long as possible. If (R1) or (R2) or (R3) is applicable, then clearly G has no A-B-matching cut. So let us assume that none of (R1), (R2) and (R3) was ever applied and none of (R4) and (R5) is applicable. Then each connected component Z of $G - (X \cup Y)$ is monochromatic. To see this, let (X', Y') be an A-B-matching cut of G with $X \subset X'$, $Y \subset Y'$. By Fact 12, any vertex in $A \cup B$ is non-adjacent to any vertex in Z, any vertex in A has neighbors only in $X \cup B$, any vertex in B has neighbors only in $Y \cup A$. Since

G has diameter 2, therefore, $N(v) \cap N(a) \neq \emptyset$ for all $v \in Z$ and $a \in A$. Thus, v must have a neighbor in $X \setminus A$. Similarly, each vertex $v \in Z$ must have a neighbor in $Y \setminus B$. Now, suppose for a contrary that Z is not monochromatic. Then, by connectedness, there is an edge uv in Z with $u \in X'$ and $v \in Y'$, say. But then u, v and a neighbor of v in $X \subseteq X'$ induce a bad P_3, contradicting the assumption that (X', Y') is a matching cut.

Thus, any connected component of $G - (X \cup Y)$ is monochromatic. Therefore, by Lemma 13, it can be decided in time $O(|V|^2|E|)$ if G has an A-B-matching cut. Since we have at most $|E|$ many choices for the edge ab, we conclude that Matching Cut can be solved in time $O(|V|^2|E|^2)$ for graphs of diameter two. We remark that the known algorithm posed in [6] has slower running time $O(|V|^2|E|^3)$. Moreover, in comparison to their algorithm, our is much simpler.

4.3 Diameter 3 bipartite graphs

Given a connected bipartite graph $G = (V, E)$ with a bipartition $V = V_1 \dot\cup V_2$ into independent sets V_1, V_2. We will use the following fact which is easy to see:

$$\begin{aligned}&G \text{ has diameter at most 3 if and only if, for each } i = 1, 2 \\ &\text{and for every two vertices } u, v \in V_i,\ N(u) \cap N(v) \neq \emptyset.\end{aligned} \tag{1}$$

Let G have diameter at most 3. Since graphs having a bridge have a matching cut, we may assume that G is 2-edge connected. Hence every matching cut of G, if any, must have at least two edges. Our algorithm consists of two phases. In the phase 1, we will check if G has a matching cut (X, Y) containing two edges a_1b_1, a_2b_2 such that $a_1, a_2 \in V_1$, $b_1, b_2 \in V_2$ and $\{a_1, b_2\} \subseteq X$ and $\{a_2, b_1\} \subseteq Y$. In case phase 1 is unsuccessful, phase 2 will be started. In the phase 2, we will check if G has a matching cut (X, Y) containing two edges a_1b_1, a_2b_2 such that $a_1, a_2 \in V_1$, $b_1, b_2 \in V_2$ and $\{a_1, a_2\} \subseteq X$ and $\{b_1, b_2\} \subseteq Y$. The fact that G has diameter at most 3 will ensure that each of phase 1 and 2 can be performed in polynomial time.

Phase 1. Guess two edges $a_1b_1, a_2b_2 \in E$ such that $a_1, a_2 \in V_1$ and $b_1, b_2 \in V_2$. Set $X = A = \{a_1, b_2\}$, $Y = B = \{a_2, b_1\}$ and write $R = V(G) \setminus (X \cup Y)$. Apply (R1)–(R5) as long as possible, and let us assume that none of (R1), (R2) and (R3) was ever applied.

Then each connected component Z of $G - (X \cup Y)$ is monochromatic. Indeed, let (X', Y') be an A-B-matching cut with $X \subseteq X'$, $Y \subseteq Y'$, and consider an arbitrary vertex $v \in Z$. If $v \in V_1$, then, since $a_1 \in A \cap V_1$ and $a_2 \in B \cap V_1$, v must have, by (1), a neighbor in X and a neighbor in Y. Similarly, if $v \in V_2$, then, since $b_1 \in A \cap V_2$ and $b_2 \in B \cap V_2$, v must have a neighbor in X and a neighbor in Y, too. Thus, every vertex in Z has a neighbor in X and a neighbor in Y. Therefore, by the same argument explained in the diameter 2 case, Z is monochromatic. Hence, by Lemma 13, we can decide in polynomial time if G has an A-B-matching cut. Since there are at most $|E|^2$ choices for A and B, we conclude that, in polynomial time, phase 1 can decide if G has a matching cut (X, Y) containing two edges a_1b_1, a_2b_2 such that $a_1, a_2 \in V_1$, $b_1, b_2 \in V_2$ and $\{a_1, b_2\} \subseteq X$ and $\{a_2, b_1\} \subseteq Y$.

Phase 2. In this second phase we assume that phase 1 is unsuccessful, that is, G has no matching cut (X, Y) containing two edges a_1b_1, a_2b_2 such that $a_1, a_2 \in V_1$, $b_1, b_2 \in V_2$ and $\{a_1, b_2\} \subseteq X$ and $\{a_2, b_1\} \subseteq Y$.

Guess an edge $ab \in E$ such that $a \in V_1$ and $b \in V_2$. Set $X = A = \{a\}$, $Y = B = \{b\}$ and write $R = V(G) \setminus (X \cup Y)$. Apply (R1)–(R5) as long as possible, and let us assume that none of (R1), (R2) and (R3) was ever applied.

Then, by (1), every vertex $v \in R_1 = R \cap V_1$ has a neighbor in X (as $a \in A \cap V_1$) and every vertex $w \in R_2 = R \cap V_2$ has a neighbor in Y (as $b \in B \cap V_2$). Thus, assuming G has an A-B-matching cut (X', Y') with $X \subseteq X', Y \subseteq Y'$, then R_1 must belong to X' and R_2 must belong to Y'. For, if $v \in R_1$ was in Y', then the A-B-matching cut (X', Y') would contain the edges ab and uv, where u is the neighbor of v in $X \subseteq X'$, with $a, v \in V_1$ and $b, u \in V_2$, contradicting the assumption that phase 1 was unsuccessful. The case of R_2 is completely similar. Therefore, G has an A-B-matching cut (X', Y') with $X \subseteq X', Y \subseteq Y'$ if and only if $(X \cup R_1, Y \cup R_2)$ is a matching cut. As the second property can be decided in polynomial time, and there are at most $|E|$ choices for A and B, we conclude that phase 2 can be performed in polynomial time. Putting all together we obtain:

▶ **Theorem 15.** Matching Cut *can be solved in polynomial time when restricted to bipartite graphs of diameter at most* 3.

5 Concluding remarks

In this paper we have shown that Matching Cut is NP-complete when restricted to graphs of diameter d, for fixed $d \geq 4$, and to bipartite graphs of diameter d, for fixed diameter $d \geq 5$. We also have given a polynomial-time algorithm solving Matching Cut in bipartite graphs of diameter at most 3. It is known that Matching Cut is polynomially solvable when restricted to graphs of diameter 2 ([6]; cf. also Section 4). Thus, it would be very interesting to close the gap, obtaining a dichotomy theorem:

- What is the computational complexity of Matching Cut in graphs of diameter 3?
- What is the the computational complexity of Matching Cut in bipartite graphs of diameter 4?

Finally, we remark that Matching Cut can be solved in linear time in planar graphs (and in graphs with fixed genus) of fixed diameter. This is because a planar graph with diameter d has tree-width at most $3d - 2$ ([9]). (More general, a graph with diameter d and genus g has tree-width $O(gd)$; see [10].) In [5], it is shown that Matching Cut can be expressed in monadic second order logic (MSOL), and it is well-known ([2]) that all graph properties definable in MSOL can be decided in linear time for classes of graphs with bounded tree-width, when a tree-decomposition is given. It is also well-known ([4]) that a tree-decomposition of bounded width of a given graph can be found in linear time. Combining these facts, it follows that Matching Cut can be solved in linear time for planar graphs (and in graphs with fixed genus) of fixed diameter.

References

1 Júlio Araújo, Nathann Cohen, Frédéric Giroire, and Frédéric Havet. Good edge-labelling of graphs. *Discrete Applied Mathematics*, 160(18):2502–2513, 2012. doi:10.1016/j.dam.2011.07.021.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

3 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123. Erratum: 14 (1982) 195, 1979. doi:10.1016/0020-0190(79)90002-4.

4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

5 Paul S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62(2):109–126, 2009. doi:10.1002/jgt.20390.

6 Mieczyslaw Borowiecki and Katarzyna Jesse-Józefczyk. Matching cutsets in graphs of diameter 2. *Theor. Comput. Sci.*, 407(1-3):574–582, 2008. doi:10.1016/j.tcs.2008.07.002.

7 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8(1):51–53, 1984. doi:10.1002/jgt.3190080106.

8 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.

9 David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3), 1999. URL: http://www.cs.brown.edu/publications/jgaa/accepted/99/Eppstein99.3.3.pdf.

10 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/s004530010020.

11 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976. doi:10.1137/0205048.

12 Arthur M. Farley and Andrzej Proskurowski. Networks immune to isolated line failures. *Networks*, 12(4):393–403, 1982. doi:10.1002/net.3230120404.

13 Ron L. Graham. On primitive graphs and optimal vertex assignments. *Ann. N.Y. Acad. Sci.*, 175:170–186, 1970. URL: http://www.math.ucsd.edu/~ronspubs/70_04_primitive_graphs.pdf.

14 Dieter Kratsch and Van Bang Le. Algorithms solving the matching cut problem. *Theor. Comput. Sci.*, 609:328–335, 2016. doi:10.1016/j.tcs.2015.10.016.

15 Van Bang Le and Bert Randerath. On stable cutsets in line graphs. *Theor. Comput. Sci.*, 1-3(301):463–475, 2003. doi:10.1016/S0304-3975(03)00048-3.

16 Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13(5):527–536, 1989. doi:10.1002/jgt.3190130502.

17 Maurizio Patrignani and Maurizio Pizzonia. The complexity of the matching-cut problem. In *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, pages 284–295, 2001. doi:10.1007/3-540-45477-2_26.

18 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226, 1978. doi:10.1145/800133.804350.

On the Optimality of Tape Merge of Two Lists with Similar Size*

Qian Li[1], **Xiaoming Sun**[2], **and Jialin Zhang**[3]

1 CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; and University of Chinese Academy of Sciences, Beijing, China
liqian@ict.ac.cn

2 CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; and University of Chinese Academy of Sciences, Beijing, China
sunxiaoming@ict.ac.cn

3 CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; and University of Chinese Academy of Sciences, Beijing, China
zhangjialin@ict.ac.cn

Abstract

The problem of merging sorted lists in the least number of pairwise comparisons has been solved completely only for a few special cases. Graham and Karp [18] independently discovered that the tape merge algorithm is optimal in the worst case when the two lists have the same size. Stockmeyer and Yao[28], Murphy and Paull[24], and Christen[6] independently showed when the lists to be merged are of size m and n satisfying $m \le n \le \lfloor \frac{3}{2}m \rfloor + 1$, the tape merge algorithm is optimal in the worst case. This paper extends this result by showing that the tape merge algorithm is optimal in the worst case whenever the size of one list is no larger than 1.52 times the size of the other. The main tool we used to prove lower bounds is Knuth's adversary methods [18]. In addition, we show that the lower bound cannot be improved to 1.8 via Knuth's adversary methods. We also develop a new inequality about Knuth's adversary methods, which might be interesting in its own right. Moreover, we design a simple procedure to achieve constant improvement of the upper bounds for $2m - 2 \le n \le 3m$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases comparison-based sorting, tape merge, optimal sort, adversary method

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.51

1 Introduction

Suppose there are two disjoint linearly ordered lists A and B: $a_1 < a_2 < \cdots < a_m$ and $b_1 < b_2 < \cdots < b_n$ respectively, where the $m + n$ elements are distinct. The problem of merging them into one ordered list is one of the most fundamental algorithmic problems which has many practical applications as well as important theoretical significance. This problem has been extensively studied under different models, such as comparison-based model [18], parallel model [9], in-place merging model [11], etc. In this paper we focus on the classical

* This work was supported in part by the National Natural Science Foundation of China Grant 61222202, 61433014, 61502449, 61602440 and the China National Program for support of Top-notch Young Professionals.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 51; pp. 51:1–51:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

comparison-based model, where the algorithm is a sequence of pairwise comparisons. People are interested in this model due to two reasons. Firstly, it is independent to the underlying order relation used, no matter it is "<" in **R** or another abstract order relation. Secondly, it is unnecessary to access the value of elements in this model. Such a restriction could come from security or privacy concerns where the only operation available is a zero-knowledge pairwise comparison which reveals only the order relation between elements.

The main theoretical question in this merge problem is to determine $M(m, n)$, the minimum number of comparisons which is always sufficient to merge the lists[18]. Given any algorithm g[1] to solve the (m, n) merging problem (i.e. where $|A| = m$ and $|B| = n$), let $M_g(m, n)$ be the number of comparisons required by algorithm g in the worst case, then

$$M(m, n) = \min_g M_g(m, n).$$

An algorithm g is said to be optimal on (m, n) if $M_g(m, n) = M(m, n)$. By symmetry, it is clear that $M(m, n) = M(n, m)$. To much surprise, this problem seems quite difficult in general, and exact values are known for only a few special cases. Knuth determined the value of $M(m, n)$ for the case $m, n \leq 10$ in his book [18]. Graham [18] and Hwang and Lin [16] completely solved the case $m = 2$ independently. The case $m = 3$ is quite a bit harder and was solved by Hwang [15] and Murphy [25]. Mönting solved the case $m = 4$ and also obtained strong results about $m = 5$ [26]. In addition, Smith and Lang [27] devised a computer program based on game solver techniques such as alpha beta search to compute $M(m, n)$. They uncovered many interesting facts including $M(7, 12) = 17$, while people used to believe $M(7, 12) = 18$.

Several different algorithms have been developed for the merge problem, among them *tape merge* or *linear merge* might be the simplest and the most commonly used one. In this algorithm, two smallest elements (initially a_1 and b_1) are compared, and the smaller one will be deleted from its list and placed on the end of the output list. Then repeat the process until one list is exhausted. It's easy to see that this algorithm requires $m+n-1$ comparisons in the worst case, hence $M(m, n) \leq m + n - 1$. However, when m is much smaller than n, it is obvious that this algorithm becomes quite inefficient. For example, when $m = 1$, the merging problem is equivalent to an insertion problem and the rather different *binary insertion* procedure is optimal, i.e. $M(1, n) = \lceil \lg(n + 1) \rceil$.

One nature question is "*when is tape merge optimal?*". By symmetry, we can assume $n \geq m$ and define $\alpha(m)$ be the maximum integer $n(\geq m)$ such that tape merge is optimal, i.e.

$$\alpha(m) = \max\{n \in \mathbb{N} \mid M(m, n) = m + n - 1, n \geq m\}.$$

Assume a conjecture proposed by Knuth [18], which asserts that $M(m, n+1) \leq M(m, n)+1 \leq M(m + 1, n)$, for $m \leq n$, is correct, it's easy to see tape merge is optimal if and only if $n \leq \alpha(m)$, and $\alpha(m)$ is monotone increasing.

Graham and Karp [18] independently discovered that $M(m, m) = 2m - 1$ for $m \geq 1$. Then Knuth [18] proved $\alpha(m) \geq 4$ for $m \leq 6$. Stockmeyer and Yao[28], Murphy and Paull[24], and Christen[6] independently significantly improved the lower bounds by showing $\alpha(m) \geq \lfloor \frac{3}{2}m \rfloor + 1$, that is $M(m, n) = m + n - 1$, for $m \leq n \leq \lfloor \frac{3}{2}m \rfloor + 1$. On the other hand, Hwang[14] showed that $M(m, 2m) \leq 3m - 2$, which implies $\alpha(m) \leq 2m - 1$. For

[1] We only consider deterministic algorithms in this paper.

$m \le n \le 2m-1$, the best known merge algorithm is tape merge algorithm. It is conjectured by Fernandez et al. [8] that $\alpha(m) = \frac{1+\sqrt{5}}{2}m \pm o(m)$.

For general $n \ge m$, Hwang and Lin[14] proposed an in-between algorithm called *binary merge*, which excellently compromised between binary insertion and tape merge in such a way that the best features of both are retained. It reduces to tape merge when $n \le 2m$, and reduces to binary insertion when $m = 1$. Let $M_{bm}(m,n)$ be the worst-case complexity of this algorithm. They showed that

$$M_{bm}(m,n) = m(1 + \lfloor \lg \frac{n}{m} \rfloor) + \lfloor \frac{n}{2^{\lfloor \lg \frac{n}{m} \rfloor}} \rfloor - 1.$$

Hwang and Deutsch[13] designed an algorithm which is optimal over all *insertive algorithms* including binary merge, where for each element of the smaller list, the comparisons involving it are made consecutively. However, the improvement for fixed n/m over binary merge increases more slowly than linearly in m[23]. Here we say that algorithm A_1 with complexity $M_{A_1}(m,n)$ is significantly faster for some fixed ratio n/m than algorithm A_2 with complexity $M_{A_2}(m,n)$, if $M_{A_2}(m,n) - M_{A_1}(m,n) = \Omega(m)$. The first significant improvement over binary merge was proposed by Manacher[23], which can decrease the number of comparisons by $\frac{31}{336}m$ for $n/m \ge 8$, and Thanh and Bui[31] further improved this number to $\frac{13}{84}m$. In 1978, Christen[5] proposed an elegant algorithm, called *forward testing and backward insertion*, which is better than binary merge when $n/m \ge 3$ and saves at least $\sum_{j=1}^{k} \lfloor \frac{m-1}{4^j} \rfloor$ comparisons over binary merging, for $n \ge 4^k m$. Thus it saves about $m/3$ comparisons when $n/m \to \infty$. Moreover, Christen's procedure is optimal for $5m - 3 \le n \le 7m$, i.e. $M(m,n) = \lfloor (11m + n - 3)/4 \rfloor$.

On the lower bound side, there are two main techniques in proving lower bounds. The first one is the information theoretic lower bound $I(m,n) = \lceil \lg \binom{m+n}{m} \rceil$. Hwang and Lin[14] have proved that

$$I(m,n) \le M(m,n) \le M_{bm}(m,n) \le I(m,n) + m.$$

The second one is called *Knuth's adversary methods* [18]. The idea is that the optimal merge problem can be viewed as a two-player game with perfect information, in which the algorithm chooses the comparisons, while the adversary chooses (consistently) the results of these comparisons. It is easy to observe that $M(m,n)$ is actually the min-max value of this game. Thus a given strategy of the adversary provides a lower bound for $M(m,n)$. Mainly because of the consistency condition on the answers, general strategies are rather tedious to work with. Knuth proposed the idea of using of "disjunctive" strategies, in which a splitting of the remaining problem into two disjoint problems is provided, in addition to the result of the comparison. With this restricted adversary, he used term $.M.(m,n)$ to represent the minimum number of comparisons required in the algorithm, which is also a lower bound of $M(m,n)$. The detail will be specified in Section 2.

1.1 Our Results

In this paper, we first improve the lower bounds of $\alpha(m)$ from $\lfloor \frac{3}{2}m \rfloor + 1$ to $\lfloor \frac{38}{25}m \rfloor$ by using Knuth's adversary methods.

▶ **Theorem 1.** *$M(m,n) = m + n - 1$, if $m \le n \le \frac{38}{25}m$.*

We then show limitations of Knuth's adversary methods.

▶ **Theorem 2.** *$.M.(m,n) < m + n - 1$, if $n \ge 9\lceil m/5 \rceil$.*

This means that by using Knuth's adversary methods, it's impossible to show $\alpha(m) \geq 9\lceil m/5 \rceil \approx \frac{9}{5}m$ for any m.

When $m \leq n \leq 3m$, binary merge is the best known algorithm, which reduces to tape merge for $n \in [m, 2m]$ and gives $M_{bm}(m, 2m+k) = 3m + \lfloor k/2 \rfloor - 1$ for $k \in [0, m]$. In this paper, we give improved upper bounds for $M(m,n)$ for $2m - 2 \leq n \leq 3m$. In particular, it also improves the upper bounds of $\alpha(m)$, that is, $\alpha(m) \leq 2m - 3$ for $m \geq 7$.

▶ **Theorem 3.**

(a) $M(m, 2m+k) \leq 3m + \lfloor k/2 \rfloor - 2 = M_{bm}(m, 2m+k) - 1$, *if* $m \geq 5$ *and* $k \geq -1$.

(b) $M(m, 2m-2) \leq 3m - 4 = M_{bm}(m, 2m-2) - 1$, *if* $m \geq 7$. *That is* $\alpha(m) \leq 2m - 3$ *for* $m \geq 7$.

(c) $M(m, 2m) \leq 3m - 3 = M_{bm}(m, 2m) - 2$, *if* $m \geq 10$.

1.2 Related work

Besides the worst-case complexity, the average-case complexity has also been investigated for merge problems. Tanner [29] designed an algorithm which uses at most $1.06 I(m,n)$ comparisons on average. The average case complexity of insertive merging algorithms as well as binary merge has also been investigated [7, 8].

Bui et al. [30] gave the optimal randomized algorithms for $(2,n)$ and $(3,n)$ merge problems and discovered that the optimal expected value differs from the optimal worst-case value by at most 1. Fernandez et al. [8] designed a randomized merging algorithm which performs well for any ratio n/m and is significantly faster than binary merge for $n/m > (\sqrt{5}+1)/2 \approx 1.618$. More preciously, they showed that

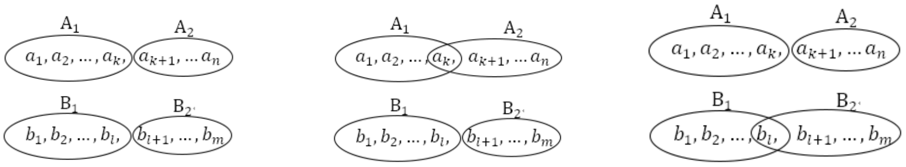
$$M_F(m,n) = \begin{cases} sn + (1+s)m, & if \quad 1 + s \leq n/m \leq 2 + s, \\ 2\sqrt{mn}, & if \quad 2 + s \leq n/m \leq 2r, \end{cases} \tag{1}$$

where $s = (\sqrt{5} - 1)/2 \approx 0.618$ and $r = (\sqrt{2} - 1 + \sqrt{2}s)^2 \approx 1.659$.

Nathan Linial [21] studied a more general problem where partial order relations are already known. He showed the information-theoretic lower bound is good, that is, an algorithms exists which merges A and B in no more than $(\lg(\sqrt{5}+1)/2)^{-1} \lg N_0$ comparisons, where N_0 is the number of extensions of the partial order on $A \cup B$. They also pointed out that this bound is tight, and the computation needed for finding the appropriate queries can be done in time polynomial in $m + n$.

Sorting, merging and searching are always closely related to each other. Manacher et al. [22] used efficient merge algorithms to improve Ford-Johnson sorting algorithm, which was conjectured to be optimal for almost twenty years. Linial and Saks [20] observed that $M(m,n)$ is equivalent to the minimum number of pairwise comparisons to determine whether a given element is present in a $m \times n$ matrix in which distinct entries are known to be increasingly ordered along rows and columns. They also studied the generalized problem in monotone multi-dimensional arrays, and their result was further improved by Cheng et al. [4]. Ajtai et al. [1] considered the problem of sorting and selection with imprecise comparisons. The non-uniform cost model has also been investigated [3, 10, 17, 12], for example, Huang et al. [12] studied the sorting problem where only a subset of all possible pairwise comparisons is allowed. For practical use, Brown and Tarjan [2] gave a merging procedure which runs in $O(I(m,n))$ time on a real computer.

Organization. We introduce some notations and explain Knuth's adversary methods in Section 2. In Section 3 we provide some properties of Knuth's adversary methods which will



(a) Case 1, simple strategy **(b)** Case 2, complex strategy **(c)** Case 3, complex strategy

Figure 1 The Adversary's Splitting Strategies.

be used. In Section 4 we improve the lower bounds for $\alpha(m)$ via Knuth's adversary methods. Then we show limitations of this method in Section 5. Section 6 improves the upper bounds for $M(m, 2m + k)$. We conclude the paper with some open problems in Section 7.

2 Preliminaries

In this section, we introduce Knuth's adversary methods and present some notations.

We use the notations $\lambda M \rho$ proposed by Knuth in this paper. The detailed definitions can be found in Knuth's comprehensive monograph [18]. Yao et al. [28] gave an example to illustrate the use of that. Here, we briefly introduce the idea.

The basic idea of Knuth's adversary methods is to restrict the possible adversary strategies. In general, the adversary can arbitrarily answer the comparison query from the algorithm as long as there are no contradictions in his answer. But in Knuth's adversary methods, after each comparison query between a_i and b_j, the adversary is required to split each sorted list into two parts $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$ (Figure 1). The adversary guarantees that each element in A_1 or B_1 is smaller than any element in A_2 or B_2. It is also guaranteed that a_i and b_j are not in the same subproblem, i.e. neither $a_i \in A_1, b_j \in B_1$ nor $a_i \in A_2, b_j \in B_2$, thus, the comparison result between a_i and b_j is determined after the splitting. Then the merge problem will be reduced to two subproblems (A_1, B_1) and (A_2, B_2) with different left or right constraints. For example, in case 2, the constraint for subproblem (A_1, B_1) is a right constraint $b_l < a_k$ since $a_k \in A_2$ while $b_l \in B_1$.

Knuth introduced notation $\lambda M \rho$ to represent nine kinds of restrict adversaries, where $\lambda, \rho \in \{., \backslash, /\}$ are the left and right constraint. In general, the constraint notation $'.'$ means no left (or right) constraint. Left constraint $\lambda = \backslash$ or $/$ means that outcomes must be consistent with $a_1 < b_1$ or $a_1 > b_1$ respectively. Similarly, right constraint $\rho = \backslash(/)$ means the outcomes must be consistent with $a_m < b_n$ (or $a_m > b_n$ respectively). Thus, merge problem $\lambda M \rho(A, B)$ will reduce to subproblem $\lambda M.(A_1, B_1)$ and $.M\rho(A_2, B_2)$ in case 1, to subproblem $\lambda M/(A_1, B_1)$ and $\backslash M\rho(A_2, B_2)$ in case 2, and to subproblem $\lambda M\backslash(A_1, B_1)$ and $/M\rho(A_2, B_2)$ in case 3. For convenience, we say the adversary adopts a *simple* strategy if he splits the lists in the way of case 1, otherwise the adversary adopts a *complex* strategy (case 2 or 3).

There are obvious symmetries, such as $/M.(m, n) = .M\backslash(m, n) = \backslash M.(n, m) = .M/(n, m)$, $/M/(m, n) = \backslash M\backslash(m, n)$, and $/M\backslash(m, n) = \backslash M/(n, m)$, which means we can deduce the nine functions to four functions: $.M.$, $/M.$, $/M\backslash$, and $/M/$. These functions can be calculated by a computer rather quickly, and the values for all $m, n \leq 150$ and the program are available in [19].

Note $M(m, n) \geq .M.(m, n)$, but $M(m, n)$ is not equal to $.M(m, n)$. in general, since we restrict the power of adversary in the decision tree model by assuming there is a (unknown) division of the lists after each comparison. But this restrict model still covers many interesting cases. For example, when $m \leq n \leq \lfloor \frac{3}{2}m \rfloor + 1$ [6, 24, 28] or $5m - 3 \leq n \leq 7m$ [5], $.M(m,n). = M(m,n)$.

Let $\lambda M_{i,j}\rho(m,n)$ denote the number of comparisons resulted from adversary's best strategy if the first comparison is a_i and b_j, thus $\lambda M\rho(m,n) = min_{i,j}\lambda M_{i,j}\rho(m,n)$.

The following notation is also used in our paper.

▶ **Definition 4.** Let $\overline{.M.}(m,n)$ be the difference of the number of comparisons required by tape merge in the worst case and $.M.(m,n)$, i.e. $\overline{.M.}(m,n) \triangleq m + n - 1 - .M.(m,n)$.

3 Inequalities about Knuth's adversary methods

In this section, we list several inequalities about $\lambda M\rho$, which will be used in Section 4 and Section 5.

▶ **Lemma 5.** *For any $\lambda, \rho \in \{., /, \backslash\}$, we have*
(a) $.M\rho(m,n) \geq \lambda M\rho(m,n)$*;*
(b) $/M\rho(m,n) \leq .M\rho(m,n-1) + 1$.

Proof. *Part (a)* is obvious, the adversary can perform at least as well on less restrictions. In *Part (b)*, if the first comparison is a_1 and b_1 for $/M\rho(m,n)$, the adversary has to claim $a_1 > b_1$, thus it reduces to $.M\rho(m,n-1)$. Therefore we have $/M\rho(m,n) \leq /M_{1,1}\rho(m,n) = 1 + .M\rho(m,n-1)$. ◀

The following lemma shows that if $.M.(m,n) = m + n - 1$, then tape merge is optimal for any (m',n') satisfying $m' \geq m$, $n' \leq n$ and $m' \leq n'$. The proof is in Appendix A.

▶ **Lemma 6.** *For any $m,n \geq 0$, $m+n \geq 1$ and $m \leq n$, we have $.M.(m+1,n) \geq .M.(m,n) + 1 \geq .M.(m,n+1)$ or $\overline{.M.}(m+1,n) \leq \overline{.M.}(m,n) \leq \overline{.M.}(m,n+1)$.*

We can show a similar statement about $/M.$ function as well. The proof is very similar, and we omit it here.

▶ **Lemma 7.** *For any $m,n \geq 1$, we have*
(a) $/M.(m+1,n+1) \geq /M.(m,n) + 2$ *[28];*
and for any $m,n \geq 1$ and $m \leq n$, we have
(b) $/M.(m,n+1) \leq /M.(m,n) + 1$*;*
(c) $/M.(m+1,n) \geq /M.(m,n) + 1$*, except $(m,n) = (1,1), (2,2)$ or $(3,3)$.*

4 Lower bounds for $\alpha(m)$

The key step is to show that $.M.(m+25,n+38) \geq .M.(m,n) + 63$, which directly implies Theorem 1. Since it's unavoidable to show similar statements for other restricted adversaries $\lambda M\rho$, we prove them by induction in parallel. In addition, by symmetry, we have $/M.(m,n) = .M\backslash(m,n)$, $\backslash M.(m,n) = .M/(m,n)$, and $/M/(m,n) = \backslash M\backslash(m,n)$, so the following theorem is enough for our goal.

▶ **Theorem 8.** *For $m,n \geq 0$ and $m+n \geq 1$, we have*
(a) $.M.(m+25,n+38) \geq .M.(m,n) + 63$*;*
and for $m,n \geq 1$, we have
(b) $/M.(m+25,n+38) \geq /M.(m,n) + 63$*;*
(c) $\backslash M.(m+25,n+38) \geq \backslash M.(m,n) + 63$*;*

(d) $/M\backslash(m+25, n+38) \geq /M\backslash(m,n)+63$*;*
(e) $\backslash M\backslash(m+25, n+38) \geq \backslash M\backslash(m,n)+63,$ *except* $(m,n)=(1,1)$*;*
(f) $\backslash M/(m+25, n+38) \geq \backslash M/(m,n)+63,$ *except* $(m,n)=(2,1)$.

Proof. The proof is by induction on m and n. The starting values for $m, n \leq 50$ are given in [19]. Now suppose the theorem holds for any m', n' satisfying $m' \leq m$, $n' \leq n$ and $m'+n' < m+n$, we then prove the case (m,n) where $m \geq 51$ or $n \geq 51$. Recall that our task is to design a strategy for the adversary for $(m+25, n+38)$.

Part (a). Suppose an algorithm begins by comparing a_i and b_j, if $i \leq m$ and $j \geq n+1$, then the adversary claims $a_i < b_j$ and follows the simple strategy, yielding

$$.M_{i,j}.(m+25, n+38) \geq 1 + .M.(m,n) + .M.(25,38) = .M.(m,n)+63.$$

If $i \geq m+1$ and $j \leq n$, the adversary claims $a_i > b_j$ and uses the simple strategy. This leads to

$$.M_{i,j}.(m+25, n+38) \geq 1 + .M.(m,n) + .M.(25,38) = .M.(m,n)+63.$$

If $i \leq m$ and $j \leq n$, assume if we compare a_i and b_j in $.M.(m,n)$, the adversary's best strategy is $1 + .M\rho(p,q) + \lambda M.(s,t)$ where $\lambda, \rho \in \{., /, \backslash\}$, then adversary uses the same strategy here, and we get

$$\begin{aligned}.M_{i,j}.(m+25, n+38) \geq& 1 + .M\rho(p,q) + \lambda M.(s+25, t+38)\\ \geq& 1 + .M\rho(p,q) + \lambda M.(s,t) + 63 \geq .M.(m,n)+63\end{aligned}$$

by using the induction hypothesis.

If $i \geq m+1$ and $j \geq n+1$, then $i \leq 25$ and $j \leq 38$ cannot happen simultaneously, thus there are only three possible cases: $(i \geq 26,\ j \leq 38)$, $(i \leq 25,\ j \geq 39)$, or $(i \geq 26,\ j \geq 39)$. Reversing the order of the elements in A and B maps all these three cases to the above ones, so we can handle these cases as well by symmetry.

Therefore no matter which two elements are chosen to compare at the first step, the adversary can always find a strategy resulting the value not smaller than $.M.(m,n)+63$. This completes the proof of *Part (a)*.

Part (b). The proof for cases where $(i \leq m,\ j \leq n)$, $(i \geq m+1,\ j \leq n)$, and $(i \leq m,\ j \geq n+1)$ is similar with *Part (a)*.

If $i \geq m+1$ and $j \geq n+1$, then $i \leq 25$ and $j \leq 38$ cannot happen simultaneously, so we only need to consider the following cases:

If $i \leq 25$ and $j \geq 39$, or $i \geq 26$ and $j \leq 38$, the adversary uses the simple strategy, yielding

$$/M_{i,j}.(m+25, n+38) \geq 1 + /M.(25,38) + .M.(m,n) \geq 1+62+.M.(m,n) \geq /M.(m,n)+63.$$

If $i \geq 26$ and $j \geq 39$: assume if we compare a_{i-25} and b_{j-38} in $/M.(m,n)$, the adversary's best strategy is $1 + /M\rho(p,q) + \lambda M.(s,t)$. If $(p,q,\rho) \neq (1,1,/)$, the adversary uses the same strategy, and we get

$$\begin{aligned}/M_{i,j}.(m+25, n+38) \geq& 1 + /M\rho(p+25, q+38) + \lambda M.(s,t)\\ \geq& 1 + /M\rho(p,q) + 63 + \lambda M.(s,t) \geq /M.(m,n)+63\end{aligned}$$

by the induction hypothesis. If $(p,q,\rho) = (1,1,/)$, then we know that $i \geq 27$ and $j = 39$, and the adversary can use the simple strategy, yielding

$$/M_{i,j}.(m{+}25,n{+}38) \geq 1{+}/M.(25,39){+}.M.(m,n{-}1) = 1{+}63{+}.M.(m,n{-}1) \geq 63{+}/M.(m,n).$$

The last inequality is due to Lemma 5.

Therefore the adversary can always find a strategy resulting the value not smaller than $/M.(m,n)+63$, no matter what the first comparison is. This completes the proof of *Part (b)*.

Part (c). The proof for cases where $(i \leq m,\ j \leq n)$, $(i \geq m+1,\ j \leq n)$, and $(i \leq m,\ j \geq n+1)$ is similar with *Part (a)*.

Similar with the above argument, if $i \geq m+1$ and $j \geq n+1$, we only need to investigate the following cases:

If $i \leq 25$ and $j \geq 39$, or $i \geq 27$ and $j \leq 38$, the adversary uses the complex strategy with a_{26} in both subproblems. This leads to

$$\backslash M_{i,j}.(m{+}25,n{+}38) \geq 1{+}\backslash M/(26,38){+}\backslash M.(m,n) = 1{+}62{+}\backslash M.(m,n) = \backslash M.(m,n){+}63.$$

If $i = 26$ and $j \leq 38$, since $i \geq m+1$ and $j \geq n+1$, then $m \leq 25$ and $n \leq 37$ and these cases have been checked as starting values.

If $i \geq 26$ and $j \geq 39$: assume if we compare a_{i-25} and b_{j-38} in $\backslash M.(m,n)$, the adversary's best strategy is $1 + \backslash M\rho(p,q) + \lambda M.(s,t)$. If $(p,q,\rho) \neq (1,1,\backslash)$ or $(2,1,/)$, the adversary uses the same strategy, yielding

$$\begin{aligned}\backslash M_{i,j}.(m+25,n+38) \geq& 1 + \backslash M\rho(p+25,q+38) + \lambda M.(s,t)\\ \geq& 1 + \backslash M\rho(p,q) + 63 + \lambda M.(s,t) \geq \backslash M.(m,n) + 63\end{aligned}$$

by the induction hypothesis. If $(p,q,\rho) = (1,1,\backslash)$, we have $i = 26$ and $j \geq 40$. The adversary claims $a_i < b_j$ and follows the simple strategy, yielding

$$\backslash M_{i,j}.(m{+}25,n{+}38) \geq 1{+}\backslash M.(26,38){+}.M.(m{-}1,n) = 1{+}63{+}.M.(m{-}1,n) \geq 63{+}\backslash M.(m,n)$$

by using Lemma 5. If $(p,q,\rho) = (2,1,/)$, we have $i \geq 28$ and $j = 39$ or $i = 26$ and $j > 39$, since the case where $i = 26$ and $j > 39$ has already been considered, we only need to investigate the case where $i \geq 28$ and $j = 39$. Notice that $j \geq n+1$, i.e. $n \leq 38$, hence $m \geq 50$. If $i > 28$, the adversary claims $a_i > b_j$ and follows the complex strategy with a_{28} in both subproblems. This leads to

$$\begin{aligned}\backslash M_{i,j}.(m+25,n+38) &\geq 1 + \backslash M/(28,39) + \backslash M.(m-2,n-1)\\ &= 1 + 66 + \backslash M.(m-2,n-1)\\ &\geq 66 + \backslash M.(m-1,n-1)\\ &\geq 63 + 1 + \backslash M/(2,1) + \backslash M.(m-1,n-1)\\ &= 63 + \backslash M_{i-25,1}.(m,n)\\ &\geq \backslash M.(m,n) + 63.\end{aligned}$$

The second inequality is according to Lemma 7 and the second equality is the assumption of the best strategy for the adversary. If $i = 28$ and $j = 39$, we have $m \leq 27$ and $n \leq 38$, which have been checked as starting values.

Therefore the adversary can always find a strategy resulting the value not smaller than $\backslash M.(m,n) + 63$. This completes the proof of *Part (c)*.

Part (d), (e), (f). Due to the space constraint, we put the proofs of Part (d), Part (e) and Part (f) in Appendix B. ◀

Now, we are ready to prove Theorem 1.

Proof. The small cases $1 \leq m \leq 25$ and $1 \leq n \leq 38$ are given in [19]. Given any pair (m, n) satisfying $m \leq n \leq \frac{38}{25}m$, let $m = 25p + s$, $n = 38q + t$ where $0 < s \leq 25$, $0 < t \leq 38$, and observe that $m \geq 25q + \lceil \frac{25}{38}t \rceil$, thus

$$\overline{.M.}(m, n) = \overline{.M.}(25p + s, 38q + t) \leq \overline{.M.}(25q + \lceil \tfrac{25}{38}t \rceil, 38q + t) \leq \overline{.M.}(\lceil \tfrac{25}{38}t \rceil, t) = 0.$$

The first inequality is due to Lemma 6 and the second one is due to Theorem 8. ◀

5 Limitations of Knuth's adversary methods

In this section, we prove Theorem 2, which shows Knuth's adversary methods can not provide lower bounds beyond $\alpha(m) \geq 9\lceil m/5 \rceil$. Actually, we prove a stronger result:

▶ **Theorem 9.** *$.M.(5k, 9k + 12t) \leq 14k + 11t - 2$, for $k, t \geq 0$ and $t + k \geq 1$.*

With this theorem, Theorem 2 is obvious, since if $n \geq 9\lceil m/5 \rceil$, $\overline{.M.}(m, n) \geq \overline{.M.}(m, 9\lceil m/5 \rceil) \geq 1$.

Proof. The proof is by induction on k and t. We verify the case $k \leq 10$ first: when $t \geq k/10 + 2/5$, we have

$$.M.(5k, 9k + 12t) \leq M(5k, 10k + 12t - k) \leq M_{bm}(5k, 10k + 12t - k) \leq 14k + 11t - 2.$$

When $t < k/10 + 2/5$, these finite cases can be checked in [19].

Now suppose $k \geq 11$ and we have already proven this theorem for any (k', t') satisfying $k' < k$, or $k' = k$ and $t' < t$. Since $.M.(m, n) = min_{i,j}.M_{i,j}.(m, n) \leq .M_{50,79}.(m, n)$, thus it's enough to show $.M_{50,79}.(5k, 9k + 12t) \leq 14k + 11t - 2$. In other word, an algorithm which begins by comparing a_{50} with b_{79} can "beat" the adversary. We'll prove it by enumerating the adversary's best strategy.

Case(a). The adversary claims $a_{50} < b_{79}$ and follows three possible strategies.

(i) The adversary uses the simple strategy, then

$$.M_{50,79}.(5k, 9k + 12t) = 1 + .M.(50 + x, 78 - y) + .M.(5k - 50 - x, 9k + 12t - 78 + y),$$

where $x, y \geq 0$. Thus it's sufficient to show

$$\overline{.M.}(50 + x, 78 - y) + \overline{.M.}(5k - 50 - x, 9k + 12t - 78 + y) \geq \overline{.M.}(5k - 50, 9k - 78 + 12t) \geq t + 2.$$

The first inequality is according to Lemma 6 and the second one is by the induction hypothesis.

(ii) The adversary uses the complex strategy, with a_{51+x} in both subproblems.

$$\begin{aligned} &.M_{50,79}.(5k, 9k + 12t) \\ &\quad = 1 + .M/(51 + x, 78 - y) + \backslash M.(5k - 50 - x, 9k + 12t - 78 + y) \\ &\quad \leq 1 + .M.(51 + x, 78 - y) + .M.(5k - 50 - x, 9k + 12t - 78 + y), \end{aligned}$$

where $x, y \geq 0$. Thus it's equivalent to show

$$\begin{aligned} &\overline{.M.}(51 + x, 78 - y) + \overline{.M.}(5k - 50, 9k + 12t - 78 + y) - 1 \\ &\qquad \geq \overline{.M.}(5k - 50, 9k - 78 + 12t) - 1 \geq t + 1. \end{aligned}$$

(iii) The adversary uses the complex strategy with b_{78-y} in both subproblems.

$$\begin{aligned}
&.M_{50,79}.(5k{,}9k + 12t)\\
&\qquad = 1 + .M\backslash(50 + x, 78 - y) + /M.(5k - 50 - x, 9k + 12t - 77 + y)\\
&\qquad \leq 1 + .M.(50 + x, 78 - y) + .M.(5k - 50 - x, 9k + 12t - 77 + y),
\end{aligned}$$

where $x, y \geq 0$. Thus it's equivalent to show

$$\begin{aligned}
&\overline{.M.}(50 + x, 78 - y) + \overline{.M.}(5k - 50 - x, 9k + 12t - 77 + y) - 1\\
&\qquad \geq \overline{.M.}(5k - 50, 9k - 78 + 12t) - 1 \geq t + 1.
\end{aligned}$$

Case(b). The adversary claims $a_{50} > b_{79}$ and follows three possible strategies.

(i) The adversary uses the simple strategy, then

$$.M_{50,79}.(5k, 9k+12t) = 1+.M.(49-x, 79+y)+.M.(5k-50+1+x, 9k+12t-79-y),$$

where $x, y \geq 0$. Thus it's sufficient to show

$$\overline{.M.}(49 - x, 79 + y) + \overline{.M.}(5k - 50 + 1 + x, 9k + 12t - 79 - y) \geq t + 2.$$

Let $5p \leq x \leq 5p+4$ and $12q-10 \leq y \leq 12q+1$, then we claim that $\overline{.M.}(49-x, 79+y) \geq p + q + 1$. If $q \leq 2$, these finite cases can be checked in [19]. Otherwise ($q \geq 3$), then $79+y > 2 \times (49-5p)$, and $.M.(49-5p, 81+12(q-1)) \leq M_{bm}(49-5p, 81+12(q-1)) \leq 127+11(q-1)-6p$. Therefore $\overline{.M.}(49-x, 79+y) \geq \overline{.M.}(49-5p, 81+12(q-1)) \geq 1+p+q$ due to Lemma 6.
Since $\overline{.M.}(49-x, 79+y) \geq p+q+1$, if $p+q \geq t+1$, we've done. If $p+q \leq t$, according to Lemma 6 and the induction hypothesis, we have

$$\begin{aligned}
&\overline{.M.}(5k - 50 + 1 + x, 9k + 12t - 79 - y)\\
&\qquad \geq \overline{.M.}(5k - 50 + 5p + 5, 9k - 90 + 9p + 9 + 12(t - q - p))\\
&\qquad \geq t - p - q + 1.
\end{aligned}$$

Thus $\overline{.M.}(49 - x, 79 + y) + \overline{.M.}(5k - 50 + 1 + x, 9k + 12t - 79 - y) \geq t + 2$.

(ii) The adversary uses the complex strategy with a_{49-x} in both subproblems, then

$$\begin{aligned}
&.M_{50,79}.(5k, 9k + 12t)\\
&\qquad = 1 + .M/(49 - x, 79 + y) + \backslash M.(5k - 50 + 2 + x, 9k + 12t - 79 - y)\\
&\qquad \leq 2 + .M.(49 - x, 79 + y) + 1 + .M.(5k - 50 + 1 + x, 9k + 12t - 79 - y)\\
&\qquad \leq 14k + 11t - 2,
\end{aligned}$$

where $x, y \geq 0$.

(iii) The adversary uses the complex strategy with b_{80+y} in both subproblems, then

$$\begin{aligned}
&.M_{50,79}.(5k, 9k + 12t)\\
&\qquad = 1 + .M\backslash(49 - x, 80 + y) + /M.(5k - 50 + 1 + x, 9k + 12t - 79 - y)\\
&\qquad \leq 2 + .M.(49 - x, 79 + y) + .M.(5k - 50 + 1 + x, 9k + 12t - 79 - y)\\
&\qquad \leq 14k + 11t - 2,
\end{aligned}$$

where $x, y \geq 0$.

◀

6 Upper bounds for $\alpha(m)$

In this section, we give better upper bounds for $\alpha(m)$ by proposing a simple procedure. This procedure only involves the first two elements in each A and B and can be viewed as a modification of binary merge.

Algorithm 1 Modified Binary Merge

```
Compare $a_1$ and $b_2$
if $a_1 > b_2$ then
  merge $(m, n-2)$.
else
  compare $a_2$ and $b_2$
  if $a_2 > b_2$ then
    compare $a_1$ and $b_1$, then merge $(m-1, n-2)$.
  else
    compare $a_2$ and $b_1$
    if $a_2 > b_1$ then
      compare $a_1$ and $b_1$, then merge $(m-2, n-1)$ .
    else
      merge $(m-2, n)$.
    end if
  end if
end if
```

It is easy to see that this procedure induces the following recurrence relation:

$$M(m,n) \le \max\{M(m,n-2)+1, M(m-1,n-2)+3, M(m-2,n)+3, M(m-2,n-1)+4\}.$$

In the following, we'll use the induction to give better upper bounds for $n \in [2m-2, 3m]$. The following proofs are very similar, but we give all the details for sake of completeness.

▶ **Theorem 10.** *$M(m, 2m+2k) \le 3m+k-2$, for $m \ge 3$ and $k \ge -1$.*

Proof. We induce on k and m. The case for $k = -1$ just follows tape merge algorithm. The case for $m = 3$ are given by Hwang [15] and Murphy [25].

Now suppose that $m \ge 4$ and $k \ge 0$, and the claim has already been proven for any (m', k') satisfying $m' + k' \le m + k - 1$. According to the procedure, we have

$$\begin{aligned} &M(m{,}2m+2k) \\ &\quad \le \max\{M(m, 2m+2(k-1))+1, M(m-1, 2m+2k-2)+3, \\ &\qquad\qquad M(m-2, 2m+2k)+3, M(m-2, 2m+2k-1)+4\} \end{aligned}$$

$\le \max\{3m+k-2$ (the induction hypothesis), $3(m-1)+k-2+3$ (the induction hypothesis), $3(m-2)+1+k+3$ (binary merge), $3(m-2)+k+4$ (binary merge)$\} \le 3m+k-2$. ◀

▶ **Theorem 11.** *$M(m, 2m+2k-1) \le 3m+k-3$, for $m \ge 5$ and $k \ge -1$.*

Proof. We induce on k and m. The case for $k = -1$ just follows tape merge algorithm. The case for $m = 5$ are given by Mönting [26].

Now suppose that $m \geq 6$ and $k \geq 0$, and the claim has already been proven for any (m', k') satisfying $m' + k' \leq m + k - 1$. According to the procedure, we have

$$\begin{aligned}&M(m,2m+2k-1)\\&\quad\leq \max\{M(m,2m+2(k-1)-1)+1, M(m-1,2m+2k-1-2)+3,\\&\qquad\qquad M(m-2,2m+2k-1)+3, M(m-2,2m+2k-2)+4\}\end{aligned}$$

$\leq \max\{3m+k-3$ (the induction hypothesis), $3(m-1)+k-3+3$ (the induction hypothesis), $3(m-2)+k+3$ (binary merge), $3(m-2)+k-1+4$ (Theorem 10)$\} \leq 3m+k-3$. ◀

▶ **Theorem 12.** *$M(m, 2m-2) \leq 3m-4$, for $m \geq 7$.*

Proof. We induce on m. The case for $m = 7$ has been verified by Smith and Lang[27]. Now suppose that $m \geq 8$ and the claim has already been proven for $m - 1$. According to S, we have

$$\begin{aligned}&M(m,2m-2)\\&\quad\leq \max\{M(m,2m-4)+1, M(m-1,2m-4)+3,\\&\qquad\qquad M(m-2,2m-2)+3, M(m-2,2m-3)+4\}\end{aligned}$$

$\leq \max\{3m-4$(tape merge), $3(m-1)-4+3$ (the induction hypothesis), $3(m-2)-1+3$ (Theorem 10), $3(m-2)-2+4$ (Theorem 11)$\} \leq 3m-4$. ◀

▶ **Theorem 13.** *$M(m, 2m) \leq 3m-3$, for $m \geq 10$.*

Proof. We do the induction on m. Smith and Lang[27] have verified the case for $m = 10$. Now suppose that $m \geq 11$ and the claim has already been proven for $m - 1$. According to the procedure, we have

$$\begin{aligned}&M(m,2m)\\&\quad\leq \max\{M(m,2m-2)+1, M(m-1,2m-2)+3,\\&\qquad\qquad M(m-2,2m)+3, M(m-2,2m-1)+4\}\end{aligned}$$

$\leq \max\{3m-3$ (Theorem 12), $3(m-1)-3+3$ (the induction hypothesis), $3(m-2)+3$ (Theorem 10), $3(m-2)-1+4$ (Theorem 11)$\} \leq 3m-3$. ◀

Finally, we put together the above theorems to get Theorem 3.

As we can see in the proofs, if better basic cases can be provided, we can get better upper bounds by using this procedure. However, there is a barrier of this approach: if we want to show $\alpha(m) \leq 2m - k$ or $M(m, 2m - k + 1) < 3m - k$, it's necessary to obtain $\alpha(m-1) \leq 2(m-1) - k$ or $M(m-1, 2m-2-k+1) < 3m-3-k$ at first, thus it is impossible to show $\alpha(m) \leq 2m - \omega(1)$ via this approach.

7 Conclusion

In this paper we improve the lower bounds for $\alpha(m)$ from $\lfloor \frac{3}{2}m \rfloor + 1$ to $\lfloor \frac{38}{25}m \rfloor$ via Knuth's adversary methods. We also show that it is impossible to get $\alpha(m) \geq 9\lceil m/5 \rceil \approx \frac{9}{5}m$ for any m by using this methods. We then design an algorithm which saves at least one comparison compared to binary merge for $2m - 2 \leq n \leq 3m$. Specially, for the case $M(m, 2m-2)$, our algorithm uses one comparison less than tape merge or binary merge, which means we can improve the upper bounds of $\alpha(m)$ by 1. We wonder whether there exists a universal efficient algorithm to give significantly better upper bounds for $M(m,n)$ in the case $n \leq 2m$, or maybe it's intrinsically hard to compute M functions since there doesn't exist general patterns or underlying structures in the corresponding decision trees.

Besides that, we are also curious about the following conjectures proposed by Knuth [18]:

▶ **Conjecture 14.** *$M(m+1,n+1) \geq 2 + M(m,n)$.*

Via a similar proof with Lemma 7, the above conjecture implies the following conjecture which has been mentioned in Section 1.

▶ **Conjecture 15.** *$M(m+1,n) \geq 1 + M(m,n) \geq M(m,n+1)$, for $m \leq n$.*

In the attempt to prove these two conjectures, we introduced the notation $.M^{(k)}.(m,n)$. Roughly speaking, $.M^{k}.(m,n)$ is the adversary which can delay k steps to give the splitting strategy, and $.M.(m,n) = .M^{0}.(m,n) \leq .M^{1}.(m,n) \cdots \leq .M^{m+n-2}.(m,n) = M(m,n)$. In the case $k = 0$, it is exactly Lemma 16 in appendix A, but it becomes much harder even for $k = 1$.

References

1 Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and selection with imprecise comparisons. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP'09, pages 37–48, 2009.

2 Mark R. Brown and Robert E. Tarjan. A fast merging algorithm. *J. ACM*, 26(2):211–226, 1979.

3 Jean Cardinal and Samuel Fiorini. *On Generalized Comparison-Based Sorting Problems*, pages 164–175. Springer Berlin Heidelberg, 2013.

4 Yongxi Cheng, Xiaoming Sun, and Yiqun Lisa Yin. Searching monotone multi-dimensional arrays. *Discrete Mathematics*, 308(11):2213–2221, 2008.

5 C. Christen. Improving the bounds on optimal merging. In *Proc 19th IEEE conf on the foundations of computer science*, pages 259–266, 1978.

6 C. Christen. On the optimality of the straight merging algorithm, 1978.

7 W. Fernandez de la Vega, M. A. Frieze, and M. Santha. Average-case analysis of the merging algorithm of hwang and lin. *Algorithmica*, 22(4):483–489, 1998.

8 Wenceslas Fernandez de la Vega, Sampath Kannan, and Miklos Santha. Two probabilistic results on merging. *SIAM Journal on Computing*, 22(2):261–271, 1993.

9 Fănică Gavril. Merging with parallel processors. *Commun. ACM*, 18(10):588–591, 1975.

10 A. Gupta and A. Kumar. Sorting and selection with structured costs. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, page 416, 2001.

11 Bing-Chao Huang and Michael A. Langston. Practical in-place merging. *Commun. ACM*, 31(3):348–352, 1988.

12 Z. Huang, S. Kannan, and S. Khanna. Algorithms for the generalized sorting problem. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science*, pages 738–747, 2011.

13 F. K. Hwang and D. N. Deutsch. A class of merging algorithms. *J. ACM*, 20(1):148–159, 1973.

14 Frank Hwang and Shen Lin. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM Journal on Computing*, 1(1):31–39, 1972.

15 Frank K. Hwang. Optimal merging of 3 elements with n elements. *SIAM Journal on Computing*, 9(2):298–320, 1980.

16 Frank K. Hwang and Shen Lin. Optimal merging of 2 elements with n elements. *Acta Informatica*, 1(2):145–158, 1971.

17 Sampath Kannan and Sanjeev Khanna. Selection with monotone comparison costs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'03, pages 10–17, 2003.

18 Donald Knuth. The art of computer programming. *Sorting and searching*, 3:197–207, 1999.

19 Qian Li, Xiaoming Sun, and Jialin Zhang. Tables of λmρ and the program, 2016. URL: http://theory.ict.ac.cn/liqian.

20 N. Linial and M. Saks. Searching ordered structures. *Journal of Algorithms*, 6(1):86–103, 1985.

21 Nathan Linial. The information-theoretic bound is good for merging. *SIAM Journal on Computing*, 13(4):795–801, 1984.

22 G. K. Manacher, T. D. Bui, and T. Mai. Optimum combinations of sorting and merging. *J. ACM*, 36(2):290–334, 1989.

23 Glenn K. Manacher. Significant improvements to the hwang-lin merging algorithm. *J. ACM*, 26(3):434–440, 1979.

24 Paul E. Murphy and Marvin C. Paull. Minimum comparison merging of sets of approximately equal size. *Information and Control*, 42(1):87–96, 1979.

25 P. E. Murphy. A problem in optimal meging. Technical Report DCS-TR-69, Rutgers University, Dept. of Computer Sciences, 1978.

26 Jürgen Schulte Mönting. Merging of 4 or 5 elements with n elements. *Theoretical Computer Science*, 14(1):19–37, 1981.

27 Warren D. Smith and Kevin J. Lang. Values of the merging function and algorithm design as a game, 1994.

28 Paul K. Stockmeyer and F. Frances Yao. On the optimality of linear merge. *SIAM Journal on Computing*, 9(1):85–90, 1980.

29 R. Michael. Tanner. Minimean merging and sorting: An algorithm. *SIAM Journal on Computing*, 7(1):18–38, 1978.

30 Mai Thanh, V. S. Alagar, and T. D. Bui. Optimal expected-time algorithms for merging. *Journal of Algorithms*, 7(3):341–357, 1986.

31 Mai Thanh and T. D. Bui. An improvement of the binary merge algorithm. *BIT Numerical Mathematics*, 22(4):454–462, 1982.

A Proof of Lemma 6

We show the following lemma first:

▶ **Lemma 16.** $.M.(m+1,n+1) \geq .M.(m,n)+2$, *for any* $m,n \geq 0$ *and* $m+n \geq 1$.

Proof. The proof is by induction on m and n. The starting values for $m,n \leq 3$ can be easily checked in [18]. Now suppose the theorem holds for any m',n' satisfying $m' \leq m$, $n' \leq n$ and $m'+n' < m+n$, we then prove the case (m,n). Note that our task is to design a strategy for the adversary for $(m+1,n+1)$.

Suppose an algorithm begins by comparing a_i and b_j, where $i \leq m$, $j = n+1$. The adversary claims that $a_i < b_j$, and follows the simple strategy, yielding

$$.M_{i,j}.(m+1,n+1) \geq 1 + .M.(m,n) + .M.(1,1) \geq .M.(m,n)+2\,.$$

If $i = m+1$ and $j \leq n$, the adversary claims that $a_i > b_j$, and uses the simple strategy. This leads to

$$.M_{i,j}.(m+1,n+1) \geq 1 + .M.(m,n) + .M.(1,1) \geq .M.(m,n)+2\,.$$

If $i \le m$ and $j \le n$, assume that if we compare a_i and b_j in $.M.(m,n)$, the adversary's best strategy is $1 + .M\rho(p,q) + \lambda M.(s,t)$, where $\rho, \lambda \in \{., /, \backslash\}$ and $s+t \ge 1$ if $\lambda = .$ and $s,t \ge 1$ if $\lambda = \{/, \backslash\}$. Then the adversary uses the same strategy here, and we get

$$\begin{aligned} .M_{i,j}.(m+1,n+1) &\ge 1 + .M\rho(p,q) + \lambda M.(s+1,t+1) \\ &\ge 1 + .M\rho(p,q) + \lambda M.(s,t) + 2 = .M.(m,n) + 2 \end{aligned}$$

by applying the induction hypothesis and Lemma 7.

If $i = m+1$ and $j = n+1$, we can handle this case as well by reversing the order of the elements in A and B.

Therefore adversary can always find a strategy which is not smaller than $.M.(m,n)+2$, no matter what the first comparison is. This completes our proof. ◀

Now we can give the proof of Lemma 6

Proof. We induce on m and n. The case for $1 \le m+n \le 10$ are given in [18]. Now suppose that $m+n \ge 11$ and the lemma is already established for any (m', n') satisfying $m' + n' < m+n$.

Part (a).

If $m = 1$, we have $.M.(1, n+1) \le .M_{1,n+1}.(1,n+1) = \max\{1, 1+.M.(1,n)\} = 1+.M.(1,n)$. If $m \ge 2$, then $.M_{m,n+1}.(m, n+1) = \max\{.M.(m-1,n+1), .M.(m,n)\} + 1$, and by the induction we know $.M.(m,n) \ge .M.(m-1,n) + 1 \ge .M.(m-1,n+1)$, thus

$$.M.(m,n+1) \le .M_{m,n+1}.(m,n+1) \le \max\{.M.(m-1,n+1), .M.(m,n)\} + 1 = .M.(m,n)+1.$$

Part (b).

When $m = n$, $.M.(m+1,m) \ge .M.(m,m-1) + 2 \ge .M.(m-1,m-2) + 4 \ge \cdots \ge .M.(2,1) + 2m - 2 = 2m$ according to Lemma 16, thus $.M.(m+1,m) \ge .M.(m,m)+1$, since $.M.(m,m) \le 2m-1$. When $m < n$, we have

$$.M.(m+1,n) \ge .M.(m,n-1) + 2 \ge .M.(m,n) + 1.$$

The first inequality is due to Lemma 16, and the second one is by the induction hypothesis. ◀

B Proof of Theorem 8

Proof.

Part (d). The proof for cases where $i \le 25$ and $j > 38$, $i > 25$ and $j \le 38$, and $i > 25$ and $j > 38$ is similar with *Part (b).*

If $i \le 25$ and $j \le 38$, reserving the order of the elements maps this case to the above cases, thus we can handle this case as well by symmetry.

Therefore the adversary can always find a strategy which is not smaller than $/M\backslash(m,n) + 63$, no matter which the first comparison is. So *Part (d)* is true.

Part (e). The adversary's strategies for cases where $(i \le m,\ j \le n)$, $(i > m,\ j \le n)$, and $(i \le m,\ j > n)$ are similar with *Part (d).*

If $i \ge m+1$ and $j \ge n+1$, then $i \le 25$ and $j \le 38$ cannot happen simultaneously, so we only need to investigate the following cases:

If $i \le 25$ and $j > 38$, or $i \ge 27$ and $j \le 38$, the adversary uses the complex strategy with a_{26} in both subproblems. This leads to

$$\backslash M_{i,j}\backslash(m+25, n+38) \ge 1 + \backslash M/(26,38) + \backslash M\backslash(m,n) = \backslash M\backslash(m,n) + 63.$$

If $i = 26$ and $j \leq 38$, then $m \leq 25$ and $n \leq 38$, which have been checked as starting values.

If $i \geq 26$ and $j \geq 39$: assume if we compare a_{i-25} and b_{j-38} in $\backslash M\backslash(m, n)$, the adversary's best strategy is $1 + \backslash M\rho(p, q) + \lambda M\backslash(s, t)$. If $(p, q, \rho) \neq (1, 1, \backslash)$ or $(2, 1, /)$, the adversary uses the same strategy, yielding

$$\backslash M_{i,j}\backslash(m + 25, n + 38) \geq 1 + \backslash M\rho(p + 25, q + 38) + \lambda M\backslash(s, t) \geq \backslash M\backslash(m, n) + 63$$

by the induction hypothesis. If $(p, q, \rho) = (1, 1, \backslash)$, we have $i = 26$ and $j > 39$. The adversary claims $a_i < b_j$ and uses the simple strategy, leading to

$$\backslash M_{i,j}\backslash(m{+}25, n{+}38) \geq 1{+}\backslash M.(26, 38){+}.M\backslash(m{-}1, n) = 64{+}.M\backslash(m{-}1, n) \geq 63{+}\backslash M\backslash(m, n)$$

by using Lemma 5.

If $(p, q, \rho) = (2, 1, /)$, the case where $i \geq 28$ and $j = 39$ is the only unconsidered case. If we compare a_2 with b_1 in $\backslash M\backslash(m, n)$, and the adversary claims $a_2 > b_1$, then the best strategy must be $1 + \backslash M.(1, 1) + .M\backslash(m - 1, n - 1)$, otherwise the adversary claims $a_2 < b_1$, and the best strategy must be $1 + .M\backslash(m - 2, n)$. So we get $\backslash M\backslash(m, n) \leq \backslash M_{2,1}\backslash(m, n) = \max\{1{+}.M\backslash(m{-}2, n), 2{+}.M\backslash(m{-}1, n{-}1)\}$. If $.M\backslash(m{-}2, n) \geq 1{+}.M\backslash(m{-}1, n{-}1)$, then $.M\backslash(m - 2, n) + 1 \geq \backslash M\backslash(m, n)$, and the adversary splits the problem $\backslash M\backslash(m + 25, n + 38)$ into two independent subproblems $\backslash M.(27, 38)$ and $.M\backslash(m - 2, n)$ before the algorithm begins, and this leads to

$$\backslash M\backslash(m + 25, n + 38) \geq \backslash M.(27, 38) + .M\backslash(m - 2, n) \geq 63 + \backslash M\backslash(m, n)$$

Otherwise $(.M\backslash(m - 2, n) < 1 + .M\backslash(m - 1, n - 1))$, then $2 + .M\backslash(m - 1, n - 1) \geq \backslash M\backslash(m, n)$ and the adversary claims $a_i > b_j$ and uses the simple strategy, yielding

$$\backslash M\backslash(m + 25, n + 38) \geq \backslash M.(26, 39) + .M\backslash(m - 1, n - 1) + 1 \geq \backslash M\backslash(m, n) + 63.$$

Therefore the adversary can always find a strategy resulting the value not smaller than $\backslash M\backslash(m, n) + 63$. This completes the proof of Part (e).

Part (f). If $n > 50$, we can assume $j \geq \lfloor \frac{38+n}{2} \rfloor \geq 44$ by symmetry. If $i \leq 25$, the adversary claims $a_i < b_j$ and uses the complex strategy with a_{26} in both subproblems. This leads to

$$\backslash M/(m + 25, n + 38) \geq 1 + \backslash M/(26, 38) + \backslash M/(m, n) = 63 + \backslash M/(m, n).$$

If $i \geq 26$: assume that if we compare a_{i-25} and b_{j-38} in $\backslash M\backslash(m, n)$, the adversary's best strategy is $1 + \backslash M\rho(p, q) + \lambda M/(s, t)$. If $(p, q, \rho) \neq (1, 1, \backslash), (2, 1, /)$, adversary uses the same strategy, yielding

$$\backslash M_{i,j}/(m + 25, n + 38) \geq 1 + \backslash M\rho(p + 25, q + 38) + \lambda M/(s, t) \geq \backslash M/(m, n) + 63$$

by using the induction hypothesis. If $(p, q, \rho) = (1, 1, \backslash)$, we have $i = 26$ and $j \geq 40$. The adversary claims $a_i < b_j$ and use the simple strategy, yielding

$$\backslash M_{i,j}/(m{+}25, n{+}38) \geq 1{+}\backslash M.(26, 28){+}.M/(m{-}1, n) = 64{+}.M/(m{-}1, n) \geq 63{+}\backslash M/(m, n)$$

by using Lemma 5. If $(p, q, \rho) = (2, 1, /)$, we have $i \geq 28$ and $j = 39$, violating the assumption $j \geq 44$.

If $n \leq 50$, then $m > 50$ and we can assume $i \geq \lfloor \frac{25+m}{2} \rfloor \geq 38$ by symmetry. If $j \leq 38$, the adversary claims $a_i > b_j$, and uses the complex strategy with a_{26} in both subproblems, yielding

$$\backslash M/(m+25, n+38) \geq 1 + \backslash M/(26, 38) + \backslash M/(m, n) \geq 63 + \backslash M/(m, n).$$

If $j \geq 39$: assume if we compare a_{i-25} and b_{j-38} in $\backslash M\backslash(m, n)$, the adversary's best strategy is $1 + \backslash M\rho(p, q) + \lambda M/(s, t)$. If $(p, q, \rho) \neq (1, 1, \backslash)$ or $(2, 1, /)$, then adversary uses the same strategy, thus

$$\backslash M_{i,j}/(m+25, n+38) \geq 1 + \backslash M\rho(p+25, q+38) + \lambda M/(s, t) \geq \backslash M/(m, n) + 63$$

by using the induction hypothesis. If $(p, q, \rho) = (2, 1, /)$, we have $j = 39$. Similar with the argument in Part (e), we get $\backslash M/(m, n) \leq \backslash M_{2,1}/(m, n) = \max\{1 + .M/(m-2, n), 2 + .M/(m-1, n-1)\}$. If $.M/(m-2, n) \geq 1 + .M/(m-1, n-1)$, the adversary splits the problem $\backslash M/(m+25, n+38)$ into two independent subproblems $\backslash M.(27, 38)$ and $.M/(m-2, n)$ before the first comparison begins, and this leads to

$$\backslash M/(m+25, n+38) \geq \backslash M.(27, 38) + .M/(m-2, n) \geq 63 + \backslash M/(m, n).$$

Otherwise ($.M/(m-2, n) < 1 + .M/(m-1, n-1)$), then the adversary claims $a_i > b_j$, and uses the simple strategy, yielding

$$\begin{aligned} \backslash M/(m+25, n+38) \quad &\geq \backslash M.(26, 39) + .M/(m-1, n-1) + 1 \geq 65 + .M/(m-1, n-1) \\ &\geq \backslash M/(m, n) + 63 \end{aligned}$$

If $(p, q, \rho) = (1, 1, \backslash)$, then $i = 26$, violating the assumption $i \geq 38$.

Therefore the adversary can always find a strategy resulting the value not smaller than $\backslash M/(m, n) + 63$. This completes the proof of Part (f). ◀

Dispersing Points on Intervals[*][†]

Shimin Li[1] **and Haitao Wang**[2]

1 **Department of Computer Science, Utah State University, Logan, UT 84322, USA**
shiminli@aggiemail.usu.edu

2 **Department of Computer Science, Utah State University, Logan, UT 84322, USA**
haitao.wang@usu.edu

Abstract

We consider a problem of dispersing points on disjoint intervals on a line. Given n pairwise disjoint intervals sorted on a line, we want to find a point in each interval such that the minimum pairwise distance of these points is maximized. Based on a greedy strategy, we present a linear time algorithm for the problem. Further, we also solve in linear time the cycle version of the problem where the intervals are given on a cycle.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.1.2 Algorithms, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases dispersing points, intervals, min-max, algorithms, cycles

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.52

1 Introduction

The problems of dispersing points have been extensively studied and can be classified to different categories by their different constraints and objectives, e.g., [6, 10, 13, 14, 15, 19].

In this paper, we consider problems of dispersing points on intervals in linear domains including lines and cycles. Let $\mathcal{I}$ be a set of n intervals on a line ℓ, and no two intervals of $\mathcal{I}$ intersect. The problem is to find a point in each interval of $\mathcal{I}$ such that the minimum distance of any pair of points is maximized. We assume the intervals of $\mathcal{I}$ are given sorted on ℓ. In this paper we present an $O(n)$ time algorithm for the problem.

We also consider the *cycle version* of the problem where the intervals of $\mathcal{I}$ are given on a cycle $\mathcal{C}$. The intervals of $\mathcal{I}$ are also pairwise disjoint and are given sorted cyclically on $\mathcal{C}$. Note that the distance of two points on $\mathcal{C}$ is the length of the shorter arc of $\mathcal{C}$ between the two points. By making use of our "line version" algorithm, we solve this cycle version problem in linear time as well.

1.1 Related Work

To the best of our knowledge, we have not found any previous work on the two problems studied in this paper. Our problems essentially belong to a family of geometric dispersion problems, which are NP-hard in general in two and higher dimensional space. For example, Baur and Fekete [1] studied the problems of distributing a number of points within a polygonal

[*] A full version of the paper is available at https://arxiv.org/abs/1611.09485.

[†] This research was supported in part by NSF under Grant CCF-1317143.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 52; pp. 52:1–52:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

region such that the points are dispersed far away from each other, and they showed that the problems cannot be approximated arbitrarily well in polynomial time, unless P=NP.

Wang and Kuo [19] considered the following two problems. Given a set S of points and a value d, find a largest subset of S in which the distance of any two points is at least d. Given a set S of points and an integer k, find a subset of k points of S to maximize the minimum distance of all pairs of points in the subset. It was shown in [19] that both problems in 2D are NP-hard but can be solved efficiently in 1D. Refer to [2, 5, 7, 8, 12] for other geometric dispersion problems. Dispersion problems in various non-geometric settings were also considered [6, 10, 13, 14, 15]. These problems are in general NP-hard; approximation and heuristic algorithms were proposed for them.

On the other hand, problems on intervals usually have many applications. For example, some problems on intervals are related to scheduling because the time period between the release time and the deadline of a job or task in scheduling problems can be considered as an interval on the line. From the interval point of view, Garey et al. [9] studied the following problem on intervals: Given n intervals on a line, determine whether it is possible to find a unit-length sub-interval in each input interval, such that these sub-intervals do not intersect. An $O(n \log n)$ time algorithm was given in [9] for this problem. The optimization version of the above problem was also studied [4, 17], where the goal is to find a maximum number of intervals that contain non-intersecting unit-length sub-intervals. Chrobak et al. [4] gave an $O(n^5)$ time algorithm for the problem, and later Vakhania [17] improved the algorithm to $O(n^2 \log n)$ time. The online version of the problem was also considered [3]. Other optimization problems on intervals have also been considered, e.g., see [9, 11, 16, 18].

1.2 Our Approaches

For the line version of the problem, our algorithm is based on a greedy strategy. We consider the intervals of $\mathcal{I}$ incrementally from left to right, and for each interval, we will "temporarily" determine a point in the interval. During the algorithm, we maintain a value $d_{\min}$, which is the minimum pairwise distance of the "temporary" points that so far have been computed. Initially, we put a point at the left endpoint of the first interval and set $d_{\min} = \infty$. During the algorithm, the value $d_{\min}$ will be monotonically decreasing. In general, when the next interval is considered, if it is possible to put a point in the interval without decreasing $d_{\min}$, then we put such a point as far left as possible. Otherwise, we put a point on the right endpoint of the interval. In the latter case, we also need to adjust the points that have been determined temporarily in the previous intervals that have been considered. We adjust these points in a greedy way such that $d_{\min}$ decreases the least. A straightforward implementation of this approach can only give an $O(n^2)$ time algorithm. In order to achieve the $O(n)$ time performance, during the algorithm we maintain a "critical list" $\mathcal{L}$ of intervals, which is a subset of intervals that have been considered. This list has some properties that help us implement the algorithm in $O(n)$ time.

We should point out that our algorithm is fairly simple and easy to implement. In contrast, the rationale of the idea is quite involved and it is not an easy task to argue its correctness. Indeed, discovering the critical list is the most challenging work and it is the key idea for solving the problem in linear time.

To solve the cycle version, we convert it to a problem instance on a line and then apply our line version algorithm. More specifically, we make two copies of the intervals of $\mathcal{I}$ to a line and then apply our line version algorithm on these $2n$ intervals. The line version algorithm will find $2n$ points in these intervals and we show that a particular subset of n consecutive points of them correspond to an optimal solution for the original problem on $\mathcal{C}$.

In the following, we will present our algorithms for the line version in Section 2. The cycle version is discussed in Section 3. Due to the space limit, some proofs are omitted but can be found in the full version of the paper.

2 The Line Version

Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be the set of intervals sorted from left to right on ℓ. For any two points of p and q on ℓ, we use $|pq|$ to denote their distance. Our goal is to find a point p_i in I_i for each $1 \leq i \leq n$, such that the minimum pairwise distance of these points, i.e., $\min_{1\leq i<j\leq n} |p_i p_j|$, is maximized.

For each interval I_i, $1 \leq i \leq n$, we use l_i and r_i to denote its left and right endpoints, respectively. We assume ℓ is the x-axis. With a little abuse of notation, for any point $p \in \ell$, depending on the context, p may also refer to its coordinate on ℓ. Therefore, for each $1 \leq i \leq n$, it is required that $l_i \leq p_i \leq r_i$.

For simplicity of discussion, we make a general position assumption that no two endpoints of the intervals of $\mathcal{I}$ have the same location (our algorithm can be easily extended to the general case). Note that this implies $l_i < r_i$ for any interval I_i.

The rest of this section is organized as follows. In Section 2.1, we discuss some observations. In Section 2.2, we give an overview of our algorithm. The algorithm details are presented in Section 2.3. Finally, we discuss the correctness and analyze the running time in Section 2.4.

2.1 Observations

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of sought points. Since all intervals are disjoint, $p_1 < p_2 < \ldots < p_n$. Note that the minimum pairwise distance of the points of P is also the minimum distance of all pairs of adjacent points.

Denote by d_{opt} the minimum pairwise distance of P in an optimal solution, and d_{opt} is called the *optimal objective value*. We have the following lemma.

▶ **Lemma 1.** *$d_{opt} \leq \frac{r_j - l_i}{j-i}$ for any $1 \leq i < j \leq n$.*

Proof. Assume to the contrary that this is not true. Then there exist i and j with $i < j$ such that $d_{opt} > \frac{r_j - l_i}{j-i}$. Consider any optimal solution OPT. Note that in OPT, $p_i, p_{i+1}, \ldots, p_j$ are located in the intervals $I_i, I_{i+1}, \ldots, I_j$, respectively, and $|p_i p_j| \geq d_{opt} \cdot (j-i)$. Hence, $|p_i p_j| > r_j - l_i$. On the other hand, since $l_i \leq p_i$ and $p_j \leq r_j$, it holds that $|p_i p_j| \leq r_j - l_i$. We thus obtain contradiction. ◀

The preceding lemma leads to the following corollary and our algorithm will find such a solution as stated in the corollary.

▶ **Corollary 2.** *Suppose we find a solution (i.e., a way to place the points of P) in which the minimum pairwise distance of P is equal to $\frac{r_j - l_i}{j-i}$ for some $1 \leq i < j \leq n$. Then the solution is an optimal solution.*

2.2 The Algorithm Overview

Our algorithm will consider and process the intervals of $\mathcal{I}$ one by one from left to right. Whenever an interval I_i is processed, we will "temporarily" determine p_i in I_i. We say "temporarily" because later the algorithm may change the location of p_i. During the algorithm, a value $d_{\min}$ and two indices i^* and j^* will be maintained such that $d_{\min} = (r_{j^*} - l_{i^*})/(j^* - i^*)$ always holds.

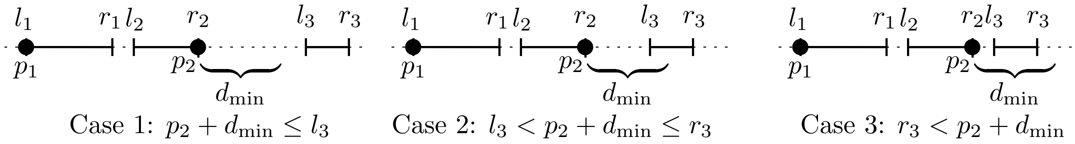

l_1
r_1
l_2
r_2
l_3
r_3
p_1
p_2
$d_{\min}$
Case 1: $p_2 + d_{\min} \leq l_3$
Case 2: $l_3 < p_2 + d_{\min} \leq r_3$
Case 3: $r_3 < p_2 + d_{\min}$

Figure 1 Illustrating the three cases when I_3 is being processed.

Initially, we set $p_1 = l_1$ and $d_{\min} = \infty$, with $i^* = j^* = 1$. In general, suppose the first $i - 1$ intervals have been processed; then $d_{\min}$ is equal to the minimum pairwise distance of the points $p_1, p_2, \ldots, p_{i-1}$, which have been temporarily determined. In fact, $d_{\min}$ is the optimal objective value for the sub-problem on the first $i - 1$ intervals. During the execution of algorithm, $d_{\min}$ will be monotonically decreasing. After all intervals are processed, $d_{\min}$ is d_{opt}. When we process the next interval I_i, we temporarily determine p_i in a greedy manner as follows. If $p_{i-1} + d_{\min} \leq l_i$, we put p_i at l_i. If $l_i < p_{i-1} + d_{\min} \leq r_i$, we put p_i at $p_{i-1} + d_{\min}$. If $p_{i-1} + d_{\min} > r_i$, we put p_i at r_i. In the first two cases, $d_{\min}$ does not change. In the third case, however, $d_{\min}$ will decrease. Further, in the third case, in order to make the decrease of $d_{\min}$ as small as possible, we need to move some points of $\{p_1, p_2, \ldots, p_{i-1}\}$ leftwards. By a straightforward approach, this moving procedure can be done in $O(n)$ time. But this will make the entire algorithm run in $O(n^2)$ time.

To have any hope of obtaining an $O(n)$ time algorithm, we need to perform the above moving "implicitly" in $O(1)$ amortized time. To this end, we need to find a way to answer the following question: Which points of $p_1, p_2, \ldots, p_{i-1}$ should move leftwards and how far should they move? To answer the question, the crux of our algorithm is to maintain a "critical list" $\mathcal{L}$ of interval indices, which bears some important properties that eventually help us implement our algorithm in $O(n)$ time.

In fact, our algorithm is fairly simple. The most "complicated" part is to use a linked list to store $\mathcal{L}$ so that the following three operations on $\mathcal{L}$ can be performed in constant time each: remove the front element; remove the rear element; add a new element to the rear. Refer to Algorithm 1 for the pseudocode.

Although the algorithm is simple, the rationale of the idea is rather involved and it is also not obvious to see the correctness. Indeed, discovering the critical list is the most challenging task and the key idea for designing our linear time algorithm. To help in understanding and give some intuition, below we use an example of only three intervals to illustrate how the algorithm works.

Initially, we set $p_1 = l_1$, $d_{\min} = \infty$, $i^* = j^* = 1$, and $\mathcal{L} = \{1\}$.

To process I_2, we first try to put p_2 at $p_1 + d_{\min}$. Clearly, $p_1 + d_{\min} > r_2$. Hence, we put p_2 at r_2. Since p_1 is already at l_1, which is the leftmost point of I_1, we do not need to move it. We update $j^* = 2$ and $d_{\min} = r_2 - l_1$. Finally, we add 2 to the rear of $\mathcal{L}$. This finishes the processing of I_2.

Next we process I_3. We try to put p_3 at $p_2 + d_{\min}$. Depending on whether $p_2 + d_{\min}$ is to the left of I_3, in I_3, or to the right of I_3, there are three cases (e.g., see Fig. 1).

1. If $p_2 + d_{\min} \leq l_3$, we set $p_3 = l_3$. We reset $\mathcal{L}$ to $\{3\}$. None of $d_{\min}$, i^*, and j^* needs to be changed in this case.
2. If $l_3 < p_2 + d_{\min} \leq r_3$, we set $p_3 = p_2 + d_{\min}$. None of $d_{\min}$, i^*, and j^* needs to be changed. Further, the critical list $\mathcal{L}$ is updated as follows.
   We first give some "motivation" on why we need to update $\mathcal{L}$. Assume later in the algorithm, say, when we process the next interval, we need to move both p_2 and p_3 leftwards simultaneously so that $|p_1p_2| = |p_2p_3|$ during the moving (this is for making

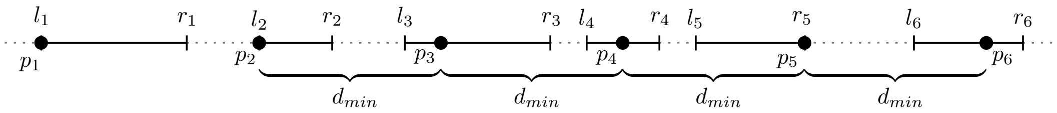


Figure 2 Illustrating the solution computed by our algorithm, with $i^* = 2$ and $j^* = 5$.

$d_{\min}$ as large as possible). The moving procedure stops once either p_2 arrives at l_2 or p_3 arrives at l_3. To determine which case happens first, it suffices to determine whether $l_2 - l_1 > \frac{l_3 - l_1}{2}$.

a. If $l_2 - l_1 > \frac{l_3-l_1}{2}$, then p_2 will arrive at l_2 first, after which p_2 cannot move leftwards any more in the rest of the algorithm but p_3 can still move leftwards.
b. Otherwise, p_3 will arrive at l_3 first, after which p_3 cannot move leftwards any more. However, although p_2 can still move leftwards, doing that would not help in making $d_{\min}$ larger.

We therefore update $\mathcal{L}$ as follows. If $l_2 - l_1 > \frac{l_3-l_1}{2}$, we add 3 to the rear of $\mathcal{L}$. Otherwise, we first remove 2 from the rear of $\mathcal{L}$ and then add 3 to the rear.

3. If $r_3 < p_2 + d_{\min}$, we set $p_3 = r_3$. Since $|p_2p_3| < d_{\min}$, $d_{\min}$ needs to be decreased. To make $d_{\min}$ as large as possible, we will move p_2 leftwards until either $|p_1p_2|$ becomes equal to $|p_2p_3|$ or p_2 arrives at l_2. To determine which event happens first, we only need to check whether $l_2 - l_1 > \frac{r_3-l_1}{2}$.

a. If $l_2 - l_1 > \frac{r_3-l_1}{2}$, the latter event happens first. We set $p_2 = l_2$ and update $d_{\min} = r_3 - l_2$ ($= |p_2p_3|$), $i^* = 2$, and $j^* = 3$. Finally, we remove 1 from the front of $\mathcal{L}$ and add 3 to the rear of $\mathcal{L}$, after which $\mathcal{L} = \{2, 3\}$.
b. Otherwise, the former event happens first. We set $p_2 = l_1 + \frac{r_3-l_1}{2}$ and update $d_{\min} = (r_3 - l_1)/2$ ($= |p_1p_2| = |p_2p_3|$) and $j^* = 3$ (i^* is still 1). Finally, we update $\mathcal{L}$ in the same way as the above second case. Namely, if $l_2 - l_1 > \frac{l_3-l_1}{2}$, we add 3 to the rear of $\mathcal{L}$; otherwise, we remove 2 from $\mathcal{L}$ and add 3 to the rear.

One may verify that in any case the above obtained $d_{\min}$ is an optimal objective value for the three intervals.

As another example, Fig. 2 illustrates the solution found by our algorithm on six intervals.

2.3 The Algorithm

We are ready to present the details of our algorithm. For any two indices $i < j$, let $P(i, j) = \{p_i, p_{i+1}, \ldots, p_j\}$.

Initially we set $p_1 = l_1$, $d_{\min} = \infty$, $i^* = j^* = 1$, and $\mathcal{L} = \{1\}$. Suppose interval $i-1$ has just been processed for some $i > 1$. Let the current critical list be $\mathcal{L} = \{k_s, k_{s+1}, \ldots k_t\}$ with $1 \le k_s < k_{s+1} < \cdots < k_t \le i-1$, i.e., $\mathcal{L}$ consists of $t - s + 1$ sorted indices in $[1, i-1]$. Our algorithm maintains the following *invariants.*

1. The "temporary" location of p_{i-1} is known.
2. $d_{\min} = (r_{j^*} - l_{i^*})/(j^* - i^*)$ with $1 \le i^* \le j^* \le i-1$.
3. $k_t = i - 1$.
4. $p_{k_s} = l_{k_s}$, i.e., p_{k_s} is at the left endpoint of the interval I_{k_s}.

5. The locations of all points of $P(1,k_s)$ have been explicitly computed and *finalized* (i.e., they will never be changed in the later algorithm).
6. For each $1 \leq j \leq k_s$, p_j is in I_j.
7. The distance of every pair of adjacent points of $P(1,k_s)$ is at least $d_{\min}$.
8. For each j with $k_s+1 \leq j \leq i-1$, p_j is "implicitly" set to $l_{k_s} + d_{\min} \cdot (j - k_s)$ and $p_j \in I_j$. In other words, the distance of every pair of adjacent points of $P(k_s, i-1)$ is exactly $d_{\min}$.
9. The critical list $\mathcal{L}$ has the following *priority property*: If $\mathcal{L}$ has more than one element (i.e., $s < t$), then for any h with $s \leq h \leq t-1$, Inequality (1) holds for any j with $k_h + 1 \leq j \leq i-1$ and $j \neq k_{h+1}$.

$$\frac{l_{k_{h+1}} - l_{k_h}}{k_{h+1} - k_h} > \frac{l_j - l_{k_h}}{j - k_h}. \tag{1}$$

We give some intuition on what the priority property implies. Suppose we move all points in $P(k_s+1, i-1)$ leftwards simultaneously such that the distances between all adjacent pairs of points of $P(k_s, i-1)$ keep the same (by the above eighth invariant, they are the same before the moving). Then, Inequality (1) with $h = s$ implies that $p_{k_{s+1}}$ is the first point of $P(k_s+1, i-1)$ that arrives at the left endpoint of its interval. Once $p_{k_{s+1}}$ arrives at the interval left endpoint, suppose we continue to move the points of $P(k_{s+1}+1, i-1)$ leftwards simultaneously such that the distances between all adjacent pairs of points of $P(k_{s+1}, i-1)$ are the same. Then, Inequality (1) with $h = s+1$ makes sure that $p_{k_{s+2}}$ is the first point of $P(k_{s+1}+1, i-1)$ that arrives at the left endpoint of its interval. Continuing the above can explain the inequality for $h = s+2, s+3, \ldots, t-1$.

The priority property further leads to the following observation.

▶ **Observation 1.** *For any h with $s \leq h \leq t-2$, the following holds:*

$$\frac{l_{k_{h+1}} - l_{k_h}}{k_{h+1} - k_h} > \frac{l_{k_{h+2}} - l_{k_{h+1}}}{k_{h+2} - k_{h+1}}.$$

Proof. Note that $k_h + 1 \leq k_{h+1} < k_{h+2} \leq i-1$. Let $j = k_{h+2}$. By Inequality (1), we have

$$\frac{l_{k_{h+1}} - l_{k_h}}{k_{h+1} - k_h} > \frac{l_{k_{h+2}} - l_{k_h}}{k_{h+2} - k_h}. \tag{2}$$

Note that for any four positive numbers a, b, c, d such that $a < c$, $b < d$, and $\frac{a}{b} > \frac{c}{d}$, it holds that $\frac{a}{b} > \frac{c-a}{d-b}$. Applying this to Inequality (2) will obtain the observation. ◀

▶ Remark. By Corollary 2, Invariants (2), (6), (7), and (8) together imply that $d_{\min}$ is the optimal objective value for the sub-problem on the first $i-1$ intervals.

One may verify that initially after I_1 is processed, all invariants trivially hold (we finalize p_1 at l_1). In the following we describe the general step of our algorithm to process the interval I_i. We will also show that all algorithm invariants hold after I_i is processed.

Depending on whether $p_{i-1} + d_{\min}$ is to the left of I_i, in I_i, or to the right of I_i, there are three cases.

2.3.1 The case $p_{i-1} + d_{\min} \leq l_i$

In this case, $p_{i-1} + d_{\min}$ is to the left of I_i. We set $p_i = l_i$ and finalize it. We do not change $d_{\min}$, i^*, or j^*. Further, for each $j \in [k_s+1, i-1]$, we explicitly compute $p_j = l_{k_s} + d_{\min} \cdot (j - k_s)$ and finalize it. Finally, we reset $\mathcal{L} = \{i\}$. The proof of Lemma 3 is omitted.

▶ **Lemma 3.** *In the case $p_{i-1} + d_{\min} \leq l_i$, all algorithm invariants hold after I_i is processed.*

2.3.2 The case $l_i < p_{i-1} + d_{\min} \leq r_i$

In this case, $p_{i-1} + d_{\min}$ is in I_i. We set $p_i = p_{i-1} + d_{\min}$. We do not change $d_{\min}$, i^*, or j^*. We update the critical list $\mathcal{L}$ by the following *rear-processing procedure* (because the elements of $\mathcal{L}$ are considered from the rear to the front).

If $s = t$, i.e., $\mathcal{L}$ only has one element, then we simply add i to the rear of $\mathcal{L}$. Otherwise, we first check whether the following inequality is true.

$$\frac{l_{k_t} - l_{k_{t-1}}}{k_t - k_{t-1}} > \frac{l_i - l_{k_{t-1}}}{i - k_{t-1}}. \tag{3}$$

If it is true, then we add i to the end of $\mathcal{L}$.

If it is not true, then we remove k_t from $\mathcal{L}$ and decrease t by 1. Next, we continue to check whether Inequality (3) (with the decreased t) is true and follow the same procedure until either the inequality becomes true or $s = t$. In either case, we add i to the end of $\mathcal{L}$. Finally, we increase t by 1 to let k_t refer to i.

This finishes the rear-processing procedure for updating $\mathcal{L}$. The proof of Lemma 4 is omitted.

▶ **Lemma 4.** *In the case $l_i < p_{i-1} + d_{\min} \leq r_i$, all algorithm invariants hold after I_i is processed.*

2.3.3 The case $p_{i-1} + d_{\min} > r_i$

In this case, $p_{i-1} + d_{\min}$ is to the right of I_i. We first set $p_i = r_i$. Then we perform the following *front-processing procedure* (because it processes the elements of $\mathcal{L}$ from the front to the rear).

If $\mathcal{L}$ has only one element (i.e., $s = t$), then we stop.

Otherwise, we check whether the following is true

$$\frac{l_{k_{s+1}} - l_{k_s}}{k_{s+1} - k_s} > \frac{r_i - l_{k_s}}{i - k_s}. \tag{4}$$

If it is true, then we perform the following *finalization step*: for each $j = k_s + 1, k_s + 2, \ldots, k_{s+1}$, we explicitly compute $p_j = l_{k_s} + \frac{l_{k_{s+1}} - l_{k_s}}{k_{s+1} - k_s} \cdot (j - k_s)$ and finalize it. Further, we remove k_s from $\mathcal{L}$ and increase s by 1. Next, we continue the same procedure as above (with the increased s), i.e., first check whether $s = t$, and if not, check whether Inequality (4) is true. The front-processing procedure stops if either $s = t$ (i.e., $\mathcal{L}$ only has one element) or Inequality (4) is not true.

After the front-processing procedure, we update $d_{\min} = (r_i - l_{k_s})/(i - k_s)$, $i^* = k_s$, and $j^* = i$. Finally, we update the critical list $\mathcal{L}$ using the rear-processing procedure, in the same way as in the above second case where $l_i < p_{i-1} + d_{\min} \leq r_i$. We also "implicitly" set $p_j = l_{k_s} + d_{\min} \cdot (j - k_s)$ for each $j \in [k_s + 1, i]$ (this is only for the analysis and our algorithm does not do so explicitly).

This finishes the processing of I_i. The proof of Lemma 5 is omitted.

▶ **Lemma 5.** *In the case $p_{i-1} + d_{\min} > r_i$, all algorithm invariants hold after I_i is processed.*

The above describes a general step of the algorithm for processing the interval I_i. In addition, if $i = n$ and $k_s < n$, we also need to perform the following additional finalization step: for each $j \in [k_s + 1, n]$, we explicitly compute $p_j = l_{k_s} + d_{\min} \cdot (j - k_s)$ and finalize it. This finishes the algorithm. The pseudocode is given in Algorithm 1.

Algorithm 1: The algorithm for the line version of the problem

```
Input: $n$ intervals $I_1, I_2, \ldots, I_n$ sorted from left to right on $\ell$
Output: $n$ points $p_1, p_2, \ldots, p_n$ with $p_i \in I_i$ for each $1 \leq i \leq n$

$p_1 \leftarrow l_1$, $i^* \leftarrow 1$, $j^* \leftarrow 1$, $d_{\min} \leftarrow \infty$, $\mathcal{L} \leftarrow \{1\}$;
for $i \leftarrow 2$ to $n$ do
    if $p_{i-1} + d_{\min} \leq l_i$ then
        $p_i \leftarrow l_i$, $\mathcal{L} \leftarrow \{i\}$;
    else
        if $l_i < p_{i-1} + d_{\min} \leq r_i$ then
            $p_i \leftarrow p_{i-1} + d_{\min}$;
        else /* $p_{i-1} + d_{\min} > r_i$                                  */
            $p_i \leftarrow r_i$, $k_s \leftarrow$ the front element of $\mathcal{L}$;
            while $|\mathcal{L}| > 1$ do            /* the front-processing procedure */
                if $\frac{l_{k_{s+1}} - l_{k_s}}{k_{s+1} - k_s} > \frac{r_i - l_{k_s}}{i - k_s}$ then
                    for $j \leftarrow k_s + 1$ to $k_{s+1}$ do
                        $p_j \leftarrow l_{k_s} + \frac{l_{k_{s+1}} - l_{k_s}}{k_{s+1} - k_s} \cdot (j - k_s)$;
                    remove $k_s$ from $\mathcal{L}$,   $k_s \leftarrow$ the front element of $\mathcal{L}$;
                else
                    break;
            $i^* \leftarrow k_s$, $j^* \leftarrow i$, $d_{\min} \leftarrow \frac{r_{j^*} - l_{i^*}}{j^* - i^*}$;
        while $|\mathcal{L}| > 1$ do                 /* the rear-processing procedure */
            $k_t \leftarrow$ the rear element of $\mathcal{L}$;
            if $\frac{l_{k_t} - l_{k_{t-1}}}{k_t - k_{t-1}} > \frac{l_i - l_{k_{t-1}}}{i - k_{t-1}}$ then break;
            ;
            remove $k_t$ from $\mathcal{L}$;
        add $i$ to the rear of $\mathcal{L}$;
$k_s \leftarrow$ the front element of $\mathcal{L}$;
if $k_s < n$ then
    for $j \leftarrow k_s + 1$ to $n$ do
        $p_j \leftarrow l_{k_s} + d_{\min} \cdot (j - k_s)$;
```

2.4 The Correctness and the Time Analysis

Based on the algorithm invariants and Corollary 2, the following lemma proves the correctness of the algorithm.

▶ **Lemma 6.** *The algorithm correctly computes an optimal solution.*

Proof. Suppose $P = \{p_1, p_2, \ldots, p_n\}$ is the set of points computed by the algorithm. Let $d_{\min}$ be the value and $\mathcal{L} = \{k_s, k_{s+1}, \ldots, k_t\}$ be the critical list after the algorithm finishes.

We first show that for each $j \in [1, n]$, p_j is in I_j. According to the sixth algorithm invariant of $\mathcal{L}$, for each $j \in [1, k_s]$, p_j is in I_j. If $k_s = n$, then we are done with the proof. Otherwise, for each $j \in [k_s + 1, n]$, according to the additional finalization step after I_n is processed, $p_j = l_{k_s} + d_{\min} \cdot (j - k_s)$, which is in I_j by the eighth algorithm invariant.

Next we show that the distance of every pair of adjacent points of P is at least $d_{\min}$. By the seventh algorithm invariant, the distance of every pair of adjacent points of $P(1, k_s)$ is at least $d_{\min}$. If $k_s = n$, then we are done with the proof. Otherwise, it is sufficient to show that the distance of every pair of adjacent points of $P(k_s, n)$ is at least $d_{\min}$, which is true according to the additional finalization step after I_n is processed.

The above proves that P is a *feasible solution* with respect to $d_{\min}$, i.e., all points of P are in their corresponding intervals and the distance of every pair of adjacent points of P is at least $d_{\min}$.

To show that P is also an optimal solution, based on the second algorithm invariant, it holds that $d_{\min} = \frac{r_{j^*} - l_{i^*}}{j^* - i^*}$. By Corollary 2, $d_{\min}$ is an optimal objective value. Therefore, P is an optimal solution. ◀

The running time of the algorithm is analyzed in the proof of Theorem 7.

▶ **Theorem 7.** *Our algorithm computes an optimal solution of the line version of points dispersion problem in $O(n)$ time.*

Proof. By Lemma 6, we only need to show that the running time of the algorithm is $O(n)$.

To process an interval I_i, according to our algorithm, we only spend $O(1)$ time in addition to two possible procedures: a front-processing procedure and a rear-processing procedure. Note that the front-processing procedure may contain several finalization steps. There may also be an additional finalization step after I_n is processed. For the purpose of analyzing the total running time of the algorithm, we exclude the finalization steps from the front-processing procedures.

For processing I_i, the front-processing procedure (excluding the time of the finalization steps) runs in $O(k + 1)$ time where k is the number of elements removed from the front of the critical list $\mathcal{L}$. An easy observation is that any element can be removed from $\mathcal{L}$ at most once in the entire algorithm. Hence, the total time of all front-processing procedures in the entire algorithm is $O(n)$.

Similarly, for processing I_i, the rear-processing procedure runs in $O(k + 1)$ time where k is the number of elements removed from the rear of $\mathcal{L}$. Again, since any element can be removed from $\mathcal{L}$ at most once in the entire algorithm, the total time of all rear-processing procedures in the entire algorithm is $O(n)$.

Clearly, each point is finalized exactly once in the entire algorithm. Hence, all finalization steps in the entire algorithm together take $O(n)$ time.

Therefore, the algorithm runs in $O(n)$ time in total. ◀

3 The Cycle Version

In the cycle version, the intervals of $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ in their index order are sorted cyclically on $\mathcal{C}$. Recall that the intervals of $\mathcal{I}$ are pairwise disjoint.

For each $i \in [1, n]$, let l_i and r_i denote the two endpoints of I_i, respectively, such that if we move from l_i to r_i clockwise on $\mathcal{C}$, we will always stay on I_i.

For any two points p and q on $\mathcal{C}$, we use $|\overrightarrow{pq}|$ to denote the length of the arc of $\mathcal{C}$ from p to q clockwise, and thus the distance of p and q on $\mathcal{C}$ is $\min\{|\overrightarrow{pq}|, |\overrightarrow{qp}|\}$.

For each interval $I_i \in \mathcal{I}$, we use $|I_i|$ to denote its length; note that $|I_i| = |\overrightarrow{l_i r_i}|$. We use $|\mathcal{C}|$ to denote the total length of $\mathcal{C}$.

Our goal is to find a point p_i in I_i for each $i \in [1, n]$ such that the minimum distance between any pair of these points, i.e., $\min_{1 \le i < j \le n} |p_i p_j|$, is maximized.

Let $P = \{p_1, p_2, \ldots, p_n\}$ and let d_{opt} be the optimal objective value. It is obvious that $d_{opt} \leq \frac{|\mathcal{C}|}{n}$. Again, for simplicity of discussion, we make a general position assumption that no two endpoints of the intervals have the same location on $\mathcal{C}$.

3.1 The Algorithm

The main idea is to convert the problem to a problem instance on a line and then apply our line version algorithm. More specifically, we copy all intervals of $\mathcal{I}$ twice to a line ℓ and then apply our line version algorithm on these $2n$ intervals. The line version algorithm will find $2n$ points in these intervals. We will show that a subset of n points in n consecutive intervals correspond to an optimal solution for our original problem on $\mathcal{C}$. The details are given below.

Let ℓ be the x-axis. For each $1 \leq i \leq n$, we create an interval $I'_i = [l'_i, r'_i]$ on ℓ with $l'_i = |\overrightarrow{l_1 l_i}|$ and $r'_i = l'_i + |I_i|$, which is actually a copy of I_i. In other words, we first put a copy I'_1 of I_1 at ℓ such that its left endpoint is at 0 and then we continuously copy other intervals to ℓ in such a way that the pairwise distances of the intervals on ℓ are the same as the corresponding clockwise distances of the intervals of $\mathcal{I}$ on $\mathcal{C}$. The above only makes one copy for each interval of $\mathcal{I}$. Next, we make another copy for each interval of $\mathcal{I}$ in a similar way: for each $1 \leq i \leq n$, we create an interval $I'_{i+n} = [l'_{i+n}, r'_{i+n}]$ on ℓ with $l'_{i+n} = l'_i + |\mathcal{C}|$ and $r'_{i+n} = r'_i + |\mathcal{C}|$. Let $\mathcal{I}' = \{I'_1, I'_2, \ldots, I'_{2n}\}$. Note that the intervals of $\mathcal{I}'$ in their index order are sorted from left to right on ℓ.

We apply our line version algorithm on the intervals of $\mathcal{I}'$. However, a subtle change is that here we initially set $d_{\min} = \frac{|\mathcal{C}|}{n}$ instead of $d_{\min} = \infty$. The rest of the algorithm is the same as before. We want to emphasize that this change on initializing $d_{\min}$ is necessary to guarantee the correctness of our algorithm for the cycle version. A consequence of this change is that after the algorithm finishes, if $d_{\min}$ is still equal to $\frac{|\mathcal{C}|}{n}$, then $\frac{|\mathcal{C}|}{n}$ may not be the optimal objective value for the above line version problem, but if $d_{\min} < \frac{|\mathcal{C}|}{n}$, then $d_{\min}$ must be the optimal objective value. As will be clear later, this does not affect our final solution for our original problem on the cycle $\mathcal{C}$. Let $P' = \{p'_1, \ldots, p'_{2n}\}$ be the points computed by the line version algorithm with $p'_i \in I'_i$ for each $i \in [1, 2n]$.

Let k be the largest index in $[1, n]$ such that $p'_k = l'_k$. Note that such an index k always exists since $p'_1 = l'_1$. Due to that we initialize $d_{\min} = \frac{|\mathcal{C}|}{n}$ in our line version algorithm, we can prove the following lemma.

▶ **Lemma 8.** *It holds that $p'_{k+n} = l'_{k+n}$.*

Proof. We prove the lemma by contradiction. Assume to the contrary that $p'_{k+n} \neq l'_{k+n}$. Since $p'_{k+n} \in I'_{k+n}$, it must be that $p'_{k+n} > l'_{k+n}$. Let p'_i be the rightmost point of P' to the left of p'_{k+n} such that p'_i is at the left endpoint of its interval I'_i. Depending on whether $i \leq n$, there are two cases.

1. If $i > n$, then let $j = i - n$. Since $i < k + n$, $j < k$. We claim that $|p'_j p'_k| < |p'_{j+n} p'_{n+k}|$. Indeed, since $p'_j \geq l'_j$ and $p'_k = l'_k$, we have $|p'_j p'_k| \leq |l'_j l'_k|$. Note that $|l'_j l'_k| = |l'_{j+n} l'_{k+n}|$. On the other hand, since $p'_{j+n} = l'_{j+n}$ and $p'_{k+n} > l'_{k+n}$, it holds that $|p'_{j+n} p'_{k+n}| > |l'_{j+n} l'_{k+n}|$. Therefore, the claim follows.

   Let d be the value of $d_{\min}$ right before the algorithm processes I'_i. Since during the execution of our line version algorithm $d_{\min}$ is monotonically decreasing, it holds that $|p'_j p'_k| \geq d \cdot (k - j)$. Further, by the definition of i, for any $m \in [i+1, k+n]$, $p'_m > l'_m$. Thus, according to our line version algorithm, the distance of every adjacent pair of points of $p'_i, p'_{i+1} \ldots, p'_{k+n}$ is at most d. Thus, $|p'_i p'_{k+n}| \leq d \cdot (k + n - i)$. Since $j = i - n$, we have $|p'_{j+n}p'_{k+n}| \leq d \cdot (k-j)$. Hence, we obtain $|p'_jp'_k| \geq |p'_{j+n}p'_{k+n}|$. However, this contradicts with our above claim.

2. If $i \leq n$, then by the definition of k, we have $i = k$. Let d be the value of $d_{\min}$ right before the algorithm processes I'_i. By the definition of i, the distance of every adjacent pair of points of $p'_k, p'_{k+1} \ldots, p'_{k+n}$ is at most d. Hence, $|p'_kp'_{k+n}| \leq n \cdot d$. Since $p'_k = l'_k$ and $p'_{n+k} > l'_{n+k}$, we have $|p'_kp'_{n+k}| > |l'_kl'_{n+k}| = |\mathcal{C}|$. Therefore, we obtain that $n \cdot d > |\mathcal{C}|$. However, since we initially set $d_{\min} = |\mathcal{C}|/n$ and the value $d_{\min}$ is monotonically decreasing during the execution of the algorithm, it must hold that $n \cdot d \leq |\mathcal{C}|$. We thus obtain contradiction.

Therefore, it must hold that $p'_{n+k} = l'_{n+k}$. The lemma thus follows. ◀

We construct a solution set P for our cycle version problem by mapping the points $p'_k, p'_{k+1}, \ldots, p'_{n+k-1}$ back to $\mathcal{C}$. Specifically, for each $i \in [k, n]$, we put p_i at a point on $\mathcal{C}$ with a distance $p'_i - l'_i$ clockwise from l_i; for each $i \in [1, k-1]$, we put p_i at a point on $\mathcal{C}$ at a distance $p'_{i+n} - l'_{i+n}$ clockwise from l_i. Clearly, p_i is in I_i for each $i \in [1, n]$. Hence, P is a "feasible" solution for our cycle version problem. Below we show that P is actually an optimal solution.

Consider the value $d_{\min}$ returned by the line version algorithm after all intervals of $\mathcal{I}'$ are processed. Since the distance of every pair of adjacent points of $p'_k, p'_{k+1}, \ldots, p'_{n+k}$ is at least $d_{\min}$, $p'_k = l'_k$, $p'_{n+k} = l'_{n+k}$ (by Lemma 8), and $|l'_kl'_{n+k}| = |\mathcal{C}|$, by our way of constructing P, the distance of every pair of adjacent points of P on $\mathcal{C}$ is at least $d_{\min}$.

Recall that d_{opt} is the optimal object value of our cycle version problem. The following lemma implies that P is an optimal solution.

▶ **Lemma 9.** $d_{\min} = d_{opt}$.

Proof. Since P is a feasible solution with respect to $d_{\min}$, $d_{\min} \leq d_{opt}$ holds.

If $d_{\min} = |\mathcal{C}|/n$, since $d_{opt} \leq |\mathcal{C}|/n$, we obtain $d_{opt} \leq d_{\min}$. Therefore, $d_{opt} = d_{\min}$, which leads to the lemma.

In the following, we assume $d_{\min} \neq |\mathcal{C}|/n$. Hence, $d_{\min} < |\mathcal{C}|/n$. According to our line version algorithm, there must exist $i^* < j^*$ such that $d_{\min} = \frac{r'_{j^*} - l'_{i^*}}{j^* - i^*}$. We assume there is no i with $i^* < i < j^*$ such that $d_{\min} = \frac{r'_{j^*} - l'_i}{j^* - i}$ since otherwise we could change i^* to i. Since $d_{\min} = \frac{r'_{j^*} - l'_{i^*}}{j^* - i^*}$, it is necessary that $p'_{i^*} = l'_{i^*}$ and $p'_{j^*} = r'_{j^*}$. By the above assumption, there is no $i \in [i^*, j^*]$ such that $p'_i = l'_i$. Since $p'_k = l'_k$ and $p'_{k+n} = l'_{k+n}$ (by Lemma 8), one of the following three cases must be true: $j^* < k$, $k \leq i^* < j^* < n+k$, or $n+k \leq i^*$. In any case, $j^* - i^* < n$. By our way of defining r'_{j^*} and l'_{i^*}, we have the following:

$$d_{\min} = \frac{r'_{j^*} - l'_{i^*}}{j^* - i^*} = \begin{cases} |\overrightarrow{l_{i^*}r_{j^*}}|/(j^* - i^*), & \text{if } j^* \leq n, \\ |\overrightarrow{l_{i^*}r_{j^*-n}}|/(j^* - i^*), & \text{if } i^* \leq n < j^*, \\ |\overrightarrow{l_{i^*-n}r_{j^*-n}}|/(j^* - i^*) & \text{if } n < i^*. \end{cases}$$

We claim that $d_{opt} \leq d_{\min}$ in all three cases: $j^* \leq n$, $i^* \leq n < j^*$, and $n < i^*$. In the following we only prove the claim in the first case where $j^* \leq n$ since the other two cases can be proved analogously (e.g., by re-numbering the indices).

Our goal is to prove $d_{opt} \leq \frac{|\overrightarrow{l_{i^*}r_{j^*}}|}{j^* - i^*}$. Consider any optimal solution in which the solution set is $P = \{p_1, p_2, \ldots, p_n\}$. Consider the points $p_{i^*}, p_{i^*+1}, \ldots, p_{j^*}$, which are in the intervals $I_{i^*}, I_{i^*+1}, \ldots, I_{j^*}$. Clearly, $|\overrightarrow{p_kp_{k+1}}| \geq d_{opt}$ for any $k \in [i^*, j^* - 1]$. Therefore, we have $|\overrightarrow{p_{i^*}p_{j^*}}| \geq d_{opt} \cdot (j^* - i^*)$. Note that $|\overrightarrow{p_{i^*}p_{j^*}}| \leq |\overrightarrow{l_{i^*}r_{j^*}}|$. Consequently, we obtain $d_{opt} \leq \frac{|\overrightarrow{l_{i^*}r_{j^*}}|}{j^* - i^*}$.

Since both $d_{\min} \leq d_{opt}$ and $d_{opt} \leq d_{\min}$, $d_{opt} = d_{\min}$ holds. The lemma thus follows. ◀

The above shows that P is an optimal solution with $d_{opt} = d_{\min}$. The running time of the algorithm is $O(n)$ because the line version algorithm runs in $O(n)$ time. As a summary, we have the following theorem.

▶ **Theorem 10.** *The cycle version of the points dispersion problem is solvable in $O(n)$ time.*

Acknowledgment. The authors would like to thank Minghui Jiang for suggesting the problem to them.

References

1 C. Baur and S. P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.

2 M. Benkert, J. Gudmundsson, C. Knauer, R. van Oostrum, and A. Wolff. A polynomial-time approximation algorithm for a geometric dispersion problem. *Int. J. Comput. Geometry Appl.*, 19(3):267–288, 2009.

3 M. Chrobak, C. Dürr, W. Jawor, L. Kowalik, and M. Kurowski. A note on scheduling equal-length jobs to maximize throughput. *Journal of Scheduling*, 9(1):71–73, 2006.

4 M. Chrobak, W. Jawor, J. Sgall, and T. Tichý. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM Journal of Computing*, 36(6):1709–1728, 2007.

5 E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46:48–60, 1990.

6 E. Fernández, J. Kalcsics, and S. Nickel. The maximum dispersion problem. *Omega*, 41(4):721–730, 2013.

7 R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12:133–137, 1981.

8 Z. Füredi. The densest packing of equal circles into a parallel strip. *Discrete and Computational Geometry*, 6:95–106, 1991.

9 M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal of Computing*, 10:256–269, 1981.

10 G. Jäger, A. Srivastav, and K. Wolf. Solving generalized maximum dispersion with linear programming. In *Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management*, pages 1–10, 2007.

11 T. Lang and E. B. Fernández. Scheduling of unit-length independent tasks with execution constraints. *Information Processing Letters*, 4:95–98, 1976.

12 C. D. Maranasa, C. A. Floudas, and P. M. Pardalosb. New results in the packing of equal circles in a square. *Discrete Mathematics*, 142:287–293, 1995.

13 O. A. Prokopyev, N. Kong, and D. L. Martinez-Torres. The equitable dispersion problem. *European Journal of Operational Research*, 197(1):59–67, 2009.

14 S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Facility dispersion problems: Heuristics and special cases. *Algorithms and Data Structures*, 519:355–366, 1991.

15 S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

16 B. Simons. A fast algorithm for single processor scheduling. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 246–252, 1978. doi:10.1109/SFCS.1978.4.

17 N. Vakhania. A study of single-machine scheduling problem to maximize throughput. *Journal of Scheduling*, 16(4):395–403, 2013.

18 N. Vakhania and F. Werner. Minimizing maximum lateness of jobs with naturally bounded job data on a single machine in polynomial time. *Theor. Comp. Science*, 501:72–81, 2013.

19 D. W. Wang and Y.-S. Kuo. A study on two geometric location problems. *Information Processing Letters*, 28:281–286, 1988.

Optimal Nonpreemptive Scheduling in a Smart Grid Model

Fu-Hong Liu[1], Hsiang-Hsuan Liu[2], and Prudence W. H. Wong[3]

1 Department of Computer Science, National Tsing Hua University, Taiwan
fhliu,hhliu@cs.nthu.edu.tw

2 Department of Computer Science, National Tsing Hua University, Taiwan; and Department of Computer Science, University of Liverpool, UK
hhliu,pwong@liverpool.ac.uk

3 Department of Computer Science, University of Liverpool, UK
pwong@liverpool.ac.uk

Abstract

We study a scheduling problem arising in demand response management in smart grid. Consumers send in power requests with a flexible feasible time interval during which their requests can be served. The grid controller, upon receiving power requests, schedules each request within the specified interval. The electricity cost is measured by a convex function of the load in each timeslot. The objective is to schedule all requests with the minimum total electricity cost. Previous work has studied cases where jobs have unit power requirement and unit duration. We extend the study to arbitrary power requirement and duration, which has been shown to be NP-hard. We give the first online algorithm for the general problem, and prove that the worst case competitive ratio is asymptotically optimal. We also prove that the problem is fixed parameter tractable. Due to space limit, the missing proofs are presented in the full paper.

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

Keywords and phrases Scheduling, Smart Grid, Convex function cost, Fixed parameter tractable, Online algorithms, Non-preemptive

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.53

1 Introduction

We study a scheduling problem arising in "demand response management" in a smart grid [15, 20, 28]. The electrical smart grid is one of the major challenges in the 21st century [25]. The smart grid [22] is a power grid system that makes power generation, distribution and consumption more efficient through information and communication technologies. Peak demand hours happen only for a short duration, yet makes existing electrical grid less efficient. It has been noted in [7] that in the US power grid, 10% of all generation assets and 25% of distribution infrastructure are required for less than 400 hours per year, roughly 5% of the time [25]. *Demand response management*1 attempts to overcome this problem by shifting users' demand to off-peak hours in order to reduce peak load.

It is demonstrated in [20] that demand response is of remarkable advantage to consumers, utilities, and society. Effective demand load management brings down the cost of operating the grid, energy generation and distribution [19]. It is not only advantageous to the supplier but also to the consumers as well. It is common that electricity supplier charges according to the generation cost. Therefore, it is to the consumers' advantage to reduce electricity consumption at high price and hence reduce the electricity bill [24].

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 53; pp. 53:1–53:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The operator and consumers communicate through smart metering devices [22]. A consumer sends in a power request with the power requirement, required duration of service, and the time interval that this request can be served (giving some flexibility). For example, a consumer may want the dishwasher to operate for one hour during the periods from 8am to 11am. The grid operator upon receiving requests has to schedule them in their respective time intervals using the minimum energy cost. The *load* of the grid at each timeslot is the sum of the power requirements of all requests allocated to that timeslot. The *electricity cost* is modeled by a convex function on the load: we consider the cost to be the α-th power of the load, where $\alpha > 1$ is some constant. Typically, α is small, e.g., $\alpha = 2$ [10].

Previous work. Koutsopoulos and Tassiulas [17] have formulated a similar problem to our problem where the cost function is piecewise linear. They show that the problem is NP-hard, and their proof can be adapted to show the NP-hardness of the general problem studied in this paper [6]. Burcea et al. [6] gave polynomial time optimal algorithms for the case of unit height (cf. unit power requirement) and unit width (cf. duration of request). Feng et al. [12] have claimed that a simple greedy algorithm is 2-competitive for the unit case and $\alpha = 2$. However, in our full paper [18], we show a counter example that the greedy algorithm is at least 3-competitive. This implies that it is still an open question to derive online algorithms for the problem. Salinas et al. [24] considered a multi-objective problem to minimize energy consumption cost and maximize some utility. A closely related problem is to manage the load by changing the price of electricity over time [21, 11]. Reviews of smart grid can be found in [15, 20, 28]. The combinatorial problem in this paper has analogy to the load balancing problem [3] and machine minimization problem [8, 9, 23] but the main differences are the objective being maximum load and jobs are unit height [8, 9, 23]. Minimizing maximum load has also been looked at in the context of smart grid [1, 27], some of which further consider allowing reshaping of the jobs [1]. As to be discussed in the full paper, our problem is more difficult than minimizing the maximum load. Our problem also has resemblance to the dynamic speed scaling problem [2, 26, 5] and our algorithm has employed some techniques there.

Our contribution. We propose the first online algorithm for the general Grid problem with competitive ratio polylogarithm in the max-min ratio of the duration of jobs (Section 4); and show that the competitive ratio is asymptotically optimal. The algorithm is based on an $O(1)$-competitive online algorithm for jobs with uniform duration (Section 3). We also propose $O(1)$-competitive online algorithms for jobs with uniform power requirement and agreeable deadlines (Section 5). Table 1 gives a summary. In addition, we show that the Grid problem is fixed parameter tractable by proposing the first fixed parameter exact algorithms for the problem; and derive lower bounds on the running time (Section 6). Interestingly, both our online algorithm and exact algorithms depend on the variation of the job widths but not the variation of the job heights.

Our online algorithms are based on identifying a relationship with the dynamic speed (voltage) scaling (DVS) problem [26]. The main challenge, even when jobs have uniform width or uniform height, is that in time intervals where the "workload" is low, the optimal DVS schedule may have much lower cost than the optimal Grid schedule because jobs in DVS schedules can effectively be stretched as flat as possible while jobs in Grid schedules have rigid duration and cannot be stretched. In such case, it is insufficient to simply compare with the optimal DVS schedule. Therefore, our analysis is divided into two parts: for high workload intervals, we compare with the optimal DVS schedule; and for low workload intervals, we

Table 1 Summary of online algorithms for different input instances.

Width	Height	Competitive Ratio
Uniform	Arbitrary	$O(1)$-competitive [Section 3]
Arbitrary	Arbitrary	$\Theta(\log^{\alpha}(\frac{w_{\max}}{w_{\min}}))$-competitive [Section 4]
Arbitrary	Uniform	$O(1)$-competitive [input with agreeable deadlines] [Section 5]

directly compare with the optimal Grid schedule via a lower bound on the total workload over these intervals (Lemmas 2 and 11). For jobs with arbitrary width, we adopt the natural approach of classification based on job width. We then align the "feasible interval" of each job in a more uniform way so that we can use the results on uniform width (Lemma 6).

In designing exact algorithms we use interval graphs to represent the jobs and the important notion maximal cliques to partition the time horizon into disjoint windows. Such partition usually leads to optimal substructures; however, non-preemption makes it trickier and requires a smart way to handle jobs spanning multiple windows. We describe how to handle such jobs without adding a lot of overhead. We remark that our approach can solve other problems like minimizing peak load in Grid and the machine minimization problem.

2 Definitions and preliminaries

The input. The time is labeled from 0 to τ and we consider events (release time, deadlines) occurring at integral time. We call the unit time $[t, t+1)$ *timeslot* t. We denote by $\mathcal{J}$ a set of input jobs in which each job J comes with *release time* $r(J)$, *deadline* $d(J)$, *width* $w(J)$ representing the duration required by J, and *height* $h(J)$ representing the power required by J. We assume $r(J)$, $d(J)$, $w(J)$, and $h(J)$ are integers. The *feasible interval*, denoted by $I(J)$, is defined as the interval $[r(J), d(J))$ and we say that J is *available* during $I(J)$.

In Section 4, we consider an algorithm that classifies jobs according to their widths. To ease discussion, we let $w_{\max}$ and $w_{\min}$ be the maximum and minimum width over all jobs, respectively. We further define the max-min ratio of width, denoted by K, to be $K = \frac{w_{\max}}{w_{\min}}$.

Without loss of generality, we assume that $w_{\min} = 1$. We say that a job J is in *class* C_p if and only if $2^{p-1} < w(J) \le 2^p$ for any $0 \le p \le \lceil \log K \rceil$.

Feasible schedule. A *feasible* schedule S assigns for each job J a *start time* $st(S, J) \in \mathbb{Z}$ meaning that J runs during $[st(S, J), et(S, J))$, where the *end time* $et(S, J) = st(S, J) + w(J)$. Note that this means preemption is not allowed. The *load* of S at time t, denoted by $\ell(S, t)$ is the sum of the height (power request) of all jobs running at t, i.e., $\ell(S, t) = \sum_{J : t \in [st(S,J), et(S,J))} h(J)$. We drop S and use $\ell(t)$ when the context is clear. We use $\mathcal{A}(\mathcal{J})$ to denote the schedule of an algorithm $\mathcal{A}$ on $\mathcal{J}$. We denote by $\mathcal{O}$ the optimal algorithm.

The cost of a schedule S is the sum of the α-th power of the load over all time, for a constant $\alpha > 1$, i.e., $\mathrm{cost}(S) = \sum_t (\ell(S, t))^{\alpha}$. For a set of timeslots $\mathcal{I}$ (not necessarily contiguous), we denote by $\mathrm{cost}(S, \mathcal{I}) = \sum_{t \in \mathcal{I}} (\ell(S, t))^{\alpha}$. The objective is to find a feasible schedule with minimum cost. We call this the Grid problem.

Online algorithms. We consider online algorithms, where the job information is only revealed at the time the job is released; the algorithm has to decide which jobs to run at the current time without future information and decisions made cannot be changed later. Let $\mathcal{A}$ be an online algorithm. We say that $\mathcal{A}$ is c-competitive if for all input job sets $\mathcal{J}$, we have

$\text{cost}(\mathcal{A}(\mathcal{J})) \leq c \cdot \text{cost}(\mathcal{O}(\mathcal{J}))$. In particular, we consider non-preemptive algorithms where a job cannot be preempted to resume/restart later.

Special input instances. A job J is said to be unit-width (resp. unit-height) if $w(J) = 1$ (resp. $h(J) = 1$). A job set is said to be uniform-width (resp. uniform-height) if the width (resp. height) of all jobs are the same. A job set is said to have *agreeable deadlines* if for any two jobs J_1 and J_2, $r(J_1) \leq r(J_2)$ implies $d(J_1) \leq d(J_2)$.

Relating to the speed scaling problem. The Grid problem resembles the dynamic speed scaling (DVS) problem [26] and we are going to refer to three DVS algorithms, namely, the optimal $\mathcal{YDS}$ algorithm, the online algorithms called $\mathcal{BKP}$ [4] and $\mathcal{AVR}$ [26], which are $8e^{\alpha}$- and $(2\alpha)^{\alpha}/2$-competitive, respectively. In the DVS problem, jobs come with release time $r(J)$, deadline $d(J)$, and a work requirement $p(J)$. A processor can run at speed $s \in [0, \infty)$ and consumes energy in a rate of s^{α}, for some $\alpha > 1$. The objective is to complete all jobs by their deadlines using the minimum total energy. The main differences of the DVS problem to the Grid problem include (i) jobs in DVS can be preempted while preemption is not allowed in the Grid problem; (ii) as processor speed in DVS can scale, a job can be executed for varying time duration as long as the total work is completed while in Grid a job must be executed for a fixed duration given as input; (iii) the work requirement $p(J)$ of a job J in DVS can be seen as $w(J) \times h(J)$ for the corresponding job in Grid.

Let $\mathcal{O}_D$ and $\mathcal{O}_G$ be the optimal algorithm for the DVS and Grid problems, respectively. Given a job set $\mathcal{J}_G$ for Grid, we can convert it into a job set $\mathcal{J}_D$ for DVS by keeping the release time and deadline for each job and setting the work requirement of a job in $\mathcal{J}_D$ to the product of the width and height of the corresponding job in $\mathcal{J}_G$.

▶ **Observation 1.** *Given a schedule S_G for $\mathcal{J}_G$, we can convert S_G into a feasible schedule S_D for $\mathcal{J}_D$ such that cost$(S_D(\mathcal{J}_D)) \leq$ cost$(S_G(\mathcal{J}_G))$; implying cost$(\mathcal{O}_D(\mathcal{J}_D)) \leq$ cost$(\mathcal{O}_G(\mathcal{J}_G))$.*

Note that it is not always possible to convert a feasible DVS schedule to a feasible Grid schedule. The observation does not immediately solve the Grid problem but as to be shown it provides a way to analyze algorithms for Grid.

3 Online algorithm for uniform width jobs

To handle jobs of arbitrary width and height, we first study the case when jobs have uniform width (all jobs have the same width $w \geq 1$). The proposed algorithm $\mathcal{UV}$ (Section 3.2) is based on a further restricted case of unit width, i.e., $w = 1$ (Section 3.1).

3.1 Unit width and arbitrary height

We present an online algorithm $\mathcal{V}$ which makes reference to an arbitrary feasible online algorithm for the DVS problem, denoted by $\mathcal{R}$. We require that the speed of $\mathcal{R}$ remains the same during any integral timeslot. When jobs have integral release times and deadlines, many known DVS algorithms satisfy this criteria, including $\mathcal{YDS}$, $\mathcal{BKP}$, and $\mathcal{AVR}$.

Recall in Section 2 how an input for the Grid problem is converted to an input for the DVS problem. We simulate a copy of $\mathcal{R}$ on the converted input and denote the speed used by $\mathcal{R}$ at t as $\ell(\mathcal{R}, t)$. Our algorithm makes reference to $\ell(\mathcal{R}, t)$ but not the jobs run by $\mathcal{R}$ at t.

Algorithm $\mathcal{V}$. For each timeslot t, schedule jobs to start at t until $\ell(\mathcal{V}, t)$ is at least $\ell(\mathcal{R}, t)$ or until all available jobs have been scheduled. Jobs are chosen in an EDF manner.

Analysis. Since $\mathcal{V}$ makes decision at integral time and jobs have unit width, each job is completed before any further scheduling decision is made. In other words, $\mathcal{V}$ is non-preemptive. To analyze the performance of $\mathcal{V}$, we note that $\mathcal{V}$ gives a feasible schedule (Lemma 2 (i)), and then analyze its competitive ratio (Theorem 3).

Let $h_{\max}(\mathcal{V}, t)$ be the maximum height of jobs scheduled at t by $\mathcal{V}$. We first classify each timeslot t into two types: (i) $h_{\max}(\mathcal{V}, t) < \ell(\mathcal{R}, t)$, and (ii) $h_{\max}(\mathcal{V}, t) \geq \ell(\mathcal{R}, t)$. We denote by $\mathcal{I}_1$ and $\mathcal{I}_2$ the union of all timeslots of Type (i) and (ii), respectively. Notice that $\mathcal{I}_1$ and $\mathcal{I}_2$ can be empty and the union of $\mathcal{I}_1$ and $\mathcal{I}_2$ covers the entire time line. Lemma 2 (ii) and (iii) bound the cost of $\mathcal{V}$ in each type of timeslots. By Lemma 2 and Observation 1, we obtain the competitive ratio of $\mathcal{V}$ in Theorem 3.

▶ **Lemma 2.** *(i) $\mathcal{V}$ gives a feasible schedule; (ii) cost$(\mathcal{V}, \mathcal{I}_1) \leq 2^\alpha \cdot$ cost$(\mathcal{R})$; (iii) cost$(\mathcal{V}, \mathcal{I}_2) \leq 2^\alpha \cdot$ cost$(\mathcal{O})$; and (iv) cost$(\mathcal{V})$ = cost$(\mathcal{V}, \mathcal{I}_1)$ + cost$(\mathcal{V}, \mathcal{I}_2)$.*

▶ **Theorem 3.** *Algorithm $\mathcal{V}$ is $2^\alpha \cdot (R + 1)$-competitive, where R is the competitive ratio of the reference DVS algorithm $\mathcal{R}$. $\mathcal{V}$ is $2^\alpha \cdot (8 \cdot e^\alpha + 1)$-competitive and $2^\alpha \cdot 2$-approximate when the algorithm $\mathcal{BKP}$ and $\mathcal{YDS}$ are referenced, respectively.*

3.2 Uniform width and arbitrary height

The idea of handling uniform width jobs is to treat them as if they were unit width, however, this would mean that jobs may have non-integral release times or deadlines. To remedy this, we define a procedure AlignFI to align the feasible intervals (precisely, release times and deadlines) to the new time unit.

Let $\mathcal{J}$ be a set of uniform width jobs each of width w. A job J is said to be *tight* if $|I(J)| \leq 2w$; otherwise, it is *loose*. Let $\mathcal{J}_{\mathrm{T}}$ and $\mathcal{J}_{\mathrm{L}}$ be the disjoint subsets of tight and loose jobs of $\mathcal{J}$, respectively. We design different strategies for tight and loose jobs. We observe that tight jobs can be handled easily by starting them at their release times. For any loose job, we modify it via Procedure AlignFI such that its release time and deadline is an integral multiple of w. With this alternation, we can treat the jobs as unit width and make scheduling decisions at time multiple of w.

Procedure AlignFI. Given a loose job set $\mathcal{J}_{\mathrm{L}}$ in which $w(J) = w$ and $|I(J)| > 2 \cdot w$ $\forall J \in \mathcal{J}_{\mathrm{L}}$. We define the procedure AlignFI to transform each loose job $J \in \mathcal{J}_{\mathrm{L}}$ into a job J' with release time and deadline "aligned" as follows: $r(J') \leftarrow \min_{i \geq 0}\{i \cdot w \mid i \cdot w \geq r(J)\}$; and $d(J') \leftarrow \max_{i \geq 0}\{i \cdot w \mid i \cdot w \leq d(J)\}$. We denote the resulting job set by $\mathcal{J}'$.

After AlignFI, the release time and deadline of each loose job are aligned to timeslot $i_1 \cdot w$ and $i_2 \cdot w$ for some integers $i_1 < i_2$. Hence, the job set $\mathcal{J}'$ can be treated as job set with unit width, where each unit has duration w instead of 1. We further observe that a feasible schedule of $\mathcal{J}'$ is also a feasible schedule of $\mathcal{J}_{\mathrm{L}}$.

Online algorithm $\mathcal{UV}$. The algorithm takes a job set $\mathcal{J}$ with uniform width w as input and schedules the jobs in $\mathcal{J}$ as follows. Let $\mathcal{J}_{\mathrm{T}}$ be the set of tight jobs in $\mathcal{J}$ and $\mathcal{J}_{\mathrm{L}}$ be the set of loose jobs in $\mathcal{J}$. Note that the decisions of $\mathcal{UV}$ can be made online.

1. For any tight job $J \in \mathcal{J}_{\mathrm{T}}$, schedule J to start at $r(J)$.
2. Loose jobs in $\mathcal{J}_{\mathrm{L}}$ are converted to $\mathcal{J}'$ by AlignFI. For $\mathcal{J}'$, we run Algorithm $\mathcal{V}$ in Section 3.1 with $\mathcal{BKP}$. Jobs are chosen in an earliest deadline first (EDF) manner.

The "inflexibility" of tight jobs guarantees that the simple strategy (Step 1 of $\mathcal{UV}$) gives good enough ratio. That is, since tight jobs have short feasible intervals, even the optimal schedule has to have high cost if our strategy has high load. On the other hand, AlignFI only increases the competitive factor of loose jobs by a constant factor because the feasible interval duration is only decreased by at most two thirds. The overall performance is:

▶ **Theorem 4.** *cost*$(\mathcal{UV}(\mathcal{J})) \leq 12^\alpha \cdot (8e^\alpha + 1) \cdot$ *cost*$(\mathcal{O}(\mathcal{J}))$.

4 Online algorithm for the general case

In this section, we present an algorithm $\mathcal{G}$ for jobs with arbitrary width and height. We first transform job set $\mathcal{J}$ to a "nice" job set $\mathcal{J}^*$ (to be defined) and show that such a transformation only increases the cost modestly. Furthermore, we show that for any nice job set $\mathcal{J}^*$, we can bound $\mathrm{cost}(\mathcal{G}(\mathcal{J}^*))$ by $\mathrm{cost}(\mathcal{O}(\mathcal{J}^*))$ and in turn by $\mathrm{cost}(\mathcal{O}(\mathcal{J}))$. Then we can establish the competitive ratio of $\mathcal{G}$.

4.1 Upper bound

A job J is said to be a *nice job* if $w(J) = 2^p$, for some non-negative integer p and a job set $\mathcal{J}^*$ is said to be a *nice job set* if all its jobs are nice. Note that the nice job J is in class C_p.

Procedure Convert. Given a job set $\mathcal{J}$, we define the procedure Convert to transform each job $J \in \mathcal{J}$ into a nice job J^*. We denote the resulting nice job set by $\mathcal{J}^*$ and the subset in C_p by $\mathcal{J}^*_p$. Suppose J is in class C_p. We modify it as follows: $w(J^*) \leftarrow 2^p$; $r(J^*) \leftarrow r(J)$; and $d(J^*) \leftarrow r(J^*) + \max\{d(J) - r(J), 2^p\}$.

We then define two procedures that transform schedules related to nice job sets.

Transformation RelaxSch. RelaxSch transforms a schedule S for a job set $\mathcal{J}$ into a schedule S^* for the corresponding nice job set $\mathcal{J}^*$ by moving the start and end time of every job J such that $st(S^*, J^*) = \min\{d(J^*) - w(J^*), st(S, J)\}$; and $et(S^*, J^*) = st(S^*, J^*) + w(J^*)$.

▶ **Observation 5.** *Consider any schedule S for $\mathcal{J}$ and the schedule S^* constructed by* RelaxSch *for the corresponding $\mathcal{J}^*$. We have $[st(S^*, J^*), et(S^*, J^*)] \subseteq [r(J^*), d(J^*)]$; in other words, S^* is a feasible schedule for $\mathcal{J}^*$.*

Transformation ShrinkSch. ShrinkSch converts a schedule S^* for a nice job set $\mathcal{J}^*$ to a schedule S for the corresponding $\mathcal{J}$. We set $st(S, J) \leftarrow st(S^*, J^*)$; and $et(S, J) \leftarrow st(S, J) + w(J)$, therefore, $et(S, J) \leq et(S^*, J^*)$. Let S^*_p be the partial schedule for class C_p.

Online algorithm $\mathcal{G}$. When a job J is released, it is converted to J^* by Convert and classified into one of the classes C_p. Jobs in the same class after Convert (being a uniform-width job set) are scheduled by $\mathcal{UV}$ independently of other classes. We then modify the execution time of J^* in $\mathcal{UV}$ to the execution time of J in $\mathcal{G}$ by ShrinkSch. Note that all these procedures can be done in an online fashion.

Using the results in Sections 3 and 4.1, we can compare the cost of $\mathcal{G}(\mathcal{J}_p)$ with $\mathcal{O}(\mathcal{J}^*_p)$ and $\mathcal{O}(\mathcal{J}^*_p)$ with $\mathcal{O}(\mathcal{J})$ for each class C_p. The most tricky part is in Lemma 6 (i). Intuitively, one can show that for each class, the load of $\mathcal{O}(\mathcal{J}^*_p)$ at any time is bounded above by that of $\mathcal{O}(\mathcal{J}_p)$ at the current time and $2^{p-1} - 1$ timeslots before and after the current time. This allows us to bound $\mathrm{cost}(\mathcal{O}(\mathcal{J}^*_p))$ by 3^α times of $\mathrm{cost}(\mathcal{O}(\mathcal{J}_p))$ (Lemma 6 (ii)).

▶ **Lemma 6.** *Consider any class C_p. (i) At any time t, $\ell(S^*_p, t) \leq \ell(S_p, t) + \ell(S_p, t - (2^{p-1} - 1)) + \ell(S_p, t + (2^{p-1} - 1))$. (ii) cost$(\mathcal{O}(\mathcal{J}^*_p)) \leq 3^\alpha \cdot$ cost$(\mathcal{O}(\mathcal{J}_p))$; (iii) cost$(\mathcal{O}(\mathcal{J}_p)) \leq$ cost$(\mathcal{O}(\mathcal{J}))$.*

▶ **Theorem 7.** *For any job set $\mathcal{J}$, cost$(\mathcal{G}(\mathcal{J})) \leq (36\lceil\log \frac{w_{\max}}{w_{\min}}\rceil)^\alpha \cdot (8e^\alpha + 1) \cdot$ cost$(\mathcal{O}(\mathcal{J}))$.*

4.2 Lower bound

We show a lower bound of competitive ratio for Grid problem with unit height and arbitrary width by designing an adversary for the problem. This lower bound is immediately a lower bound for the general case of Grid problem. Note that the lower bound in [23] may look similar but it does not work in our case since that adversary only guarantees a high peak load at a particular timeslot which is not sufficient for the total cost measurement.

Adversary $\varLambda$ and job instance $\mathcal{J}$. Given an online algorithm $\mathcal{A}$, a constant $\alpha > 1$ and a large number x, adversary $\varLambda$ outputs a set of jobs $\mathcal{J}$ with $\lfloor\alpha\rfloor + 1$ jobs. Let J_i be the ith job of $\mathcal{J}$. The adversary first computes a width for each job before running algorithm $\mathcal{A}$. It sets $w(J_{\lfloor\alpha\rfloor}) = x$, $w(J_{\lfloor\alpha\rfloor+1}) = x - 1$, and $w(J_i) = 3w(J_{i+1}) + 1$ for $1 \le i \le \lfloor\alpha\rfloor - 1$. Then adversary $\varLambda$ computes a release time and deadline for each job through a interaction with algorithm $\mathcal{A}$. For the first job J_1, adversary $\varLambda$ chooses any release time and deadline such that $d(J_1) - r(J_1) \ge 3w(J_1)$. For the ith job $J_i \in \mathcal{J}$ for $2 \le i \le \lfloor\alpha\rfloor + 1$, adversary $\varLambda$ sets $r(J_i) = st(\mathcal{A}, J_{i-1}) + 1$ and $d(J_i) = et(\mathcal{A}, J_{i-1})$. This limits algorithm $\mathcal{A}$ to fewer choices of start times for scheduling a new job. A job can only be scheduled in the execution interval of the previous job by algorithm $\mathcal{A}$. On the other hand, no two jobs have the same release time. Algorithm $\mathcal{A}$ shall schedule the jobs accordingly from J_1 to $J_{\lfloor\alpha\rfloor+1}$ and one job at a time.

▶ **Lemma 8.** *$cost(\mathcal{O}(\mathcal{J})) \le x \cdot 3^{\lfloor\alpha\rfloor}$.*

▶ **Theorem 9.** *For any deterministic online algorithm $\mathcal{A}$ for Grid problem with unit height and arbitrary width, adversary $\varLambda$ constructs an instance $\mathcal{J}$ such that $\frac{cost(\mathcal{A}(\mathcal{J}))}{cost(\mathcal{O}(\mathcal{J}))} \ge \left(\frac{1}{3}\log\frac{w_{\max}}{w_{\min}}\right)^{\alpha}$.*

5 Online algorithm for uniform height jobs

We consider (i) jobs with uniform-height h and unit-width and (ii) jobs with uniform-height h, arbitrary width and agreeable deadlines. To ease the discussion, we define the *density* of J, denoted by $\mathrm{den}(J)$, to be $\frac{h(J)*w(J)}{d(J)-r(J)}$. Roughly speaking, the density signifies the average load required by the job over its feasible interval. We then define the "average" load at any time t as $\mathrm{avg}(t) = \sum_{J:t\in I(J)} \mathrm{den}(J)$.

Basically, at any time t, $\mathcal{AVR}$ runs the processor at a speed which is the sum of the densities of jobs that are available at t. By Observation 1, we have the following corollary.

▶ **Corollary 10.** *For any input $\mathcal{J}_G$ and the corresponding input $\mathcal{J}_D$, $cost(\mathcal{AVR}(\mathcal{J}_D)) \le \frac{(2\alpha)^{\alpha}}{2} \cdot cost(\mathcal{O}_G)$.*

Main Ideas

The main idea is to make reference to the online algorithm $\mathcal{AVR}$ and consider two types of intervals, $\mathcal{I}_{>h}$ where the average load is higher than h and $\mathcal{I}_{\le h}$ where the average load is at most h. For the former, we show that we can base on the competitive ratio of $\mathcal{AVR}$ directly; for the latter, our load could be much higher than that of $\mathcal{AVR}$ and in such case, we compare directly to the optimal algorithm. Combining the two cases, we have Lemma 11, which holds for any job set. We show how we can use this lemma to obtain algorithms for the special cases. Notice that the number $\lceil\frac{\mathrm{avg}(t)}{h}\rceil$ is the minimum number of jobs needed to make the load at t at least $\mathrm{avg}(t)$.

▶ **Lemma 11.** *Suppose we have an algorithm $\mathcal{A}$ for any job set $\mathcal{J}$ such that (i) $\ell(\mathcal{A}, t) \le c \cdot \lceil avg(t)\rceil$ for all $t \in \mathcal{I}_{>h}$, and (ii) $\ell(\mathcal{A}, t) \le c'$ for all $t \in \mathcal{I}_{\le h}$. Then we have $cost(\mathcal{A}(\mathcal{J})) \le (\frac{(4c\alpha)^{\alpha}}{2} + c'^{\alpha}) \cdot cost(\mathcal{O}(\mathcal{J}))$.*

Uniform-height and unit-width

We consider jobs with uniform-height and unit-width, i.e., $w(J) = 1$ and $h(J) = h \ \forall J$. Note that such case is a subcase discussed in Section 3.1. Here we illustrate a different approach using the ideas above and describe the algorithm $\mathcal{UU}$ for this case. The competitive ratio of $\mathcal{UU}$ is better than that of Algorithm $\mathcal{V}$ in Section 3.1 when $\alpha < 3.22$.

Algorithm $\mathcal{UU}$. At any time t, choose $\lceil \frac{\text{avg}(t)}{h} \rceil$ jobs according to the EDF rule and schedule them to start at t. If there are fewer jobs available, schedule all available jobs.

▶ **Theorem 12.** *Algorithm $\mathcal{UU}$ gives feasible schedules and it is $(\frac{(4\alpha)^\alpha}{2} + 1)$-competitive.*

Uniform-height, arbitrary width and agreeable deadlines

We then consider jobs with agreeable deadlines. We first note that simply scheduling $\lceil \frac{\text{avg}(t)}{h} \rceil$ number of jobs may not return a feasible schedule.

To schedule these jobs, we first observe that for a set of jobs with total densities at most h, it is feasible to schedule them such that the load at any time is at most h. Roughly speaking, we consider jobs in the order of release, and hence in EDF manner since the jobs have agreeable deadlines. We keep the current ending time of all jobs that have been considered. As a new job is released, if its release time is earlier than the current ending time, we set its start time to the current ending time (and increase the current ending time by the width of the new job); otherwise, we set its start time to be its release time.

Using this observation, we then partition the jobs into "queues" each of which has sum of densities at most h. Each queue $\mathcal{Q}_i$ is scheduled independently and the resulting schedule is to "stack up" all these schedules. The queues are formed in a Next-Fit manner: (i) the current queue $\mathcal{Q}_q$ is kept "open" and a newly arrived job is added to the current queue if including it makes the total densities stays at most 1; (ii) otherwise, the current queue is "closed" and a new queue $\mathcal{Q}_{q+1}$ is created as open.

Algorithm $\mathcal{AD}$. The algorithm consists of the following components: InsertQueue, SetStartTime and ScheduleQueue.

InsertQueue: We keep a counter q for the number of queues created. When a job J arrives, if $\text{den}(J) + \sum_{J' \in \mathcal{Q}_q} \text{den}(J') \leq h$, then job J is added to $\mathcal{Q}_q$; otherwise, job J is added to a new queue $\mathcal{Q}_{q+1}$ and we set $q \leftarrow q + 1$.

SetStartTime: For the current queue, we keep a current ending time E, initially set to 0. When a new job J is added to the queue, if $r(J) \leq E$, we set $st(J) \leftarrow E$; otherwise, we set $st(J) \leftarrow r(J)$. We then update E to $st(J) + w(J)$.

ScheduleQueue: At any time t, schedule all jobs in all queues with start time set at t.

▶ **Lemma 13.** *(i) $\ell(\mathcal{AD}, t) \leq 3 \cdot h \cdot \lceil \frac{avg(t)}{h} \rceil$ for $t \in \mathcal{I}_{>h}$; (ii) $\ell(\mathcal{AD}, t) \leq h$ for $t \in \mathcal{I}_{\leq h}$.*

By Lemmas 11 and 13, we have Theorem 14 by setting $c = 3$ and $c' = 1$.

▶ **Theorem 14.** *For jobs with uniform height, arbitrary width and agreeable deadlines, $\mathcal{AD}$ is $(\frac{(12\alpha)^\alpha}{2} + 1)$-competitive. For jobs with uniform height, arbitrary width and same release time or same deadline, the competitive ratio can be improved to $(\frac{(8\alpha)^\alpha}{2} + 1)$ by using first-fit instead of next-fit for InsertQueue.*

6 Exact Algorithms

We first present some key notions. An algorithm with parameters $p_1, p_2, \ldots$ is said to be a *fixed parameter algorithm* if it runs in $f(p_1, p_2, \ldots) \cdot O(g(N))$ time for any function f and any polynomial function g, where N is the size of input. A problem is *fixed-parameter tractable (FPT)* if it can be solved by a fixed parameter algorithm. In this section, we show that the general case of the Grid problem is FPT with respect to a few parameters, and derive lower bounds of it.

We design two fixed parameter algorithms that are based on dynamic programming. Roughly speaking, we divide the timeline into k contiguous windows in a specific way, where each window W_i represents a time interval $[b_i, b_{i+1})$ for $1 \leq i \leq k$. The algorithm visits all windows accordingly from left to right and maintains a candidate set of schedules for the visited windows that no optimal solution is deleted from the set. The parameters of the algorithms emerge if we interpret the input as an "interval graph".

Interval graph. A graph $G = (V, E)$ is an *interval graph* if it captures the intersection relation for some set of intervals on the real line. Formally, for each $v \in V$, we can associate v to an interval I_v such that (u, v) is in E if and only if $I_u \cap I_v \neq \emptyset$. It has been shown in [13, 14] that an interval graph has a "consecutive clique arrangement", i.e., its maximal cliques can be linearly ordered in a way that for every vertex v in the graph, the maximal cliques containing v occur consecutively in the linear order. For any instance of the Grid problem, we can transform it into an interval graph $G = (V, E)$: For each job J with interval $I(J)$, we create a vertex $v(J) \in V$ and an edge is added between $v(J)$ and $v(J')$ if and only if $I(J)$ intersects $I(J')$. We can then obtain a set of maximal cliques in linear order, C_1, C_2, $\cdots$, C_k, by sweeping a vertical line from left to right, where k denotes the number of maximal cliques thus obtained. Our parameter, the maximum number of overlapped feasible intervals, is the maximum size of these maximal cliques.

Boundaries and windows. Based on the maximal cliques described above, we define some "windows" W_1, W_2, $\cdots$, W_k with "boundaries" b_1, b_2, $\cdots$, b_{k+1} as follows. We first give the definition of boundaries for the first algorithm. This definition will be generalized in Section 6.1 for the second algorithm. For $1 \leq i \leq k$, the i-th *boundary* b_i is defined as the earliest release time of jobs in clique C_i but not in cliques before C_i, precisely, $b_i = \min\{t \mid t = r(J) \text{ and } J \in C_i \setminus (\cup_{s=1}^{i-1} C_s)\}$. The rightmost boundary b_{k+1} is defined as the latest deadline among all jobs. With the boundaries, we partition the timeslots into contiguous intervals called *windows*. Figure 1 is an example of a set of jobs, its corresponding interval graph and maximal cliques.

6.1 Fixed parameter algorithms

Framework of the algorithms

We propose two exact algorithms, both run in k stages one for each of the k windows. We maintain a table T_{left} storing all "valid configurations" of jobs in all windows that have been considered so far. A row in the table consists of the configurations of all jobs. In addition, for each window W_i, we compute a table T_{right_i} to store all possible configurations of start and end time of jobs available in W_i. The configurations in T_{right_i} is then "concatenated" to some configurations in T_{left} that are "compatible" with each other. These merged configurations will be filtered to remove non-optimal ones. The remaining configurations become the new

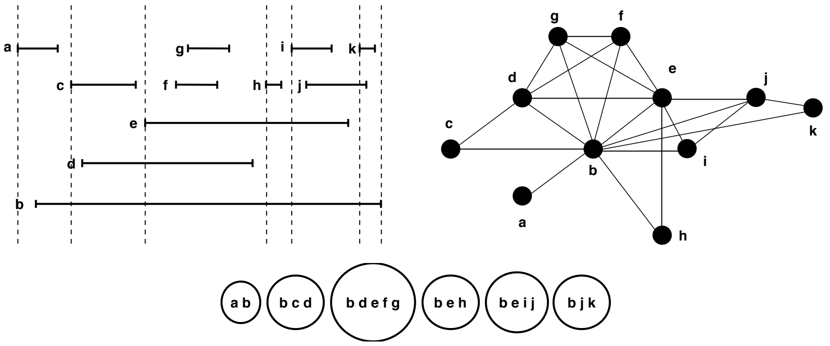


Figure 1 (Left) A set of jobs, where the horizontal line segments are the feasible time intervals of jobs and the vertical lines are boundaries of windows. (Right) An interval graph of the corresponding job set. (Bottom) A set of all the maximal cliques in the interval graph.

T_{left} for the next window. We denote by W_{left} the union of the windows corresponding to T_{left}. We drop the subscript i in T_{right_i} when the context is clear.

Configurations, validity, compatibility, concatenation. A *configuration* of job J in window W_i is an execution segment $[st_i(J), et_i(J))$. It is *valid* if $[st_i(J), et_i(J)) \subseteq [r(J), d(J))$. The cost of a window W_i with respect to some configurations is $\sum_{t \in W_i} (\sum_{J : t \in [st_i(J), et_i(J))} h(J))^{\alpha}$. Two configurations of the same job J are *compatible* if the union of the two configurations is a valid and contiguous execution segment with length exactly $w(J)$. To *concatenate* two configurations from two different windows, we make a union of the two configurations, and the corresponding cost is to add the costs of the two windows.

An algorithm with three parameters

Algorithm $\mathcal{E}$. We first transform the input job set $\mathcal{J}$ to an interval graph, and obtain the maximal cliques C_i for $1 \leq i \leq k$ and the corresponding windows W_i. We start with T_{left} containing the only configuration, which sets all the jobs to be not yet executed. Then we visit the windows from left to right with three procedures: ListConfigurations, ConcatenateTables and FilterTable.

ListConfigurations: For window W_i and jobs in C_i, we construct T_{right} storing all configurations of $J \in C_i$ with their cost in a brute force manner. We also delete the invalid configurations.

ConcatenateTables: We then concatenate compatible configurations in T_{left} and T_{right}. The resulting table is the new T_{left}. We also delete the invalid configurations in the new T_{left}.

FilterTable: After concatenation, we filter non-optimal configurations. We leave the configuration with the lowest cost among the configurations with the same execution segment.

After processing all the windows, the configuration with the lowest cost in the final T_{left} is returned as the solution. Note that a configuration is deleted only when it is invalid or its cost is higher than another configuration with the same execution segment for each job. Thus the algorithm outputs an optimal solution.

Let n to be the number of jobs, $w_{\max}$ to be the maximum width of jobs, m to be the maximum size of cliques, and $W_{\max}$ to be the maximum length of windows. Through a detail analysis, the running time of the three procedures depend mainly on $w_{\max}$, m and $W_{\max}$. So we have:

▶ **Theorem 15.** *Algorithm $\mathcal{E}$ computes an optimal solution in $f(w_{\max}, m, W_{\max}) \cdot O(n^2)$ time for some function f, i.e., the problem is FPT with respect to $w_{\max}$, m, and $W_{\max}$.*

An algorithm with two parameters

Algorithm $\mathcal{E}^+$. This algorithm is similar to algorithm $\mathcal{E}$ except the definitions of boundaries and ListConfigurations. Given a set of jobs $\mathcal{J}$, the algorithm uses the set of boundaries $\{r(J) \mid J \in \mathcal{J}\} \cup \{d(J) \mid J \in \mathcal{J}\}$ to construct the windows and obtain the corresponding cliques. For ListConfigurations, the number of timeslots we are considering for each window is at most a constant times the total width of all jobs in the window, i.e., when the window length is relatively larger than the total width of all jobs in the window, we consider much fewer than the window length. Let n to be the number of jobs, $w_{\max}$ to be the maximum width of jobs, and m to be the maximum size of cliques. We observed that the running time of the three procedures depend only on $w_{\max}$ and m. So we have:

▶ **Theorem 16.** *Algorithm $\mathcal{E}^+$ computes an optimal solution in $f(w_{\max}, m) \cdot O(n^2)$ time for some function f. Hence, the Grid problem is FPT with respect to $w_{\max}$ and m.*

6.2 Exact algorithm without parameter

For the case with unit width and arbitrary height of Grid problem, we can use Algorithm $\mathcal{E}$ to design an exact algorithm that its time complexity is only measured in the size of the input. In the case with unit width and arbitrary height, one may observe that the functionalities of the components of Algorithm $\mathcal{E}$ are not affected by the length of the windows. Without loss of generality, we assume that the number of timeslots τ is even. And we enforce all windows to have length 2. By this setting, the new algorithm runs in $O((\tau/2) \cdot 4^{2n} \cdot n)$ time where n is the number of jobs. Note that the input size N of the problem is $3n \log \tau + n \log h_{\max}$ where $h_{\max}$ is the maximum height over all jobs. Since $\log \tau = O(N)$, the running time becomes $2^{O(N)}$. Thus we have the following theorem.

▶ **Theorem 17.** *There is an exact algorithm running in $2^{O(N)}$ time for the Grid problem with unit width and arbitrary height where N is the length of the input.*

Jansen et al. [16] derived several lower bounds for scheduling and packing problems which can be used to develop lower bounds for our problem. Their lower bounds assume *Exponential Time Hypothesis* (ETH) holds, which conjectures that there is a positive real ϵ such that 3-SAT cannot be decided in time $2^{\epsilon n} N^{O(1)}$ where n is the number of variables in the formula and N is the length of the input. A lower bound for other problems can be shown by making use of strong reductions, i.e. reductions that increase the parameter at most linearly. Through a sequence of strong reductions, they obtain two lower bounds for PARTITION, $2^{o(n)} N^{O(1)}$ and $2^{o(\sqrt{N})}$ where n is the cardinality of the given set and N is the length of the input. We design a strong reduction from PARTITION to the decision version of Grid problem with unit width and arbitrary height. For each integer s in an integer set S, we convert it to a job J with $r(J) = 0$, $d(J) = 2$, $w(J) = 1$ and $h(J) = 2s$. We claim that S is a partition if and only if the set of jobs can be scheduled with cost at most $2(\sum_{s \in S} s)^{\alpha}$. By setting the length of the input or the number of jobs as the parameter, we have:

▶ **Theorem 18.** *There is a lower bound of $2^{o(\sqrt{N})}$ and a lower bound of $2^{o(n)} N^{O(1)}$ on the running time for the Grid problem unless ETH fails, where n is the number of jobs and N is the length of the input.*

7 Conclusion

We develop the first online algorithm with polylogarithm-competitive ratio and the first FPT algorithms for non-preemptive smart grid scheduling problem for the general case. Our algorithm in Section 4 relies on a classification into powers of 2. In the full paper, we show that one can classify into powers of $1+\lambda$, for $\lambda > 0$, and the competitive ratio only changes by a constant factor. We also discuss in the full paper why we base on the preemptive instead of non-preemptive DVS problem.

There are many future directions: different cost functions to capture varying electricity cost over time or measuring the maximum cost instead of the total [27]; jobs with varying power requests during its execution (it is a constant value in this paper); other objectives like response time. A preliminary result is that we can extend our online algorithm to the case where a job may have varying power requests during its execution, in other words, a job can be viewed as having rectilinear shape instead of being rectangular. In such case, the competitive ratio is increased by a factor which measures the maximum height to the minimum height ratio of a job (see the full paper for more details).

References

1 Soroush Alamdari, Therese Biedl, Timothy Chan, Elyot Grant, Krishnam Jampani, Srinivasan Keshav, Anna Lubiw, and Vinayak Pathak. Smart-grid electricity allocation via strip packing with slicing. In *WADS*, pages 25–36, 2013.

2 Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

3 Yossi Azar. On-line load balancing. In *Online Algorithms*, pages 178–195, 1998.

4 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

5 Paul C. Bell and Prudence W. H. Wong. Multiprocessor speed scaling for jobs with arbitrary sizes and deadlines. *J. Comb. Optim.*, 29(4):739–749, 2015.

6 Mihai Burcea, Wing-Kai Hon, Hsiang-Hsuan Liu, Prudence W. H. Wong, and David K. Y. Yau. Scheduling for electricity cost in smart grid. In *COCOA*, pages 306–317, 2013.

7 Chen Chen, K. G. Nagananda, Gang Xiong, Shalinee Kishore, and Lawrence V. Snyder. A communication-based appliance scheduling scheme for consumer-premise energy management systems. *IEEE Trans. Smart Grid*, 4(1):56–65, 2013.

8 Lin Chen, Nicole Megow, and Kevin Schewior. An $O(\log m)$-competitive algorithm for online machine minimization. In *SODA*, pages 155–163, 2016.

9 Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *IFIP TCS*, pages 209–222, 2004.

10 Marijana Živić Djurović, Aleksandar Milačić, and Marko Kršulja. A simplified model of quadratic cost function for thermal generators. In *DAAAM*, pages 25–28, 2012.

11 Kan Fang, Nelson A. Uhan, Fu Zhao, and John W. Sutherland. Scheduling on a single machine under time-of-use electricity tariffs. *Annals OR*, 238(1-2):199–227, 2016.

12 Xin Feng, Yinfeng Xu, and Feifeng Zheng. Online scheduling for electricity cost in smart grid. In *COCOA*, pages 783–793, 2015.

13 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

14 Rudolf Halin. Some remarks on interval graphs. *Combinatorica*, 2(3):297–304, 1982.

15 Katherine Hamilton and Neel Gulhar. Taking demand response to the next level. *IEEE Power and Energy Mag.*, 8(3):60–65, 2010.

16 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. In *WADS*, pages 439–450, 2013.

17 Iordanis Koutsopoulos and Leandros Tassiulas. Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In *Proc. e-Energy*, pages 41–50, 2011.

18 Fu-Hong Liu, Hsiang-Hsuan Liu, and Prudence W. H. Wong. Optimal nonpreemptive scheduling in a smart grid model. *CoRR*, abs/1602.06659, 2016.

19 Thillainathan Logenthiran, Dipti Srinivasan, and Tan Shun. Demand side management in smart grid using heuristic optimization. *IEEE Trans. Smart Grid*, 3(3):1244–1252, 2012.

20 T. Lui, W. Stirling, and H. Marcy. Get smart. *IEEE Power & Energy Mag.*, 8(3):66–78, 2010.

21 Sabita Maharjan, Quanyan Zhu, Yan Zhang, Stein Gjessing, and Tamer Basar. Dependable demand response management in the smart grid: A stackelberg game approach. *IEEE Trans. Smart Grid*, 4(1):120–132, 2013.

22 Gilbert Masters. *Renewable and efficient electric power systems.* John Wiley & Sons, 2013.

23 Barna Saha. Renting a cloud. In *FSTTCS*, pages 437–448, 2013.

24 Sergio Salinas, Ming Li, and Pan Li. Multi-objective optimal energy consumption scheduling in smart grids. *IEEE Trans. Smart Grid*, 4(1):341–348, 2013.

25 US Department of Energy. The Smart Grid: An Introduction. `http://www.oe.energy.gov/SmartGridIntroduction.htm`, 2009.

26 F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

27 Sean Yaw and Brendan Mumey. An exact algorithm for non-preemptive peak demand job scheduling. In *COCOA*, pages 3–12, 2014.

28 Zpryme Research & Consulting. Power systems of the future: The case for energy storage, distributed generation, and microgrids, 2012.

Distributed and Robust Support Vector Machine*

Yangwei Liu[1], Hu Ding[2], Ziyun Huang[3], and Jinhui Xu[4]

1 Department of Computer Science and Engineering, State University of New York at Buffalo, US
2 Department of Computer Science and Engineering, Michigan State University, US
3 Department of Computer Science and Engineering, State University of New York at Buffalo, US
4 Department of Computer Science and Engineering, State University of New York at Buffalo, US

Abstract

In this paper, we consider the distributed version of Support Vector Machine (SVM) under the coordinator model, where all input data (*i.e.,* points in $\mathbb{R}^d$ space) of SVM are arbitrarily distributed among k nodes in some network with a coordinator which can communicate with all nodes. We investigate two variants of this problem, with and without outliers. For distributed SVM without outliers, we prove a lower bound on the communication complexity and give a distributed $(1-\epsilon)$-approximation algorithm to reach this lower bound, where ϵ is a user specified small constant. For distributed SVM with outliers, we present a $(1-\epsilon)$-approximation algorithm to explicitly remove the influence of outliers. Our algorithm is based on a deterministic distributed top t selection algorithm with communication complexity of $O(k \log(t))$ in the coordinator model. Experimental results on benchmark datasets confirm the theoretical guarantees of our algorithms.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Distributed Algorithm, Communication Complexity, Robust Algorithm, SVM

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.54

1 Introduction

Training a *Support Vector Machine (SVM)* [5] in a distributed setting is a commonly encountered problem in the big data era. This could be due to the fact that the data set is too large to be stored in a centralized site (*e.g.,* bioinformatics data [15]), or simply because the data set is collected in a distributed environment (*e.g.,* wireless sensor network data [22]). In many scenarios, large volume data transmission between different sites could be rather expensive and time consuming. In some applications, this may not even be allowed due to privacy concerns. Thus efficient distributed SVM algorithms are needed to minimize the communication cost and meanwhile preserve the quality of solution.

In recent years, a significant amount of effort has been devoted to this problem ([9, 3, 12, 14, 4, 16, 8, 18, 6]), and a number of distributed SVM algorithms with different strength have been developed. However most of them are still suffering from various limitations, such as high communication complexity, sub-optimal quality of solution, and slow convergence

* The research of this work was supported in part by NSF through grants CCF-1422324, IIS-1422591, and CNS-1547167. Ding was supported by a start-up fund from Michigan State University.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 54; pp. 54:1–54:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

rate. For instance, one type of extensively studied algorithms in recent years are the family of *incremental construction* algorithms [9, 3].

Such algorithms often have good performance in practice and some other nice features related to robustness and decentralization; but they generally do not have theoretical guarantee on the communication complexity, and some of them even have no quality guarantee on their solutions. Another type of popular algorithms are those which parallelize existing centralized algorithms ([12, 14, 4]). These algorithms typically focus on enhancing the ability of dealing with extremely large size data sets, but in general have no quality guarantee on communication complexity. There are also another family of algorithms called distributed stochastic gradient descent algorithms [20]; the main issue of such algorithms is that their running time (or number of iteration) is mostly sub-optimal, and they do not have a guarantee on communication cost. Very recently, there is an interesting result [2] which presents a similar lower bound on communication cost. However their lower bound applies only to those coreset-based algorithms, not the general algorithms, whereas the lower bound result in this paper is applicable to any distributed SVM algorithm.

From a geometric point of view, training an SVM can be interpreted as finding a hyperplane that separates two classes of points while maximizing the separating margin. It is also well known to be equivalent to computing the polytope distance of the two point sets. Recent research [10] shows that we can find an $(1-\epsilon)$-approximation of the polytope distance using Gilbert algorithm with a running time linearly depending on the input size. Roughly speaking, Gilbert algorithm is a gradient descent procedure that in each step greedily computes an optimal direction along which the primal solution should improve.

In this paper we present a distributed SVM algorithm that is theoretically guaranteed to have the lowest possible communication cost together with a guaranteed near-optimal solution, based on the classical Gilbert algorithm [11]. Comparing to previous distributed SVM algorithms, our algorithm has several advantages. (1) Our algorithm has a communication complexity which is theoretically guaranteed to reach the lower bound; (2) it does not make any assumption on the input data and its distribution; (3) its running time is only linearly dependent on the input size; and (4) it produces a $(1-\epsilon)$-approximation for the problem which is sparse (*i.e.,* the number of support vectors is small).

Since SVM is well known to be sensitive to outliers, we also consider the case of distributed SVM with outliers. We show that it is possible to explicitly avoid the influence of outliers in distributed settings by using a combinatorial tool called *Random Gradient Descent (RGD)* tree [7] to achieve a $(1-\epsilon)$-approximation on the quality of solution and meanwhile significantly reduce the communication cost. An underlying technique used for reducing the communication cost is an algorithm for the distributed selection problem (*i.e.,* finding the t-th smallest number from a set of numbers distributed in k sites). This problem has been extensively studied in the past ([17, 21, 13]). The best result (in terms of communication cost) is a randomized algorithm [13] which has a communication complexity of $O(k\log(t))$. The best deterministic algorithm has a communication complexity of $O(k\log^2(t))$. In this paper we give a deterministic algorithm with communication complexity $O(k\log(t))$ in the coordinator model.

In this paper We will concentrate on one-class SVM first since it captures the main difficulty of the problem, and then extend the results to two-class SVM (in the full version of this paper). We also performed experiments on benchmark datasets to evaluate the performance of the algorithms (in the full version of the paper).

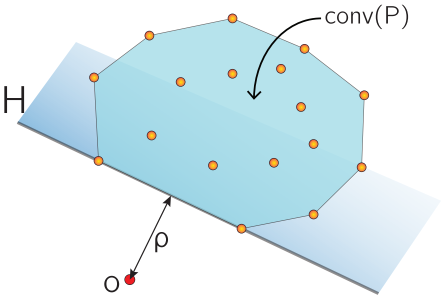


Figure 1 One-class SVM is equivalent to the polytope distance problem. ρ is the polytope distance, H is the separating hyperplane, with distance to o exactly ρ.

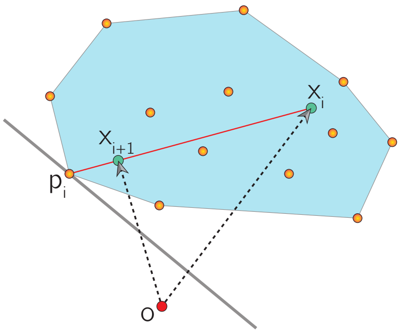


Figure 2 One step of the Gilbert Algorithm, updating x_i to x_{i+1}.

2 Preliminaries

2.1 Equivalence between SVM and Polytope Distance

In this section, we give several definitions which will be used throughout the paper.

▶ **Definition 1.** (One-class SVM): Given a point set $P \subseteq \mathbb{R}^d$, find a hyperplane H separating the origin o and P such that the separating margin (*i.e.*, the distance between o and H) is maximized.

It is well known that this problem is equivalent to the following problem of computing the polytope distance:

▶ **Definition 2.** (Polytope distance): Given a point set $P \subseteq \mathbb{R}^d$, compute the shortest distance between the origin o and a point p in the polytope $conv(P)$ (*i.e.*, the convex hull of P).

▶ **Lemma 3** ([10]). *Given a point x which realizes the polytope distance of $P \subseteq \mathbb{R}^d$, the hyperplane H passing x and orthogonal to $\overline{ox}$ is the maximum separating hyperplane of P. In other words, the polytope distance problem is equivalent to the one-class SVM problem.*

Figure 1 provides an intuitive explanation of the equivalence between one-class SVM and polytope distance. Notice that the polytope distance problem can be formulated as a standard convex quadratic optimization problem, thus can be solved optimally using standard techniques in $O(n^3)$ time. However this approach is mostly impractical because of the high time complexity. Actually in many applications, a near-optimal approximate solution is sufficient. Thus it is desirable to design efficient approximation algorithms. A $(1-\epsilon)$-*approximation* of the polytope distance can be defined as follows.

▶ **Definition 4.** ($(1-\epsilon)$-approximation of polytope distance): $x \in conv(P)$ achieves a $(1-\epsilon)$-approximation of the polytope distance problem, if

$$\|x\| - p|_x \leq \epsilon\|x\|, \forall p \in P$$

where $p|_x := \frac{\langle p,x\rangle}{\|x\|}$ is the signed length of the projection of p onto vector $\overline{ox}$.

In the rest of this paper, we also call the point x, rather its distance to o, as the approximation. This is only for ease of discussion, and does not affect the solution at all (since we can get the actual distance by simply computing the distance between x and o).

2.2 Gilbert Algorithm

Gilbert algorithm can be used to find a $(1-\epsilon)$-approximation of the polytope distance problem. Roughly speaking, Gilbert algorithm starts with an initial solution x_1 being the closest point of P to the origin. In step i, the algorithm finds the point $p_i \in P$ that has the smallest projection distance to vector $\overline{ox_i}$, and picks the point on the line segment $[p_i, x_i]$ that is closest to the origin as x_{i+1}. Figure 2 illustrates one step of Gilbert algorithm.

Algorithm 1 Gilbert Algorithm

```
1: INPUT : A $d$ dimensional point set $P$, the origin $o$.
2: OUTPUT : A $(1-\epsilon)$-approximation of the polytope distance of $P$ to $o$.
3: Initialize $i$ to be 1. Find the point $p_1$ that is closest to $o$, let $x_1 = p_1$
4: In step $i$, let $p_{i+1}$ be the point that has the smallest $p|_{x_i}$, let $x_{i+1}$ be the point that is
   closest to $o$ on line segment $[x_i, p_{i+1}]$.
5: Return $x_{i+1}$ when it is a $(1-\epsilon)$-approximation. Otherwise goto Line 4 with $i = i + 1$.
```

Despite its simplicity, it has been shown [10] that Gilbert algorithm can actually find such a $(1-\epsilon)$-approximation in $O(\frac{1}{\epsilon})$ steps. Formally, we have the following theorem.

▶ **Theorem 5** ([10]). *Gilbert algorithm succeeds after at most $O(\frac{1}{\epsilon})$ steps.*

Since each step of the algorithm involves only computing the projection of all points in P to a specific direction, which can be done in linear time, Gilbert algorithm has a total running time linearly depending on $n = |P|$ which is the number of input points.

Besides the fast running time, Gilbert Algorithm (and its variants) has also many other properties that make it a good SVM solver.

1. The algorithm works for arbitrary kernels. Since the algorithm only requires the computation of projection distance, which can be obtained by scalar product, we only need one kernel evaluation at each time when a projection distance is computed. This also implies that our proposed algorithms can be trivially kernelized.
2. The result is sparse, or in other words, the number of support vectors is small. In the kernelized version, the solution x_i is a linear combination of input points, and we can easily observe that x_i involves at most one more point in each step, so the total number of support vectors is $O(\frac{1}{\epsilon})$.

3 Communication Complexity of Distributed SVM

In a distributed setting, the point set P is arbitrarily distributed among k nodes. Based on different modes of communication, there are different models to be considered. In this paper we mainly study the widely used coordinator model which contains an extra coordinator node, and all other nodes are only allowed to communicate with the coordinator. The algorithm itself can be easily modified for a more general communication model, *e.g.*, network of general topology. Such generalization will at most induce an additional $O(D)$ (where D is the diameter of the network) [1] increase on the communication complexity.

We are interested in determining the minimum amount of communication that is needed to find a $(1-\epsilon)$-approximation of the polytope distance problem. We define the communication cost as the number of points needed to be transfered between nodes, where transferring one d-dimensional point (together with an optional $O(d)$ constants) takes $O(1)$ communication. This way of defining the communication cost simplifies the proof of Theorem 6. Below is our lower bound result.

▶ **Theorem 6.** *A $(1-\epsilon)$-approximation of the distributed polytope distance problem requires $\Omega(kd)$ communication for any $\epsilon < \frac{\sqrt{17}-4}{16d}$.*

We prove Theorem 6 by giving a reduction from the following *k-OR* problem.

▶ **Definition 7** (k-OR)**.** Given k players with each holding an n-bit binary vector, find the bitwise OR of all k vectors.

An example of the k-OR problem can be like the following: Player 1 holds vector $(1,0,0,1,0)$, Player 2 holds $(0,0,0,1,1)$, and Player 3 holds $(1,1,0,0,1)$. Then the output should be the vector $(1,1,0,1,1)$, where each bit is just the OR of the same bit of all players' vectors. For the communication complexity of this problem we have the following result.

▶ **Lemma 8** ([19])**.** *k-OR problem requires $\Omega(nk)$ communication in the coordinator model.*

Proof of Theorem 6. We prove Theorem 6 by reducing the k-OR problem to the problem of finding an ϵ-approximation of distributed polytope distance.

Given an instance of the k-OR problem with k players each holding an n-bit binary vector, construct an instance of distributed polytope distance in $d = n$ dimensional space with k nodes as follows:

For player i, for $j = 1, \cdots, d$, if the j'th bit of his vector is 0, add point e_j to node i's point set; otherwise add point λe_j to its point set, where $e_j = (\underbrace{0, \cdots, 0}_{\text{j-1}}, 1, \underbrace{0, \cdots, 0}_{\text{d-j}})$ is the d-dimensional point whose only non-zero entry is the j'th coordinate with value 1, and λ is a constant smaller than 1 to be determined later.

In this construction, each node holds exactly d points, and there are kd points in total. Since for the j'th point there are only two possible positions to place it, *i.e.*, e_j or λe_j, there will be points from different nodes sharing the same position. This does not affect the proof, since we can add a small enough perturbation to points sharing the same location.

▶ **Definition 9** (Configuration)**.** A configuration C of an instance of the polytope distance problem constructed as above is a size d point set, where the i'th point is λe_i if there is at least one λe_i in the point sets of all nodes; otherwise the i'th point is set to be e_i. The **order** of a configuration C is the number of λe it contains.

It is easy to see that a configuration encodes the solution of the corresponding k-OR problem. So if we can find out the configuration based on an ϵ-approximation solution of the distributed polytope distance problem, we can solve the k-OR problem, thus proving Theorem 6. The following lemma ensures that we can indeed achieve that.

▶ **Lemma 10.** *If $\epsilon < \frac{\sqrt{17}-4}{16d}$, it is possible to determine the configuration purely based on an ϵ-approximation of the distributed polytope distance problem with λ satisfying $\frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}} < \lambda < \min\{\sqrt{1 - d(1-(1-\epsilon)^2)}, (\frac{1}{2} + \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}})\}$.*

Notice that $\epsilon < \frac{\sqrt{17}-4}{16d}$ guarantees that $\frac{1}{4} - \frac{\epsilon d}{1-\epsilon} > 0$, $1 - d(1-(1-\epsilon)^2) > 0$, as well as $\frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}} < \sqrt{1 - d(1-(1-\epsilon)^2)}$. Thus such a λ always exists. Denote the polytope distance of a configuration C as $\rho(C)$, the order of C as *order*(C). We call the set of configurations with the same order d' as an ***order*-d' *layer*** of configurations. We prove Lemma 10 in two steps:

- **Claim 1**: *order*(C_i) = *order*(C_j) $\implies$ $\rho(C_i) = \rho(C_j)$; *order*(C_i) > *order*(C_j) $\implies$ $\rho(C_i) < (1-\epsilon)\rho(C_j)$.

- **Claim 2**: a solution x can't be an ϵ-approximation for more than one configuration in the same layer.

Proof of Claim 1. Consider a configuration $C = \{p_1, p_2, \cdots, p_d\}$ with order d'. This means that there are $d - d'$ points with their only non-zero coordinate as 1, and d' points with their non-zero coordinate as λ. Suppose that the point $x = \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_d p_d$ on $conv(C)$ is the closest point to the origin. We observe that for $i \in [1, d]$, we have $0 < \alpha_i < 1$. This can be proved by contradiction as follows. First of all, since x is on $conv(C)$, we naturally have $0 \leq \alpha_i \leq 1$ and $\sum_i \alpha_i = 1$. Suppose that there exists an $\alpha_l = 0$. This means that the l'th coordinate of x is 0, and thus x is on the simplex spanned by $C - \{p_l\}$. Notice that in this case, we have $\overline{op_l}$ perpendicular to the simplex spanned by $C - \{p_l\}$. Thus, $\angle oxp_l < \frac{\pi}{2}$ and $\angle op_l x < \frac{\pi}{2}$, which means that the projection of o onto $\overline{xp_l}$ is within the line segment $\overline{xp_l}$, resulting in a point closer to o than x, contradicting the fact that x is the closest point to o. Then $\alpha < 1$ follows immediately.

The above observation guarantees that the closest point to o is always within $conv(C)$, instead of on the boundary. Now let us compute $\rho(C)$.

We partition the points of C into two subsets based on their non-zero coordinates. C_1 contains points whose non-zero coordinates are 1, and C_λ contains the rest. By the symmetry of the dimensions, we can safely assume that C_1 contains the first $d - d'$ points, and C_λ contains the latter d' points without loss of generality. Now let $a = \frac{1}{(d-d')\lambda + d'\frac{1}{\lambda}}$, and consider the point $x = (\underbrace{\lambda a \cdot 1, \cdots, \lambda a \cdot 1}_{d-d'}, \underbrace{\frac{a}{\lambda} \cdot \lambda, \cdots, \frac{a}{\lambda} \cdot \lambda}_{d'})$. It is clear that (1) x is within the boundary of $conv(C)$, since $(d - d')\lambda a + d'\frac{a}{\lambda} = 1$; and (2) the projection distance of any point in C onto $\overline{ox}$ is $\frac{\lambda a}{\|x\|}$. Thus $\overline{ox}$ is perpendicular to the subspace spanned by C. Combining the above two facts, we know that x is the closest point to o on $conv(C)$, and $\rho(C) = \|\overline{ox}\| = \sqrt{\frac{1}{(d-d') + \frac{d'}{\lambda^2}}}$. Since $\rho(C)$ depends only on the order of the configuration, we have proved the first half of Claim 1.

Since each layer of configuration has the same polytope distance, we let $\rho_{d'}$ denote the polytope distance for order-d' layer. Now for the second half of Claim 1, let us consider two consecutive layers with order d' and $d' + 1$. Then we have

$$\begin{aligned}
\frac{\rho_{d'+1}}{\rho_{d'}} &= \sqrt{\frac{d + (\frac{1}{\lambda^2} - 1)d'}{d + (\frac{1}{\lambda^2} - 1)(d' + 1)}} \\
&\leq \sqrt{\frac{d - (1 - \lambda^2)}{d}} \qquad (1) \\
&< \sqrt{\frac{d - (1 - \sqrt{1 - d(1 - (1 - \epsilon)^2)}^{-2})}{d}} \\
&= 1 - \epsilon \qquad (2)
\end{aligned}$$

in which inequality (1) holds because of $\frac{1}{\lambda^2} - 1 > 0$. Using (2), we immediately have $\frac{\rho_{d'+t}}{\rho_{d'}} < (1 - \epsilon)^t < 1 - \epsilon$. Thus the second part of Claim 1 is proved. ◀

Proof of Claim 2. In order to prove Claim 2, we first make another observation of the ϵ-approximation x of configuration $C = \{p_1, \cdots, p_d\}$ of order d'. Since x is an ϵ-approximation, by definition it has to be on $conv(C)$. So x takes the form of $x = \sum \alpha_i p_i$, and $\sum \alpha_i = 1$. Also by definition, we have $p_i|_x > (1 - \epsilon)\|x\|$. Together with the definition of $p_i|_x$, we have

$$\alpha_i \geq \begin{cases} (1 - \epsilon)\|x\|^2 & \text{, if } p_i = e_i \\ \frac{1-\epsilon}{\lambda^2}\|x\|^2 & \text{, otherwise.} \end{cases}$$

Then we have for $p_i = e_i$,

$$\begin{aligned}\alpha_i &= 1 - \sum_{j \neq i} \alpha_j \\ &\leq 1 - (\sum_{j\neq i, p_j = e_j} (1-\epsilon)\|x\|^2 + \sum_{j \neq i, p_j = \lambda e_j} \frac{(1-\epsilon)}{\lambda^2}\|x\|^2) \\ &= 1 - ((d-d'-1)(1-\epsilon)\|x\|^2 + d'\frac{1-\epsilon}{\lambda^2}\|x\|^2) \\ &= (\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}))\|x\|^2. \qquad (3)\end{aligned}$$

Now, suppose that x is an ϵ-approximation for two configurations C_1 and C_2 in the same layer. Since all configurations in the same layer have the same order, this indicates that they have the same number of points whose non-zero coordinate is 1 and the same number of points whose non-zero coordinate is λ. This means that C_1 and C_2 differs by at least two points, with different non-zero coordinate. Denote the index of such pair of points as i and j. W.O.L.G., assume that in C_1 the i'th point $p_i^{(1)} = e_i$, and the j'th point $p_j^{(1)} = \lambda e_j$. Then in C_2, the i'th point $p_i^{(2)} = \lambda e_i$, and the j'th point $p_j^{(2)} = e_j$. Since x is an ϵ-approximation of C_1, it has to take the form of $x = \sum \alpha_i p_i^{(1)}$,, and $p_i^{(1)} = e_i$, following (3) we have

$$\alpha_i \leq (\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}))\|x\|^2.$$

Since x is also an ϵ-approximation of C_2, it has to satisfy

$$\frac{\lambda\alpha_i}{\|x\|} = p_i^{(2)}|_x \geq (1-\epsilon)\|x\|.$$

Together we have

$$\frac{1-\epsilon}{\lambda}\|x\|^2 \leq \alpha_i \leq (\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}))\|x\|^2,$$

which means that

$$\frac{1-\epsilon}{\lambda} \leq \frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}).$$

However, since $\frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}} < \lambda < \frac{1}{2} + \sqrt{\frac{1}{4} - \frac{\epsilon d}{1-\epsilon}}$, and $\|x\|^2 \geq \rho_{d'}^2 = \frac{1}{(d-d') + \frac{d'}{\lambda^2}}$, we always have

$$\begin{aligned}\frac{1}{\|x\|^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}) &\leq (d-d') + \frac{d'}{\lambda^2} - ((d-d'-1)(1-\epsilon) + d'\frac{1-\epsilon}{\lambda^2}) \\ &= d - (d-1)(1-\epsilon) + \epsilon(\frac{1}{\lambda^2} - 1)d' \\ &\leq d - (d-1)(1-\epsilon) + \epsilon(\frac{1}{\lambda^2} - 1)d \\ &= 1 - \epsilon + \frac{\epsilon d}{\lambda^2} \\ &< \frac{1-\epsilon}{\lambda},\end{aligned}$$

which is a contradiction. Therefore x cannot be an ϵ-approximation for two configurations of the same layer. ◀

Now we are ready to prove Lemma 10. At the coordinator, pre-compute all possible $\rho_{d'}$, with $d' = (0, 1, \cdots, d)$. Compare the polytope distance of the ϵ-approximation x to every segment $[\rho, \frac{1}{1-\epsilon}\rho]$. If $\|\overline{ox}\|$ falls within the segment corresponding to $\rho_{d'}$, then Claim 1 guarantees that the configuration that we want is in the layer with order d'; Then for all configurations C in that layer, check if x is an ϵ-approximation for C by testing whether for all $p \in C$, $p|_x \geq (1-\epsilon)\|x\|$. Since x is an ϵ-approximation, so there is at least one configuration C that will pass the test; and Claim 2 guarantees that there is only one such C. Return C as the desired configuration. This completes the proof for Lemma 10

Lemma 10 means that if we solve the approximate version of the distributed polytope distance problem, we can solve the k-OR problem. Hence the communication complexity of the approximate distributed SVM is $\Omega(kd)$, proving Theorem 6. ◀

Now we are ready to give the distributed version of Gilbert Algorithm (*i.e.,* Algorithm 2) in the coordinator model.

Algorithm 2 Distributed Gilbert Algorithm

```
1: INPUT : A $d$ dimensional point set $P$ arbitrarily distributed among $k$ nodes.
2: OUTPUT(by the coordinator) : A $(1-\epsilon)$-approximation of the polytope distance of $P$ to $o$.
3: Initialize $i = 1$; All nodes send to the coordinator its closest point to $o$; The coordinator picks the global closest point to $o$ as $p_1$;
4: In step $i$, the coordinator sends $x_i$ to all nodes; upon receiving $x_i$, each node picks one of its points that has the smallest $p|_{x_i}$ and send back to the coordinator; the coordinator picks the point that has the smallest $p|_{x_i}$ based on the points it received; Denote this point as $p_{i+1}$ and find $x_{i+1}$ as in non-distributed version.
5: Return $x_{i+1}$ when it is a $(1-\epsilon)$-approximation. Otherwise go to Line 4 with $i = i + 1$.
```

In each step of Algorithm 2, the coordinator sends the current solution x_i to all k nodes. Upon receiving x_i, each node computes the smallest projection distance onto vector $\overline{ox_i}$ based on its own share of points, and return it to the coordinator. The coordinator picks the point that has the smallest projection distance onto vector x_i among all returned points, and uses it as p_{i+1}. Each step of the algorithm involves one round of communication between the coordinator and all k nodes. Thus the communication of each step is $O(k)$. By Theorem 5, we have the following theorem.

▶ **Theorem 11.** *The communication complexity of Algorithm 2 is $O(\frac{k}{\epsilon})$ in the coordinator model.*

In the construction of the proof of Theorem 6 we can take $\epsilon = \Theta(\frac{1}{d})$, resulting in a communication cost of $\Theta(kd)$ for Algorithm 2. Suppose that there exists an algorithm with asymptotically smaller communication cost than Algorithm 2, it will also solve k-OR problem using $o(kd)$ communication, contradicting Lemma 8. So there doesn't exist an algorithm that computes a $(1-\epsilon)$ approximation with asymptotically smaller communication than Algorithm 2.

4 Robust Distributed SVM

In this section we present an algorithm for explicitly avoiding the influence of outliers in the distributed SVM problem. We first consider the following problem.

▶ **Definition 12** (Distributed One-class SVM with Outliers)**.** Given P as a point set of size n in d dimensional space that is arbitrarily distributed among k nodes, and γ as the fraction of outliers in P, find the subset $P^{'} \subseteq P$ of size $(1-\gamma)n$ so that the margin separating the origin and $P^{'}$ is maximized.

Notice that in this problem setting, there are two factors that need to be considered when evaluating the quality of the approximation result: the width of the margin, and the number of outliers accurately pruned out. Thus we need a new definition of approximation.

▶ **Definition 13.** For two constants $\epsilon, \delta > 0$, a margin M is an (ϵ, δ)-approximation, if the width of M is larger than or equal to $(1-\epsilon)\rho$, and the number of outliers identified by M is no more than $(1+\delta)\gamma|P|$.

In this problem, we aim to prune out exactly γn points (as outliers) so that the rest of the points can be separated from the origin by the largest possible margin. In this scenario, Gilbert algorithm may perform arbitrarily bad. This is because in each step, Gilbert algorithm finds the point that has the minimum projection distance and there is a possibility that this point happens to be an outlier. As a gradient descent procedure, Gilbert algorithm does not have the ability to recover from the negative impact of picking an outlier. To avoid this problem, a key observation is that Gilbert algorithm does not need to always identify the point with the minimum projection distance; it is actually sufficient to find one point (to maintain a fast convergence rate) as long as its projection distance is one of the w smallest for some w to be determined later. Based on this observation, Ding and Xu [7] has developed a new framework, called *Random Gradient Descent (RGD)* tree, to explicitly deal with outliers using Gilbert algorithm.

4.1 RGD Tree: Explicitly Avoiding the Influence of Outliers in SVM

Roughly speaking, RGD tree is a modified version of Gilbert algorithm. In each step, it randomly samples w points from the $t > 1$ points which have the smallest projection distances, and considers each of the w points as if it is the point with smallest projection distance. This results in a computation tree with a branching factor of w. The value w is chosen to have the property that there is a high probability that the w chosen points contain at least one point that is not an outlier. Together with the fast convergence rate of Gilbert algorithm, we can have a node in the RGD tree whose path to the root contains only points that are not outliers with high probability. Then this path is the desired computation path of Gilbert algorithm in an outlier-free environment.

The following theorem from [7] guarantees the performance of the RGD tree:

▶ **Theorem 14.** *With high probability (larger than $1-\mu$), there exists at least one node in the resulting RGD tree which yields an (ϵ, δ)-approximation, with running time linear in n and d.*

4.2 Extending RGD Tree to Distributed Settings

Given the fact that RGD tree is a variant of Gilbert algorithm, it can be naturally extended to distributed settings. One simple solution is to let the coordinator send the current solution (x_i) to each distributed node for them to compute and return its own t points with the smallest projection distance to $\overline{ox_i}$. However such a naive approach will incur large communication cost, because now each step will need $O(kt)$ communication, instead of $O(k)$ communication as in the no-outlier case. Actually the problem of finding the t-th smallest number in a distributed setting has been extensively studied ([17, 21, 13]). [13] gives a randomized algorithm that has communication complexity of $O(k \log(t))$, and a deterministic algorithm with communication complexity $O(k \log^2(t))$. In this paper we give a deterministic algorithm with communication complexity $O(k \log(t))$ in the coordinator model. Formally, consider the following problem.

▶ **Definition 15.** (Distributed t-selection): Given n distinct numbers distributed among k nodes, find the t-th smallest number.

In this problem, we only consider distinct numbers, since we can use any tie-breaker to distinguish duplicated numbers. Then we have the following result.

▶ **Lemma 16.** *Distributed t-selection can be solved deterministically with $O(k \log(t))$ transfers of numbers in the coordinator model.*

To prove Lemma 16, we give an algorithm (Algorithm 3) with the claimed communication complexity.

Before analyzing the communication complexity of Algorithm 3, we first show its *Correctness.* The algorithm acts like the classical linear-time selection algorithm. The coordinator computes two "weighted" medians of medians ($m_{\hat{l}_h}$ and $m_{\hat{l}_{h+1}}$) for all distributed nodes, and tell them to discard certain portion of its numbers based on the location of the t-th smallest number. In Line **11**, case 1 means that the t-th smallest number is smaller than the first "weighted" median $m_{\hat{l}_h}$, so it is safe to discard all numbers no smaller than $m_{\hat{l}_h}$; case 2 means that the t-th smallest number is between $m_{\hat{l}_h}$ and $m_{\hat{l}_{h+1}}$, so it is safe to discard all numbers no larger than $m_{\hat{l}_h}$ and all numbers no smaller than $m_{\hat{l}_{h+1}}$; Similarly, we can show for Case 3. We also update the value of t if numbers smaller than the t-th smallest number are discarded. The algorithm either finds the t-th smallest number during the loop of Lines **6** to **12**, or finds it by the coordinator in a non-distributed fashion (when there are fewer numbers left). Thus we have the correctness of Algorithm 3.

For communication complexity, we have the following lemma (proof of Lemma 16 is left in the appendix due to space limit).

▶ **Lemma 17.** *Each iteration of Lines **6** to **12** will discard at least a fraction of $\frac{1}{4}$ of the current numbers holden by all nodes.*

Now we are ready to present the distributed version of the RGD tree algorithm (*i.e.,* Algorithm 4), and the analysis of its communication complexity.

The RGD tree has $O(w^h)$ nodes (which is constant w.r.t. n and d). To generate one node, we need $O(k \log(t))$ communication. Notice that each time we draw the sample S_v, there are $O(w)$ extra communication. Thus on average each node in the sample (*i.e.,* its associated point belongs to the sample) is only charged $O(1)$ extra communication, which does not change asymptotically the $O(k \log(t))$ communication incurred by applying Algorithm 3. This leads to the following theorem.

▶ **Theorem 18.** *The communication complexity of Algorithm 4 is $O(w^h k \log(t))$.*

4.3 Extension to Two-Class SVM

We will briefly explain the reasons why an extension to two-class SVM is straight forward. More details are left to the full version of the paper. Finding the polytope distance between two point sets is equivalent to finding the polytope distance between the Minkowski difference of the two point sets and the origin. Taking the Minkowski difference will increase the size of the problem from $O(n)$ to $O(n^2)$, however using the tricks in [7] and [10] we can achieve the same running time and communication performance as the one-class case.

Algorithm 3 Deterministic distributed t-selection algorithm

```
INPUT : $n$ distinct numbers arbitrarily distributed among $k$ nodes, a natural number $t$.
OUTPUT(by the coordinator) : the $t$-th smallest number of the $n$ numbers.
(Pre-process:) For each node, if it holds more than $t$ numbers, do a local sorting and keep the smallest $t$ numbers and discard the rest.
(Pre-process:) The coordinator sends a distinct number $l \in [1, k]$ to each node as their label.
repeat
    For each node, send to the coordinator a message containing a triple $(m_l, n_l, l)$, where $m_l$ is the median of the numbers stored in the node, $n_l$ is the # of numbers in the node, and $l$ is its label.
    Upon receiving messages from all nodes, the coordinator first sorts the messages in ascending order of their $m_l$. Suppose in the new order we have $m_{\hat{l}_1} < m_{\hat{l}_2} < \cdots < m_{\hat{l}_k}$.
    Let $m_{\hat{l}_0} = -\infty$. The coordinator computes a value $h$ such that $\sum_{l:m_l \le m_{\hat{l}_h}} n_l < \frac{1}{2}\sum_l n_l$ and $\sum_{l:m_l \le m_{\hat{l}_{h+1}}} n_l \ge \frac{1}{2}\sum_l n_l$
    The coordinator sends a pair $(m_{\hat{l}_h}, m_{\hat{l}_{h+1}})$ to all nodes.
    Upon receiving $(m_{\hat{l}_h}, m_{\hat{l}_{h+1}})$ from the coordinator, each node sends to the coordinator a triple $(a_l, b_l, l)$, where $a_l$ is the # of its numbers smaller than $m_{\hat{l}_h}$, $b_l$ is the # of its numbers smaller than $m_{\hat{l}_{h+1}}$, $l$ is its label.
    After receiving all $(a, b, l)$ messages from all nodes, the coordinator checks which of the following cases will happen (notice that since $m_{\hat{l}_h} < m_{\hat{l}_{h+1}}$ we always have $\sum a < \sum b$):
      1. $t - 1 < \sum a$;
      2. $\sum a < t - 1 < \sum b$;
      3. $t - 1 > \sum b$;
      4. $t - 1 = \sum a$;
      5. $t - 1 = \sum b$.
    For case 4, output $m_{\hat{l}_h}$ and halt; for case 5, output $m_{\hat{l}_{h+1}}$ and halt; otherwise, send $i$ to all nodes for cases $i$. For case 2, update $t$ to be $t - \sum a$; for case 3, update $t$ to be $t - \sum b$.
    For each node, if it receives a "1" from the coordinator, discard all numbers larger than $m_{\hat{l}_h}$; if it receives a "2", discard all numbers smaller than $m_{\hat{l}_h}$ and numbers larger than $m_{\hat{l}_{h+1}}$; otherwise discard all numbers smaller than $m_{\hat{l}_{h+1}}$
until $\sum n_l = O(k)$
All nodes send their numbers to the coordinator.
Now the numbers are no longer distributed, and the coordinator simply outputs the $t$-th smallest number.
```

Algorithm 4 Distributed RGD tree

```
INPUT : A $d$ dimensional point set $P$ arbitrarily distributed among $k$ nodes, with a fraction $\gamma$ of it being outliers; three parameters $0 < \mu, \delta < 1$, $h = 2(\frac{1}{\epsilon}(\frac{D}{\rho}+1))^2 \ln(\frac{D}{\rho}+1)$.
OUTPUT(by the coordinator) : An RGD tree with each node associated with a candidate for an approximation solution to the One-class SVM with outliers problem.
The coordinator randomly select a point from $P$ as $x$. Initialize the tree at root $x$.
Recursively grow the tree in the following manner:
For a node $v$ associated with point $x_v$, if its height is $h$, it becomes a leaf; Otherwise, do the following:
    1. Let $t = (1+\delta)\gamma|P|$. The coordinator finds the point $p_t$ whose projection distances to $\overline{ox_v}$ are the $t$'th smallest using Algorithm 3;
    2. Take a random sample $S_v$ of size $w = (1+\frac{1}{\delta})\ln\frac{\mu}{h}$ in the following manner: the coordinator randomly take a label of the nodes, and ask the node with this label for a random point of its holdings whose projection distance is smaller than the projection distance of $p_t$. Repeat until the coordinator has a sample $S_v$ of size $w$. For each point $s \in S_v$, create a child of $v$ in the RGD tree and associate it with point $x_v^s$ which is the point on line segment $[sx_v]$ closest to $o$.
```

References

1 Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed clustering on graphs. *CoRR*, 2013.

2 Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. Distributed frank-wolfe algorithm: A unified framework for communication-efficient sparse learning. *CoRR*, 2014.

3 G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. *NIPS*, 2000.

4 E. Y. Chang, K. Zhu, H. Wang, H. Bai, H. Li, Z. Qiu, and H. Cui. Psvm: Parallelizing support vector machines on distributed computers. *21st Neural Info. Processing Systems Conf*, 2007.

5 C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 1995.

6 Hal Daumé, Jeff M. Phillips, Avishek Saha, and Suresh Venkatasubramanian. Efficient protocols for distributed classification and optimization. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory*, 2012.

7 H. Ding and J. Xu. Random gradient descent tree: A combinatorial approach for svm with outliers. *Association for the Advanced of Artificial Intelligence Conf.*, 2015.

8 P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, 2010.

9 G. Fung and O. L. Mangasarian. Incremental support vector machine classification. *Proc. of the SIAM Intl. Conf. on Data Mining*, 2002.

10 G. Gartner and M. Jaggi. Coresets for polytope distance. *SOCG*, 2009.

11 E. G. Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*, 1966.

12 H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. *Advances in Neural Information processing Systems*, 2005.

13 F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. *Symposium on Parallelism in Algorithms and Architectures*, 2007.

14 Y. Lu, V. Roychowdhury, and L. Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *IEEE Tran. on Neural Networks*, 2008.

15 N. M. Luscombe, D. Greenbaum, and M. Gerstein. Review: What is bioinformatics? an introduction and overview. *Yearbook of Medical Informatics*, 2001.

16 A. Navia-Vazquez, D. Gutierrez-Gonzalez, E. Parrado-Hernandez, and J. J. Navarro-Abellan. Distributed support vector machines. *IEEE Tran. on Neural Networks*, 2006.

17 A. Negro, N. Santoro, and J. Urrutia. Efficient distributed selection with bounded messages. *IEEE Trans. Parallel and Distributed Systems*, 1997.

18 D. Pechyony, L. Shen, and R. Jones. Solving large scale linear svm with distributed block minimization. *NIPS 2011 Workshop on Big Learning: Algorithms, Systems, and Tools for Learning at Scale*, 2011.

19 J. Phillips, E. Verbin, and Q. Zhang. Lower bounds for number in hand multiparty communication complexity, made easy. *SODA*, 2012.

20 Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*. NIPS, 2011.

21 N. Santoro, J. B. Sidney, and S. J. Sidney. A distributed selection algorithm and its expected communication complexity. *Theoretical Computer Science*, 1992.

22 J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 2008.

Single Machine Scheduling with Job-Dependent Machine Deterioration*

Wenchang Luo[1], Yao Xu[2], Weitian Tong[3], and Guohui Lin[4]

1 **Faculty of Science, Ningbo University. Ningbo, Zhejiang 315211, China; and Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada**
wenchang@ualberta.ca

2 **Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada**
xu2,guohui@ualberta.ca

3 **Department of Computer Sciences, Georgia Southern University. Statesboro, Georgia 30460, USA**
wtong@georgiasouthern.edu

4 **Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada**
guohui@ualberta.ca

Abstract

We consider the single machine scheduling problem with job-dependent machine deterioration. In the problem, we are given a single machine with an initial non-negative maintenance level, and a set of jobs each with a non-preemptive processing time and a machine deterioration. Such a machine deterioration quantifies the decrement in the machine maintenance level after processing the job. To avoid machine breakdown, one should guarantee a non-negative maintenance level at any time point; and whenever necessary, a maintenance activity must be allocated for restoring the machine maintenance level. The goal of the problem is to schedule the jobs and the maintenance activities such that the total completion time of jobs is minimized. There are two variants of maintenance activities: in the partial maintenance case each activity can be allocated to increase the machine maintenance level to any level not exceeding the maximum; in the full maintenance case every activity must be allocated to increase the machine maintenance level to the maximum. In a recent work, the problem in the full maintenance case has been proven NP-hard; several special cases of the problem in the partial maintenance case were shown solvable in polynomial time, but the complexity of the general problem is left open. In this paper we first prove that the problem in the partial maintenance case is NP-hard, thus settling the open problem; we then design a 2-approximation algorithm.

1998 ACM Subject Classification F.2.2 Sequencing and scheduling, F.1.3 Reducibility and completeness, I.1.2 Analysis of algorithms

Keywords and phrases Scheduling, machine deterioration, maintenance, NP-hard, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.55

[*] This work was partially supported by the K. C. Wong Magna Foundation of Ningbo University, the China Scholarship Council (Grant No. 201408330402), the Ningbo Natural Science Foundation (2016A610078) to W. L., the NSERC Canada to G. L., and the Office of the Vice President for Research & Economic Development and Georgia Southern University to W. T.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 55; pp. 55:1–55:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In many scheduling problems, processing a job on a machine causes the machine to deteriorate to some extent, and consequently maintenance activities need to be executed in order to restore the machine capacity. Scheduling problems with maintenance activities have been extensively investigated since the work of Lee and Liman [7].

A maintenance activity is normally described by two parameters, the starting time and the duration. If these two parameters are given beforehand, a maintenance activity is referred to as *fixed*; otherwise it is called *flexible*. Various scheduling models with fixed maintenance activities, on different machine environments and job characteristics, have been comprehensively surveyed by Schmidt [14], Lee [5], and Ma *et al.* [10].

A number of researchers initiated the work with flexible maintenance activities. Qi *et al.* [13] considered a single machine scheduling problem to simultaneously schedule jobs and maintenance activities, with the objective to minimize the total completion time of jobs. They showed that the problem is NP-hard in the strong sense and proposed heuristics and a branch-and-bound exact algorithm. (Qi [12] later analyzed the worst-case performance ratio for one of the heuristics, *the shortest processing time first* or SPT.) Lee and Chen [6] studied the multiple parallel machines scheduling problem where each machine must be maintained exactly once, with the objective to minimize the total weighted completion time of jobs. They proved the NP-hardness for some special cases and proposed a branch-and-bound exact algorithm based on column generation; the NP-hardness for the general problem is implied. Kubzin and Strusevich [4] considered a two-machine open shop and a two-machine flow shop scheduling problems in which each machine has to be maintained exactly once and the duration of each maintenance depends on its starting time. The objective is to minimize the maximum completion time of all jobs and all maintenance activities. Among others, the authors showed that the open shop problem is polynomial time solvable for quite general functions defining the duration of maintenance in its starting time; they also proved that the flow shop problem is binary NP-hard and presented a fully polynomial time approximation scheme (FPTAS) [4].

Returning to a single machine scheduling problem, Chen [2] studied the periodic maintenance activities of a constant duration not exceeding the available period, with the objective to minimize the maximum completion time of jobs (that is, the *makespan*). The author presented two mixed integer programs and heuristics and conducted computational experiments to examine their performance. Mosheiov and Sarig [11] considered the problem where the machine needs to be maintained prior to a given deadline, with the objective to minimize the total weighted completion time of jobs. They showed the binary NP-hardness and presented a pseudo-polynomial time dynamic programming algorithm and an efficient heuristic. Luo *et al.* [8] investigated a similar variant (to [11]) in which the jobs are weighted and the duration of the maintenance is a nondecreasing function of the starting time (which must be prior to a given deadline). Their objective is to minimize the total weighted completion time of jobs; the authors showed the weak NP-hardness, and for the special case of concave duration function they proposed a $(1 + \sqrt{2}/2 + \epsilon)$-approximation algorithm. Yang and Yang [17] considered a position-dependent aging effect described by a power function under maintenance activities and variable maintenance duration considerations simultaneously; they examined two models with the objective to minimize the makespan, and for each of them they presented a polynomial time algorithm.

Scheduling on two identical parallel machines with periodic maintenance activities was examined by Sun and Li [15], where the authors presented approximation algorithms with constant performance ratios for minimizing the makespan or minimizing the total completion time of jobs. Xu *et al.* [16] considered the case where the length of time between two consecutive maintenances is bounded; they presented an approximation algorithm for the multiple parallel machines scheduling problem to minimize the completion time of the last maintenance, and for the single machine scheduling problem to minimize the makespan, respectively.

1.1 Problem definition

Considering the machine deterioration in the real world, in a recent work by Bock *et al.* [1], a new scheduling model subject to *job-dependent machine deterioration* is introduced. In this model, the single machine must have a non-negative *maintenance level* (ML) at any time point, specifying its current maintenance state. (A negative maintenance level indicates the machine breakdown, which is prohibited.) We are given a set of jobs $\mathcal{J} = \{J_i, i = 1, 2, \ldots, n\}$, where each job $J_i = (p_i, \delta_i)$ is specified by its non-preemptive *processing time* p_i and *machine deterioration* δ_i. The machine deterioration δ_i quantifies the decrement in the machine maintenance level after processing the job J_i. (That is, if before processing the job J_i the maintenance level is ML, then afterwards the maintenance level reduces to $\mathrm{ML} - \delta_i$ — suggesting that ML has to be at least δ_i in order for the machine to process the job J_i.)

Clearly, to process all the jobs, *maintenance activities* (MAs) need to be allocated inside a schedule to restore the maintenance level, preventing machine breakdown. Given that the machine can have a maximum maintenance level of $\mathrm{ML}^{\max}$, and assuming a unit maintenance speed, an MA of a duration D would increase the maintenance level by $\min\{D, \mathrm{ML}^{\max} - \mathrm{ML}\}$, where ML is the maintenance level before the MA.

With an initial machine maintenance level ML_0, $0 \le \mathrm{ML}_0 \le \mathrm{ML}^{\max}$, the goal of the problem is to schedule the jobs and necessary MAs such that all jobs can be processed without machine breakdown, and that the total completion time of jobs is minimized.

There are two variants of the problem depending on whether or not one has the freedom to choose the duration of an MA: in the *partial maintenance* case, the duration of each MA can be anywhere in between 0 and $(\mathrm{ML}^{\max} - \mathrm{ML})$, where ML is the maintenance level before the MA; in the *full maintenance* case, however, the duration of every MA must be exactly $(\mathrm{ML}^{\max} - \mathrm{ML})$, consequently increasing the maintenance level to the maximum value $\mathrm{ML}^{\max}$. Let C_i denote the completion time of the job J_i, for $i = 1, 2, \ldots, n$. In the three field notation, the two problems discussed in this paper are denoted as $(1|p\,\mathrm{MA}\,|\sum_i C_i)$ and $(1|f\,\mathrm{MA}\,|\sum_i C_i)$, respectively, where $p\,\mathrm{MA}$ and $f\,\mathrm{MA}$ refer to the partial and the full maintenance, respectively.

1.2 Prior work and our contribution

Bock *et al.* [1] proved that $(1|f\,\mathrm{MA}\,|\sum_i C_i)$ is NP-hard, even when $p_i = p$ for all i or when $p_i = \delta_i$ for all i, both by a reduction from the Partition problem [3]; while all the jobs have the same deterioration, i.e. $\delta_i = \delta$ for all i, the problem can be solved in $O(n \log n)$ time. For the partial maintenance case, Bock *et al.* [1] showed that the SPT rule gives an optimal schedule for $(1|p\,\mathrm{MA}\,|\sum_i C_i)$ when $p_i < p_j$ implies $p_i + \delta_i \le p_j + \delta_j$ for each pair of i and j (which includes the special cases where $p_i = p$ for all i, or $\delta_i = \delta$ for all i, or $p_i = \delta_i$ for all i). The complexity of the general problem $(1|p\,\mathrm{MA}\,|\sum_i C_i)$ was left as an open problem. Also, to the best of our knowledge, no approximation algorithms have been designed for either problem.

Our main contribution in this paper is to settle the NP-hardness of the general problem $(1|p\,\mathrm{MA}\,|\sum_i C_i)$. Such an NP-hardness might appear a bit surprising at the first glance since one has so much freedom in choosing the starting time and the duration of each MA. Our reduction is from the Partition problem too, using a kind of job swapping argument. This reduction is presented in Section 3, following some preliminary properties we observe for the problem in Section 2. In Section 4, we propose a 2-approximation algorithm for $(1|p\,\mathrm{MA}\,|\sum_i C_i)$. We conclude the paper in Section 5 with some discussion on the (in-)approximability.

Lastly, we would like to point out that when the objective is to minimize the makespan $C_{\max}$, i.e. the maximum completion time of jobs, $(1|p\,\mathrm{MA}\,|C_{\max})$ can be trivially solved in $O(n)$ time and $(1|f\,\mathrm{MA}\,|C_{\max})$ is NP-hard but admits an $O\left(n^2(\mathrm{ML}^{\max})^2\log\left(\sum_{i=1}^n(p_i+\delta_i)\right)\right)$ time algorithm based on dynamic programming (and thus admits an FPTAS) [1].

2 Preliminaries

Given a feasible schedule π to the problem $(1|p\,\mathrm{MA}\,|\sum_i C_i)$, which specifies the start processing time for each job and the starting time and the duration of each MA, we abuse slightly π to also denote the permutation of the job indices $(1,2,\ldots,n)$ in which the jobs are processed in order: $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$. The following lemma is proved in [1].

▶ **Lemma 1** ([1]). *There is an optimal schedule π to $(1|p\,\mathrm{MA}\,|\sum_i C_i)$ such that the total maintenance duration before processing the job J_{π_i} equals $\max\left\{0,\sum_{j=1}^i\delta_{\pi_j}-\mathrm{ML}_0\right\}$, for each $i=1,2,\ldots,n$.*

Lemma 1 essentially states that each MA should be pushed later in the schedule as much as possible until absolutely necessary, and its duration should be minimized just for processing the succeeding job. In the sequel, we limit our discussion on the feasible schedules satisfying these two properties. We define the *separation job* in such a schedule π as the first job that requires an MA (of a positive duration).

▶ **Lemma 2.** *Suppose J_{π_k} is the separation job in an optimal schedule π to $(1|p\,\mathrm{MA}\,|\sum_i C_i)$. Then,*

- *the jobs before the separation job J_{π_k} are scheduled in the SPT order;*
- *the jobs after the separation job J_{π_k} are scheduled in the* shortest sum-of-processing-time-and-deterioration first *(SSF) order;*
- *the jobs adjacent to the separation job J_{π_k} satisfy*

$$p_{\pi_{k-1}}+\min\{\delta_{\pi_{k-1}},\delta_{\pi_k}-\delta\}\le p_{\pi_k}+(\delta_{\pi_k}-\delta)\le p_{\pi_{k+1}}+\max\{0,\delta_{\pi_{k+1}}-\delta\},$$

where $\delta=\mathrm{ML}_0-\sum_{i=1}^{k-1}\delta_{\pi_i}$ is the remaining maintenance level before the first MA.

Proof. Starting with an optimal schedule satisfying the properties stated in Lemma 1, one may apply a simple job swapping procedure if the job order is violated either in the prefix or in the suffix of job order separated by the separation job J_{π_k}. This procedure would decrease the value of the objective, contradicting to the optimality. That is, we have (see Figure 1 for an illustration)

$$p_{\pi_1}\le p_{\pi_2}\le\ldots\le p_{\pi_{k-1}},\quad\text{and}\tag{1}$$

$$p_{\pi_{k+1}}+\delta_{\pi_{k+1}}\le p_{\pi_{k+2}}+\delta_{\pi_{k+2}}\le\ldots\le p_{\pi_n}+\delta_{\pi_n}.\tag{2}$$

Let $\delta=\mathrm{ML}_0-\sum_{i=1}^{k-1}\delta_{\pi_i}$ denote the remaining maintenance level before the first MA. Because $\delta<\delta_{\pi_k}$, an (the first) MA of duration $\delta_{\pi_k}-\delta$ needs to be performed for processing

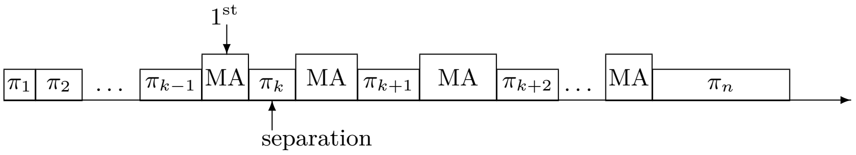


Figure 1 An illustration of the optimal schedule π stated in Lemma 2, where the separation job is J_{π_k}; the width of a framebox does not necessarily equal the processing time of a job or the duration of an MA.

the separation job J_{π_k}. From the optimality of π, swapping the two jobs J_{π_k} and $J_{\pi_{k+1}}$ should not decrease the objective, that is,

$$\begin{cases} p_{\pi_k} + (\delta_{\pi_k} - \delta) \le p_{\pi_{k+1}} + (\delta_{\pi_{k+1}} - \delta), & \text{if } \delta_{\pi_{k+1}} > \delta; \\ p_{\pi_k} + (\delta_{\pi_k} - \delta) \le p_{\pi_{k+1}}, & \text{otherwise.} \end{cases}$$

Similarly, swapping the two jobs $J_{\pi_{k-1}}$ and J_{π_k} should not decrease the objective, that is,

$$\begin{cases} p_{\pi_{k-1}} \le p_{\pi_k}, & \text{if } \delta_{\pi_{k-1}} \ge \delta_{\pi_k} - \delta; \\ p_{\pi_{k-1}} + \delta_{\pi_{k-1}} \le p_{\pi_k} + (\delta_{\pi_k} - \delta), & \text{otherwise.} \end{cases}$$

These together give

$$p_{\pi_{k-1}} + \min\{\delta_{\pi_{k-1}}, \delta_{\pi_k} - \delta\} \le p_{\pi_k} + (\delta_{\pi_k} - \delta) \le p_{\pi_{k+1}} + \max\{0, \delta_{\pi_{k+1}} - \delta\}. \tag{3}$$

This proves the lemma. ◀

From Lemma 2, one sees that the separation job in an optimal schedule is unique, in the sense that it cannot always be "appended" to either the prefix SPT order or the suffix SSF order. This is reflected in our NP-completeness reduction in Section 3, where we force a certain scenario to happen.

3 NP-hardness of the problem $(1|p\,\text{MA}\,|\sum_i C_i)$

Our reduction is from the classic NP-complete problem Partition [3], formally defined as follows:

Partition
Instance: A set X of n positive integers $X = \{x_1, x_2, \ldots, x_n\}$, with $\sum_{i=1}^n x_i = 2B$.
Query: Is there a subset $X_1 \subset X$ such that $\sum_{x \in X_1} x = \sum_{x \in X - X_1} x = B$?

We abuse X to denote the instance of Partition with the set $X = \{x_1, x_2, \ldots, x_n\}$ and $\sum_{i=1}^n x_i = 2B$. The corresponding instance I of the problem $(1|p\,\text{MA}\,|\sum_i C_i)$ is constructed in polynomial time, as follows:

$$\begin{array}{rl} \text{Number of jobs:} & 2n+3; \\ \text{Job processing time:} & p_{n+1+i} = p_i = \sum_{j=1}^i x_j, \text{ for } i = 0, 1, 2, \ldots, n, \\ & p_{2n+2} = M - 2B; \\ \text{Machine deterioration:} & \delta_{n+1+i} = \delta_i = M - 2p_i, \text{ for } i = 0, 1, 2, \ldots, n, \\ & \delta_{2n+2} = 0; \\ \text{Initial maintenance level:} & \text{ML}_0 = \sum_{i=0}^n \delta_i - 2B; \\ \text{Maximum maintenance level:} & \text{ML}^{\max} = \sum_{i=0}^n \delta_i; \\ \text{Objective threshold:} & Q = Q_0 + B, \end{array}$$

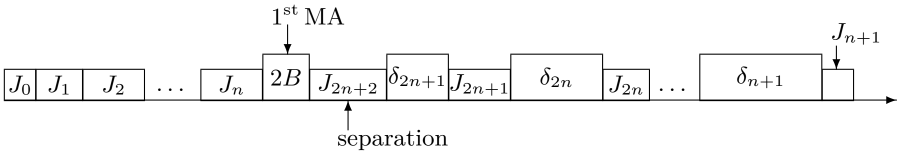


Figure 2 The initial infeasible schedule π^0 for the instance I with the separation job J_{2n+2}; π^0 satisfies all properties stated in Lemma 2. All MAs are indicated by their respective durations (for the first MA, its duration is $\delta_{2n+2} - \delta = 2B$).

(note that $p_{n+1} = p_0 = \sum_{j=1}^{0} x_j = 0$ due to the empty range for j) where M is a big integer:

$$M > (4n + 8)B, \tag{4}$$

and Q_0 is the total completion time of jobs for an *initial infeasible schedule* π^0 (see Figure 2):

$$Q_0 = \sum_{j=0}^{n}(n-j+1)p_j + (n+2)\left(\sum_{j=0}^{n} p_j + 2B + p_{2n+2}\right) + \sum_{j=0}^{n}(j+1)(p_{n+1+j} + \delta_{n+1+j}). \tag{5}$$

The job order in this initial schedule π^0 is $(J_0, J_1, \dots, J_n, J_{2n+2}, J_{2n+1}, J_{2n}, \dots, J_{n+1})$, and the first MA precedes the job J_{2n+2}, which is regarded as the separation job (see Figure 2). Before the separation job J_{2n+2}, the machine maintenance level is allowed to go into negative, but has to be restored to zero just for processing J_{2n+2}; afterwards, machine breakdown is no longer tolerated. From $\mathrm{ML}_0 = \sum_{i=0}^{n} \delta_i - 2B$, we know that π^0 is *infeasible* due to machine breakdown before the first MA; we will convert it to a feasible schedule later. The QUERY of the decision version of the problem $(1|p\,\mathrm{MA}\,|\sum_i C_i)$ is whether or not there exists a feasible schedule π such that the total completion time of jobs is no more than $Q = Q_0 + B$.

Despite the infeasibility, the initial schedule π^0 has all the properties stated in Lemma 2, with the separation job J_{2n+2} at the *center position*. The first $(n + 1)$ jobs are in the SPT order and the last $(n + 1)$ jobs are in the SSF order; since $\delta = -2B$, $p_n = p_{2n+1} = 2B$, $\delta_n = \delta_{2n+1} = M - 4B$, $p_{2n+2} = M - 2B$, $\delta_{2n+2} = 0$, Eq. (3) is also satisfied due to the big M in Eq. (4):

$$p_n + \min\{\delta_n, \delta_{2n+2} - \delta\} < p_{2n+2} + (\delta_{2n+2} - \delta) = p_{2n+1} + \max\{0, \delta_{2n+1} - \delta\}.$$

In the rest of the section, we will show that there is a subset $X_1 \subset X$ of sum exactly B if and only if the initial schedule π^0 can be converted into a feasible schedule π with the total completion time of jobs no more than $Q = Q_0 + B$, through a *repeated job swapping* procedure.

Notice that the two jobs J_i and J_{n+1+i} are identical, for $i = 0, 1, \dots, n$. In any schedule with the job J_{2n+2} at the center position, if exactly one of J_i and J_{n+1+i} is scheduled before J_{2n+2}, then we always say J_i is scheduled before J_{2n+2} while J_{n+1+i} is scheduled after J_{2n+2}. Also, when the two jobs J_i and J_{n+1+i} are both scheduled before J_{2n+2}, then J_{n+1+i} precedes J_i; when the two jobs J_i and J_{n+1+i} are both scheduled after J_{2n+2}, then J_i precedes J_{n+1+i}.

3.1 Proof of "only if"

In this subsection, we show that if there is a subset $X_1 \subset X$ of sum exactly B, then the initial infeasible schedule π^0 can be converted into a feasible schedule π with the total completion time no more than $Q = Q_0 + B$. We also demonstrate the repeated job swapping procedure leading to this successful schedule π.

Suppose the indices of the elements in the subset X_1 are $\{i_1, i_2, \ldots, i_m\}$, satisfying $1 \le i_1 < i_2 < \ldots < i_m \le n$. Starting with the initial schedule π^0, we sequentially swap the job $J_{i_\ell - 1}$ with the job J_{n+1+i_ℓ}, for $\ell = 1, 2, \ldots, m$. Let π^ℓ denote the schedule after the ℓ-th job swapping.

▶ **Lemma 3.** *For each $1 \le \ell \le m$,*

- *the schedule π^ℓ with the separation job J_{2n+2} satisfies the properties in Lemma 2;*
- *the ℓ-th job swapping decreases the total machine deterioration before the separation job J_{2n+2} by $2x_{i_\ell}$;*
- *the ℓ-th job swapping increases the total completion time by x_{i_ℓ}.*

Proof. Recall that the two jobs J_{i_ℓ} and J_{n+1+i_ℓ} are identical. Before the ℓ-th job swapping between $J_{i_\ell - 1}$ and J_{n+1+i_ℓ} (in the schedule $\pi^{\ell-1}$), the jobs in between $J_{i_\ell - 1}$ and J_{n+1+i_ℓ} are

$$(J_{i_\ell - 1}, J_{i_\ell}, J_{i_\ell + 1}, \ldots, J_n, J_{2n+2}, J_{2n+1}, J_{2n}, \ldots, J_{n+1+i_\ell+1}, J_{n+1+i_\ell}).$$

After the swapping (in the schedule π^ℓ) this sub-schedule becomes

$$(J_{n+1+i_\ell}, J_{i_\ell}, J_{i_\ell + 1}, \ldots, J_n, J_{2n+2}, J_{2n+1}, J_{2n}, \ldots, J_{n+1+i_\ell+1}, J_{i_\ell - 1}).$$

By a simple induction, all jobs before J_{n+1+i_ℓ} have their processing times less than p_{i_ℓ}, and thus the jobs before the separation job J_{2n+2} are in the SPT order; for a similar reason, the jobs after the separation job J_{2n+2} are in the SSF order.

By the ℓ-th job swapping, the change in the total machine deterioration before the separation job J_{2n+2} is $\delta_{i_\ell} - \delta_{i_\ell - 1} = -2(p_{i_\ell} - p_{i_\ell - 1}) = -2x_{i_\ell}$, that is, decreases by $2x_{i_\ell}$. Therefore the duration of the first MA also decreases by $2x_{i_\ell}$. Since J_n always directly precedes J_{2n+2} and $p_n < p_{2n+2}$, the first half of Eq. (3) holds; since $p_{2n+2} + \delta_{2n+2}$ is the smallest among all jobs, the second half of Eq. (3) holds. That is, the schedule π^ℓ satisfies all properties in Lemma 2.

For ease of presentation, let C_i denote the completion time of the job J_i in the schedule π^ℓ, and let C'_i denote the completion time of the job J_i in the schedule $\pi^{\ell-1}$. Comparing to the schedule $\pi^{\ell-1}$ ($\ell \ge 1$), after the ℓ-th job swapping between $J_{i_\ell - 1}$ and J_{n+1+i_ℓ},

- the completion time of jobs preceding J_{n+1+i_ℓ} is unchanged;
- $C_{n+1+i_\ell} - C'_{i_\ell - 1} = p_{i_\ell} - p_{i_\ell - 1} = x_{i_\ell}$;
- the completion time of each job in between J_{i_ℓ} and J_n (inclusive, $n - i_\ell + 1$ of them) increases by x_{i_ℓ};
- the duration of the first MA decreases by $2x_{i_\ell}$;
- the completion time of each job in between J_{2n+2} and $J_{n+1+i_\ell+1}$ (inclusive, $n - i_\ell + 1$ of them) decreases by x_{i_ℓ};
- $C_{i_\ell - 1} - C'_{n+1+i_\ell} = -x_{i_\ell} + (\delta_{i_\ell - 1} + p_{i_\ell - 1}) - (\delta_{i_\ell} + p_{i_\ell}) = 0$;
- from the last item, the completion time of jobs succeeding $J_{i_\ell - 1}$ is unchanged.

In summary, there are $(n - i_\ell + 2)$ jobs of which the completion time increases by x_{i_ℓ} and $(n - i_\ell + 1)$ jobs of which the completion time decreases by x_{i_ℓ}. Therefore, the ℓ-th job swapping between $J_{i_\ell - 1}$ and J_{n+1+i_ℓ} increases the total completion time by x_{i_ℓ}. This finishes the proof. ◀

▶ **Theorem 4.** *If there is a subset $X_1 \subset X$ of sum exactly B, then there is a feasible schedule π to the instance I with the total completion time no more than $Q = Q_0 + B$.*

Proof. Let the indices of the elements in the subset X_1 be $\{i_1, i_2, \ldots, i_m\}$, such that $1 \le i_1 < i_2 < \ldots < i_m \le n$. Starting with the initial schedule π^0, we sequentially swap the job $J_{i_\ell - 1}$ with the job J_{n+1+i_ℓ}, for $\ell = 1, 2, \ldots, m$. Let π^ℓ denote the schedule after the ℓ-th job swapping, and let Q_ℓ denote the total completion time of jobs in π^ℓ.

From Lemma 3 we know that the ending schedule π^m satisfies all the properties in Lemma 2. Also, the total machine deterioration before the separation job J_{2n+2} in π^m is

$$\sum_{i=0}^{n} \delta_i - 2\sum_{\ell=1}^{m} x_{i_\ell} = \sum_{i=0}^{n} \delta_i - 2B = \mathrm{ML}_0,$$

suggesting that π^m is a feasible schedule. (The first MA has zero duration and thus becomes unnecessary.)

Moreover, the total completion time of jobs in π^m is $Q_m = Q_0 + \sum_{\ell=1}^{m} x_{i_\ell} = Q_0 + B$. Therefore, the schedule π^m obtained from the initial schedule π^0 through the repeated job swapping procedure is a desired one. ◀

3.2 Proof of "if"

In this subsection, we show that if there is a feasible schedule π to the constructed instance I with the total completion time no more than $Q = Q_0 + B$, then there is a subset $X_1 \subset X$ of sum exactly B. Assume without loss of generality that the schedule π satisfies the properties in Lemma 2. We start with some structure properties which the schedule π must have; the interested readers may refer to [9] for detailed proofs.

▶ **Lemma 5** ([9]). *Excluding the job J_{2n+2}, there are at least n and at most $(n+1)$ jobs scheduled before the first* MA *in the schedule π.*

▶ **Lemma 6** ([9]). *There are at most $(n+1)$ jobs scheduled after J_{2n+2} in the schedule π.*

Combining Lemmas 5 and 6, we have the following lemma regarding the position of J_{2n+2} in the schedule π.

▶ **Lemma 7** ([9]). *In the schedule π, the position of the job J_{2n+2} has three possibilities:*

Case 1: *There are $(n+2)$ jobs before the first* MA *with $\pi_{n+2} = 2n+2$, and $J_{\pi_{n+3}}$ is the separation job.*

Case 2: *There are $(n+1)$ jobs before the first* MA, *$J_{\pi_{n+2}}$ is the separation job, and $\pi_{n+3} = 2n+2$.*

Case 3: *There are n jobs before the first* MA, *$J_{\pi_{n+1}}$ is the separation job, and $\pi_{n+2} = 2n+2$.*

Recall that the job order in the initial infeasible schedule π^0 is $(J_0, J_1, \ldots, J_n, J_{2n+2}, J_{2n+1}, J_{2n}, \ldots, J_{n+2}, J_{n+1})$, and the first MA is executed before processing the job J_{2n+2}, which is regarded as the separation job (see Figure 2). In the sequel, we will again convert π^0 into our target schedule π through a repeated job swapping procedure. During such a procedure, the job J_{2n+2} is kept at the center position, and a job swapping always involves a job before J_{2n+2} and a job after J_{2n+2}.

In Cases 1 and 3 of the schedule π, the job J_{2n+2} is at the center position (recall that there are in total $2n+3$ jobs), and therefore the target schedule is well set. In Case 2, J_{2n+2} is at position $n+3$, not the center position; we first exchange J_{2n+2} and $J_{\pi_{n+2}}$ to obtain a schedule π', which becomes our target schedule. That is, we will first convert π^0 into π' through a repeated job swapping procedure, and at the end exchange J_{2n+2} back to the position $n+3$ to obtain the final schedule π. In summary, our primary goal is to convert the schedule π^0 through a repeated job swapping procedure, keeping the job J_{2n+2} at the center position and keeping the first MA right before the job J_{2n+2} (to be detailed next). At the end, to obtain the target schedule π, in Case 1, we swap the job J_{2n+2} and the first MA (*i.e.*, moving the first MA one position backward); in Case 2, we swap J_{2n+2} and the immediate succeeding MA and the following job (with the MA merged with the first MA); in Case 3, we swap the first MA and its immediate preceding job (*i.e.*, moving the first MA one position forward).

In the target schedule (π in Cases 1 and 3, or π' in Case 2), let $R = \{r_1, r_2, \ldots, r_m\}$ denote the subset of indices such that both J_{r_j} and J_{n+1+r_j} are among the first $(n+1)$ jobs, where $0 \le r_1 < r_2 < \ldots < r_m \le n$, and $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$ denote the subset of indices such that both J_{ℓ_j} and $J_{n+1+\ell_j}$ are among the last $(n+1)$ jobs, where $0 \le \ell_1 < \ell_2 < \ldots < \ell_m \le n$. Note that J_{2n+2} is at the center position in the target schedule, and thus it has to be $|R| = |L|$ and we let $m = |R|$. Clearly, all these ℓ_j's and r_j's are distinct from each other.

In the repeated job swapping procedure leading the initial infeasible schedule π^0 to the target feasible schedule, the j-th job swapping is to swap the two jobs J_{ℓ_j} and J_{n+1+r_j}. The resultant schedule after the j-th job swapping is denoted as π^j, for $j = 1, 2, \ldots, m$. In Section 3.1, the job swapping is "*regular*" in the sense that $\ell_j = r_j - 1$ for all j, but now ℓ_j and r_j do not necessarily relate to each other. We remark that immediately after the swapping, a job sorting is needed to restore the SPT order for the prefix and the SSF order for the suffix (see the last paragraph before Section 3.1 for possible re-indexing the jobs).

The following Lemma 8 on the j-th job swapping, when $\ell_j < r_j$, is an extension of Lemma 3.

▶ **Lemma 8** ([9]). *For each $1 \le j \le m$, if the schedule π^{j-1} satisfies the first two properties in Lemma 2 and $\ell_j < r_j$, then*

- *the schedule π^j satisfies the first two properties in Lemma 2;*
- *the j-th job swapping decreases the total machine deterioration before the center job J_{2n+2} by $\delta_{\ell_j} - \delta_{r_j} = 2\sum_{k=\ell_j+1}^{r_j} x_k$;*
- *the j-th job swapping increases the total completion time by at least $\sum_{k=\ell_j+1}^{r_j} x_k$; and the increment equals $\sum_{k=\ell_j+1}^{r_j} x_k$ if and only if $\ell_j > r_{j-1}$.*

▶ **Lemma 9** ([9]). *For each $1 \le j \le m$, if the schedule π^{j-1} satisfies the first two properties in Lemma 2 and $\ell_j > r_j$, then*

- *the schedule π^j satisfies the first two properties in Lemma 2;*
- *the j-th job swapping increases the total machine deterioration before the center job J_{2n+2} by $\delta_{r_j} - \delta_{\ell_j} = 2\sum_{k=r_j+1}^{\ell_j} x_k$;*
- *the j-th job swapping increases the total completion time by at least $\sum_{k=r_j+1}^{\ell_j} x_k$.*

▶ **Theorem 10.** *If there is a feasible schedule π to the instance I with the total completion time no more than $Q = Q_0 + B$, then there is a subset $X_1 \subset X$ of sum exactly B.*

Proof. We start with a feasible schedule π, which has the first two properties stated in Lemma 2 and for which the total completion time is no more than $Q = Q_0 + B$. Excluding the job J_{2n+2}, using the first $n+1$ jobs and the last $n+1$ job in π, we determine the two subsets of indices $R = \{r_1, r_2, \ldots, r_m\}$ and $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$, and define the corresponding m job swappings. We then repeatedly apply the job swapping to convert the initial infeasible schedule π^0 into π.

In Case 1, the total machine deterioration of the first $(n+1)$ jobs in π is

$$\sum_{i=0}^{n} \delta_i - 2\sum_{\ell_j < r_j} \sum_{k=\ell_j+1}^{r_j} x_k + 2\sum_{\ell_j > r_j} \sum_{k=r_j+1}^{\ell_j} x_k = \mathrm{ML}_0 - \delta,$$

implying that

$$\sum_{\ell_j<r_j}\sum_{k=\ell_j+1}^{r_j} x_k - \sum_{\ell_j>r_j}\sum_{k=r_j+1}^{\ell_j} x_k = B + \frac{1}{2}\delta, \tag{6}$$

where $\delta \geq 0$ is the remaining machine maintenance level before the first MA.

On the other hand, the total completion time of jobs in the schedule π is at least

$$Q_0 + \sum_{\ell_j<r_j}\sum_{k=\ell_j+1}^{r_j} x_k + \sum_{\ell_j>r_j}\sum_{k=r_j+1}^{\ell_j} x_k = Q_0 + B + \frac{1}{2}\delta + 2\sum_{\ell_j>r_j}\sum_{k=r_j+1}^{\ell_j} x_k.$$

It follows that 1) $\delta = 0$; 2) there is no pair of swapping jobs J_{ℓ_j} and J_{n+1+r_j} such that $\ell_j > r_j$; and 3) $\ell_1 < r_1 < \ell_2 < r_2 < \ldots < \ell_m < r_m$ (from the third item of Lemma 8). Therefore, from Eq. (6), for the subset $X_1 = \cup_{j=1}^{m}\{x_{\ell_j+1}, x_{\ell_j+2}, \ldots, x_{r_j}\}$, $\sum_{x\in X_1} x = B$. That is, the instance X of the Partition problem is a yes-instance.

In the other two cases, one can similarly, though a bit more complex, show that the instance X is a yes-instance. The detailed proofs are in [9]. ◀

The following theorem follows immediately from Theorems 4 and 10.

▶ **Theorem 11.** *The general problem* $(1|p\,\mathrm{MA}\,|\sum_j C_j)$ *is NP-hard.*

4 A 2-approximation algorithm for $(1|p\,\mathrm{MA}\,|\sum_j C_j)$

Recall that in the problem $(1|p\,\mathrm{MA}\,|\sum_j C_j)$, we are given a set of jobs $\mathcal{J} = \{J_i, i = 1, 2, \ldots, n\}$, where each job $J_i = (p_i, \delta_i)$ is specified by its non-preemptive *processing time* p_i and *machine deterioration* δ_i. The machine deterioration δ_i quantifies the decrement in the machine maintenance level after processing the job J_i. The machine has an initial machine maintenance level ML_0, $0 \leq \mathrm{ML}_0 \leq \mathrm{ML}^{\max}$, where $\mathrm{ML}^{\max}$ is the maximum maintenance level. The goal is to schedule the jobs and necessary MAs of any duration such that all jobs can be processed without machine breakdown, and that the total completion time of jobs is minimized.

In this section, we present a 2-approximation algorithm, denoted as $\mathcal{A}_1$, for the problem. Furthermore, the algorithm $\mathcal{A}_1$ produces a feasible schedule π satisfying the first two properties stated in Lemma 2, suggesting that if the third property is violated then a local job swapping can further decrease the total completion time.

In the algorithm $\mathcal{A}_1$, the first step is to sort the jobs in SSF order (and thus we assume without loss of generality that) $p_1 + \delta_1 \leq p_2 + \delta_2 \leq \ldots \leq p_n + \delta_n$. In the second step, the separation job is determined to be J_k, where k is the maximum index such that $\sum_{i=1}^{k-1} \delta_i \leq \mathrm{ML}_0$. In the last step, the jobs preceding the separation job J_k are re-sorted in the SPT order, denoted by $(J_{i_1}, J_{i_2}, \ldots, J_{i_{k-1}})$, and the jobs succeeding the separation job are $(J_{k+1}, J_{k+2}, \ldots, J_n)$. That is, the solution schedule is

$$\pi = (J_{i_1}, J_{i_2}, \ldots, J_{i_{k-1}}; \mathrm{MA}_1, J_k; \mathrm{MA}_2, J_{k+1}, \mathrm{MA}_3, J_{k+2}, \ldots, \mathrm{MA}_{n-k+1}, J_n),$$

where $\mathrm{MA}_1 = \sum_{j=1}^{k} \delta_j - \mathrm{ML}_0$ and $\mathrm{MA}_i = \delta_{k-1+i}$ for $i = 2, 3, \ldots, n-k+1$.

Let π^* denote an optimal schedule satisfying all properties stated in Lemma 2, and its separation job is $J_{\pi^*_{k^*}}$:

$$\pi^* = (J_{\pi^*_1}, J_{\pi^*_2}, \ldots, J_{\pi^*_{k^*-1}}; \mathrm{MA}^*_1, J_{\pi^*_{k^*}}; \mathrm{MA}^*_2, J_{\pi^*_{k^*+1}}, \mathrm{MA}^*_3, J_{\pi^*_{k^*+2}}, \ldots, \mathrm{MA}^*_{n-k^*+1}, J_{\pi^*_n}).$$

Let C_i (C_i^*, respectively) denote the completion time of the job J_{π_i} ($J_{\pi_i^*}$, respectively) in the schedule π (π^*, respectively); the makespans of π and π^* are $C_{\max}$ and $C_{\max}^*$, respectively, and (recall that $\mathrm{ML}_0 < \sum_{i=1}^n \delta_i$)

$$C_{\max} = C_{\max}^* = \sum_{i=1}^n (p_i + \delta_i) - \mathrm{ML}_0 \,. \tag{7}$$

▶ **Lemma 12.** *For every $i \ge k$ we have*

$$\sum_{j=i}^n (p_j + \delta_j) \ge \sum_{j=i}^n (p_{\pi_j^*} + \delta_{\pi_j^*}).$$

Proof. Since $p_1 + \delta_1 \le p_2 + \delta_2 \le \ldots \le p_n + \delta_n$, $\sum_{j=i}^n (p_j + \delta_j)$ is the maximum sum of processing times and machine deterioration, over all possible subsets of $(n - i + 1)$ jobs. The lemma thus holds. ◀

▶ **Theorem 13.** *The algorithm $\mathcal{A}_1$ is an $O(n \log n)$-time 2-approximation algorithm for the problem $(1|p\,\mathrm{MA}\,|\sum_j C_j)$.*

Proof. We compare the two schedules π obtained by the algorithm $\mathcal{A}_1$ and π^* an optimal schedule satisfying the properties stated in Lemma 2. Using Eq. (7) and Lemma 12, it is clear that $C_i \le C_i^*$ for each $i = n, n-1, \ldots, \max\{k, k^*\}$.

Suppose $k < k^*$, then for each i such that $k \le i < k^*$, we have

$$\begin{array}{rclcl} C_i & = & C_n - \sum_{j=i+1}^n (p_j + \delta_j) & \le & C_n^* - \sum_{j=i+1}^n (p_{\pi_j^*} + \delta_{\pi_j^*}) \\ & = & \sum_{j=1}^i (p_{\pi_j^*} + \delta_{\pi_j^*}) - \mathrm{ML}_0 & = & \sum_{j=1}^i p_{\pi_j^*} - \left(\mathrm{ML}_0 - \sum_{j=1}^i \delta_{\pi_j^*}\right) \\ & \le & \sum_{j=1}^i p_{\pi_j^*} & = & C_i^*. \end{array}$$

Therefore, we have $C_i \le C_i^*$ for each $i = n, n-1, \ldots, k$. It follows that

$$\sum_{i=k}^n C_i \le \sum_{i=k}^n C_i^* \le \mathrm{OPT}. \tag{8}$$

On the other hand, by the SPT order, the algorithm $\mathcal{A}_1$ achieves the minimum total completion time of jobs of $\{J_1, J_2, \ldots, J_{k-1}\}$. One clearly sees that in the optimal schedule π^*, the sub-total completion time of $\{J_1, J_2, \ldots, J_{k-1}\}$ is upper-bounded by OPT. Therefore,

$$\sum_{i=1}^{k-1} C_i \le \mathrm{OPT}. \tag{9}$$

Merging Eqs. (8) and (9), we conclude that the total completion time of schedule π is

$$\sum_{i=1}^{k-1} C_i + \sum_{i=k}^n C_i \le 2 \cdot \mathrm{OPT}.$$

This proves the performance ratio of 2 (which can also be shown tight on a trivial 2-job instance $I = \{J_1 = (1, \lambda), J_2 = (\lambda - 1, 1), \mathrm{ML}_0 = \mathrm{ML}^{\max} = \lambda\}$, with a large λ). The running time of the algorithm $\mathcal{A}_1$ is dominated by two times of sorting, each takes $O(n \log n)$ time. ◀

5 Concluding remarks

We investigated the single machine scheduling with job-dependent machine deterioration, recently introduced by Bock *et al.* [1], with the objective to minimize the total completion time of jobs. In the partial maintenance case, we proved the NP-hardness for the general problem, thus addressing the open problem left in the previous work. From the approximation perspective, we designed a 2-approximation, for which the ratio 2 is tight on a trivial two-job instance.

The 2-approximation algorithm is simple, but it is the first such work. Our major contribution is the non-trivial NP-hardness proof, which might appear surprising at the first glance since one has so much freedom in choosing the starting time and the duration of the maintenance activities. It would be interesting to further study the (in-)approximability for the problem. It would also be interesting to study the problem in the full maintenance case, which was shown NP-hard, from the approximation algorithm perspective. Approximating the problem in the full maintenance case seems more challenging, where we need to deal with multiple bin-packing sub-problems, while the inter-relationship among them is much complex.

References

1 S. Bock, D. Briskorn, and A. Horbach. Scheduling flexible maintenance activities subject to job-dependent machine deterioration. *Journal of Scheduling*, 15:565–578, 2012.

2 J.-S. Chen. Scheduling of non-resumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operation Research*, 190:90–102, 2008.

3 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

4 M. A. Kubzin and V. A. Strusevich. Planning machine maintenance in two-machine shop scheduling. *Operations Research*, 54:789–800, 2006.

5 C.-Y. Lee. Machine scheduling with availability constraints. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 22: 1–13. 2004.

6 C.-Y. Lee and Z.-L. Chen. Scheduling jobs and maintenance activities on parallel machines. *Naval Research Logistics*, 47:145–165, 2000.

7 C.-Y. Lee and S. Liman. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 29:375–382, 1992.

8 W. Luo, L. Chen, and G. Zhang. Approximation algorithms for scheduling with a variable machine maintenance. In *Proceedings of Algorithmic Aspects in Information and Management (AAIM 2010)*, LNCS 6124, pages 209–219, 2010.

9 W. Luo, Y. Xu, W. Tong, and G. Lin. Single machine scheduling with job-dependent machine deterioration. *CoRR*, abs/1606.04157, 2016.

10 Y. Ma, C. Chu, and C. Zuo. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58:199–211, 2010.

11 G. Mosheiov and A. Sarig. Scheduling a maintenance activity to minimize total weighted completion-time. *Computers & Mathematics with Applications*, 57:619–623, 2009.

12 X. Qi. A note on worst-case performance of heuristics for maintenance scheduling problems. *Discrete Applied Mathematics*, 155:416–422, 2007.

13 X. Qi, T. Chen, and F. Tu. Scheduling the maintenance on a single machine. *Journal of the Operational Research Society*, 50:1071–1078, 1999.

14 G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121:1–15, 2000.

15 K. Sun and H. Li. Scheduling problems with multiple maintenance activities and non-preemptive jobs on two identical parallel machines. *International Journal of Production Economics*, 124:151–158, 2010.

16 D. Xu, Y. Yin, and H. Li. Scheduling jobs under increasing linear machine maintenance time. *Journal of Scheduling*, 13:443–449, 2010.

17 S. Yang and D. Yang. Minimizing the makespan on single-machine scheduling with aging effect and variable maintenance activities. *Omega*, 38:528–533, 2010.

Approximation Algorithms for Capacitated k-Travelling Repairmen Problems

Christopher S. Martin*[1] and Mohammad R. Salavatipour†[2]

1 Department of Computing Science, University of Alberta, Edmonton, Canada
csmartin@ualberta.ca

2 Department of Computing Science, University of Alberta, Edmonton, Canada
mrs@ualberta.ca

Abstract

We study variants of the capacitated vehicle routing problem. In the *multiple depot capacitated k-travelling repairmen problem* (MD-CkTRP), we have a collection of clients to be served by one vehicle in a fleet of k identical vehicles based at given depots. Each client has a given demand that must be satisfied, and each vehicle can carry a total of at most Q demand before it must resupply at its original depot. We wish to route the vehicles in a way that obeys the constraints while minimizing the *average time* (latency) required to serve a client. This generalizes the Multi-depot k-Travelling Repairman Problem (MD-kTRP) [9, 15] to the capacitated vehicle setting, and while it has been previously studied [14, 16], no approximation algorithm with a proven ratio is known.

We give a 42.49-approximation to this general problem, and refine this constant to 25.49 when clients have unit demands. As far as we are aware, these are the first constant-factor approximations for capacitated vehicle routing problems with a latency objective. We achieve these results by developing a framework allowing us to solve a wider range of latency problems, and crafting various orienteering-style oracles for use in this framework. We also show a simple LP rounding algorithm has a better approximation ratio for the maximum coverage problem with groups (MCG), first studied by Chekuri and Kumar [10], and use it as a subroutine in our framework. Our approximation ratio for MD-CkTRP when restricted to uncapacitated setting matches the best known bound for it [15]. With our framework, any improvements to our oracles or our MCG approximation will result in improved approximations to the corresponding k-TRP problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation, capacitated, latency, group coverage

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.56

1 Introduction

In many vehicle routing scenarios, minimizing response time is a much more important objective than minimizing the distances vehicles travel. Minimizing response time is commonly required in emergency response management, routing package delivery vehicles, school-bus routing, and repairman routing, and is broadly referred to as the *travelling repairman problem* (TRP).

Many variations of this problem have been studied in both the Operations Research and approximation algorithms community. In this paper (like [10]), we consider the following

[*] Supported by Alberta Innovates Technology Futures.
[†] Supported by NSERC.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 56; pp. 56:1–56:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

version (and some interesting special cases), which we call the *multiple-depot capacitated k-travelling repairmen problem* (MD-CkTRP). It has also been referred to as the *multiple depot cumulative capacitated vehicle routing problem with multiple trips* [14, 16].

We are given a collection of k identical vehicles with capacity Q, that are initially located at k depots (roots) $R = \{r_1, r_2, \ldots, r_k\}$, a set of clients $C = \{c_1, c_2, \ldots, c_n\}$, a function $w : C \to \mathbb{Z}^{>0}$ specifying the demand of each client, and an undirected metric $d(u, v)$ over the vertices $u, v \in R \cup C$. We must find a routing for the vehicles to serve all clients in C, minimizing the average service time (or *latency*) over all clients in C, subject to the following constraints:

1. Each client must be completely served in one trip (called *unsplit delivery*).
2. Each vehicle can serve a total of at most Q demand, before it must return to its depot to resupply.

We define a *walk* to be a sequence of distinct nodes traversed in a given order, and possibly ending back at the starting node (when a walk does end back at its starting node, we call this a *tour*[1]). A *capacitated walk* is a sequence of 0 or more tours rooted at r_i, followed by an additional walk from r_i, where each tour/walk contains at most Q demand. A sequence of only tours rooted at the same node form a *flower*.

A feasible solution to MD-CkTRP is a collection F_i $(1 \leq i \leq k)$ of capacitated walks, one for each vehicle that starts at a depot r_i, and where each client c belongs to exactly one F_i. The latency of a client c that belongs to a walk rooted at r_i is the sum of the lengths of the edges traversed by the i'th vehicle before visiting c.

This general problem models many scenarios in package delivery management, where serving clients requires carrying a specific-sized package in a vehicle with limited space. One can further generalize the model to the case where vehicles have non-uniform capacities, and where each client c has a service delay $\delta(c)$, which is added to the latency of c and every client served after c by that vehicle. We call this latter version MD-kTRP with service delays.

Another problem for which we propose a new (improved) approximation algorithm is the Maximum Coverage Problem with Groups (MCG). The MCG appears as a key subroutine in various approximation algorithms, including the framework we develop. The problem is the following: suppose we are given a collection of elements $\mathcal{I}$, a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of $\mathcal{I}$, and a partition of $\mathcal{S}$ into groups $G_1, G_2, \ldots G_\ell$. The objective is to pick a collection of subsets from $\mathcal{S}$ maximizing the size of their union, such that at most one subset from each group is picked.

This problem can be approximated directly via LP-rounding if $|\mathcal{S}|$ is polynomially bounded in $\mathcal{I}$ (*e.g.* with pipage rounding [1]). It is special case of submodular function optimization subject to matroid constraints [6], but in those settings the algorithm has a running time that is polynomial in $|\mathcal{S}|$ while the version we consider can have $|\mathcal{S}|$ exponentially large in $|\mathcal{I}|$; in this case we are instead given an implicit representation of $\mathcal{S}$. In such settings, suppose we were given an oracle $\mathcal{A}(i, \theta)$ that takes as input a group index i and a weight function $\theta : \mathcal{I} \to \{0, 1\}$, and returns some subset $S_j \in G_i$ such that $\sum_{e \in S_j} \theta(e)$ is maximized. Call $\mathcal{A}$ a $(1/\rho)$-approximate oracle if it returns a subset $S_j \in G_i$ such that $\sum_{e \in S_j} \theta(e) \geq \frac{1}{\rho} \max_{S' \in G_i} \sum_{e \in S'} \theta(e)$ (*i.e.* the returned subset covers at least a $\frac{1}{\rho}$-fraction of the optimal number of elements). In this paper, we focus on approximating MCG for which $|\mathcal{S}|$ can be exponential in $|\mathcal{I}|$ and we have access to an oracle $\mathcal{A}$ as above; this version will be useful in our approximation algorithms for MD-CkTRP. We will therefore never state $\mathcal{S}$ explicitly, instead only giving the oracle $\mathcal{A}$ and the groups G_i defining the input instance.

[1] This differs slightly from the typical definition of a tour, since a tour here must be composed of *exactly* one cycle.

For the approximation we develop, we will require a *weighted* version of $\mathcal{A}$; that is, the input θ will instead be a function returning any non-negative value. Many oracles (including the ones we present) can be converted to this form with only a small loss in approximation using standard techniques, such as scaling weights and duplicating elements.

1.1 Related Work

The special case of $k = 1$ and $Q = \infty$ for the MD-CkTRP is the Minimum Latency or Travelling Repairman problem, which has been studied extensively [4, 3, 9, 13, 18]. This case is known to be APX-hard in general metrics [5], and the 3.59-approximation of Chaudhuri *et al.* [9] is the best known for this case. The special case where the metric is an edge-weighted tree is also known to be NP-hard [17], and a PTAS for this was only recently found [18].

For the uncapacitated k-vehicle situation where $r_1 = r_2 = \cdots = r_k$ and $Q = \infty$, an 8.497-approximation was known [11]; this was recently improved to 7.183 [15]. For the multi-depot uncapacitated case, Chekuri and Kumar [10] proved a 24-approximation.[2] This was recently improved to 8.497 by Post and Swamy [15]. This improvement came from using a time-index configuration LP that was introduced in [8] for the single vehicle case, while extending it to the multi-vehicle setting and introducing an LP rounding algorithm.

The MCG was first considered by Chekuri and Kumar [10] in the context of their approximation for the MD-kTRP. They developed the first approximation for the problem given a $(1/\rho)$-approximate oracle, obtaining a simple greedy $1/(\rho + 1)$-approximation. The submodular maximization problem with matroid constraints generalizes MCG: the instance can be represented by a monotone submodular function $f(S)$ denoting the number of elements covered by the set S, and a partition matroid $\mathcal{M}$ over the sets in $\mathcal{S}$ that define the groups. It was shown in [6] how to obtain a $(1 - 1/e)$-approximation for this problem with running time polynomial in $|\mathcal{S}|$ and $|\mathcal{I}|$. When $\mathcal{S}$ is not given explicitly and $|\mathcal{S}|$ is exponentially large in $|\mathcal{I}|$, the result of Chekuri and Kumar [10] is currently the best known.

To the best of our knowledge, *no* approximation algorithm for any capacitated variant of the travelling repairmen problem has been developed. Our specific problem has been studied in the operations research community, but only heuristic solutions are currently known [14, 16].

1.2 Our Results

We solve the capacitated variant of the travelling repairmen problem by building off of and extending the techniques used previously for the multi-depot travelling repairmen problem and for capacitated vehicle routing. Our algorithm uses ideas from both [10] and [15], in particular the greedy combinatorial algorithm of [10], coupled with a new LP-based approximation algorithm for the MCG inspired by [15]. One feature of our algorithm is that if we restrict it to the case of $Q = \infty$ (*i.e.* the uncapacitated setting), we obtain an approximation ratio matching the best known bound for that setting [15].

To achieve this, we develop a modular framework (Theorem 5) that uses a user-provided oracle as a subroutine to solve different versions of the multi-depot travelling repairmen problem. The exact problem we solve is captured by a collection of *feasible walks*, which are separated over using the provided oracle as a black-box. Given such an oracle, we can build

[2] The approximation ratio was stated in [10] to be 12, but due to a technical issue in their analysis, they were off by a factor of 2.

$O(1)$-approximation algorithms for the various latency problems we consider. We obtain the following results with this approach:

▶ **Theorem 1.** *There is a* 25.49*-approximation to the unit-demand capacitated multi-depot* k*-TRP.*

▶ **Theorem 2.** *There is a* 42.49*-approximation to the unsplit-delivery capacitated multi-depot* k*-TRP.*

We show a simple LP-rounding gives an improved approximation for MCG, which we use as a subroutine in our framework:

▶ **Theorem 3.** *There is a* $(1 - e^{-1/\rho})$*-approximation to the MCG given a* $(1/\rho)$*-approximate oracle.*

Theorems 1 and 2 are the first (constant) approximations for the MD-CkTRP, and also extend to more general cases where we have non-uniform vehicles capacities. We may additionally add service delays $\delta(c)$ at each client with an extra $+0.5$ loss in the ratio. These extensions are covered in Section 5. The framework we develop to prove Theorems 1 and 2 is presented as Theorem 5. The algorithm we give to prove that theorem finds progressively longer rooted *flowers* from each depot that cover a large number of clients, where the length of these flowers is bounded against a rooted *walk*. Suppose that C is the set of clients to be served/covered by a walk from r_i, and B is a given budget on the length of the walk (depending on our problem, walks might be capacitated). The *single-depot orienteering problem* (SD-OP) is to find such a walk with total cost at most B starting at r_i that covers as many (distinct) clients of C as possible.

We can generalize the notion of capacitated walks/tours by giving a set $\mathcal{W}_i$ that contains all r_i-rooted walks vehicle i is allowed to traverse for the given problem. A capacitated walk/tour is then a walk/flower built using only walks from $\mathcal{W}_i$. We call these $\mathcal{W}_i$*-restricted walks/flowers.* Our approach centres around a black-box algorithm to (approximately) solve the SD-OP problem over the set of walks $\mathcal{W}_i$; that is, only walks in $\mathcal{W}_i$ are considered feasible for vehicle i.

▶ **Definition 4.** A $(1/\rho, \gamma)$-approximation to the $\mathcal{W}_i$-restricted SD-OP problem is an algorithm that finds a walk of cost $\leq \gamma B$ covering at least a $1/\rho$-fraction of the number of clients on an optimal walk.

If this black-box returns a flower rather than a walk, but with cost still bounded by the optimal walk, then we call this a $(1/\rho, \gamma)$*-flower* approximation. We use this algorithm as an oracle to find interesting walks/flowers over the sets $\mathcal{W}_i$ defined by the problem. With this, we obtain the following result:

▶ **Theorem 5.** *Let* $\mathcal{W}_i$ *be the set of all* r_i*-rooted walks that can be feasibly traversed by vehicle* i*. Then for constants* ρ, γ*, there is an* $O(1)$*-approximation algorithm to the* $\mathcal{W}_i$*-restricted multi-depot* k*-TRP, if we have a* $\mathcal{W}_i$*-restricted* $(1/\rho, \gamma)$*-flower approximation to the* $\mathcal{W}_i$*-restricted SD-OP.*

When $\mathcal{W}_i$ is the set of *all* possible walks from depot r_i, we are solving the uncapacitated multi-depot k-TRP (MD-kTRP), studied in [10, 15]. If we restrict $\mathcal{W}_i$ to only capacitated walks (with capacity Q), we are solving the unsplit MD-CkTRP variant. Using Theorem 5, we can find a constant-factor approximation to the MD-CkTRP given an oracle satisfying definition 4 that returns flowers. We give oracles for the unit-demand and unsplit-delivery cases in Section 4, which when combined with Theorem 5 yields Theorems 1 and 2.

We start by proving Theorem 3 in Section 2. We then proceed to prove Theorem 5 in Section 3, by showing how to combine ideas from [10], [9], and [15] to create a combinatorial approximation algorithm for the problem, which requires solving an MCG instance as a subroutine. Many of the proofs omitted in this extended abstract appear in the full version of the paper.

An alternative approach for approximating latency problems that avoids explicitly solving an MCG instance was introduced and expanded in [8, 15]. They solve a time-indexed configuration LP directly for the multi-depot latency problem, and use randomized rounding to obtain the final collection of tours. Our approach is in fact equivalent to theirs for that specific problem; the combination of our greedy algorithm and MCG LP yields their time-indexed LP. By writing the configuration LP for a more general covering problem (namely MCG) and using that as a subroutine in our latency algorithm we feel that the approach becomes more easily adaptable to different problems beyond latency. In a sense, we unify and generalize the combinatorial algorithm of [10] and the LP rounding algorithm of [15] in a framework using MCG rounding.

2 A $(1 - e^{-1/\rho})$-Approximation for MCG

We can express an instance of the MCG as an integer configuration program. For item $e \in \mathcal{I}$ and group G_i, let x_e^i be a binary variable indicating whether item e is being covered by a set from group G_i or not. For a set $S \in \mathcal{S}$, let z_S be a binary variable indicating whether set S is chosen to form a part of the solution. The linear relaxation of the configuration program is given as (LP) (and its dual as (DP)).

$$\begin{aligned} \max \quad & \sum_{e,i} x_e^i & & & \text{(LP)} \\ \text{s.t.} \quad & \sum_i x_e^i \le 1 & \forall e & \quad (\alpha_e) & (1) \\ & \sum_{S \in G_i} z_S \le 1 & \forall i & \quad (\beta_i) & (2) \\ & \sum_{S \in G_i : S \ni e} z_S \ge x_e^i & \forall e, i & \quad (\theta_e^i) & (3) \\ & x, z \ge 0. \end{aligned}$$

$$\begin{aligned} \min \quad & \sum_e \alpha_e + \sum_i \beta_i & & \text{(DP)} \\ \text{s.t.} \quad & \alpha_e + \theta_e^i \ge 1 & \forall e, i & \quad (4) \\ & \sum_{e \in S} \theta_e^i \le \beta_i & \forall i, S \in G_i & \quad (5) \\ & \alpha, \beta, \theta \ge 0. \end{aligned}$$

For every set $S \in G_i$ we use $\theta^i(S)$ to denote $\sum_{e \in S} \theta_e^i$. As stated before, we assume we are given a approximate *weighted* oracle $\mathcal{A}(i, \theta)$; that is, for each group i, given θ_e^i on elements it will find a set S in group G_i such that $\theta^i(S) \ge 1/\rho \max_{S' \in G_i} \theta^i(S')$.

$\mathcal{A}$ will become our approximate separation oracle for the dual. Solving an exponential size LP approximately using such an oracle is a standard technique following from the work of Carr and Vempala [7]. We briefly describe how to obtain a good solution following the more recent presentation in [12].

Define the polytope $\mathcal{P}(\upsilon, a) = \{(\alpha, \beta, \theta) : (4), (5), \sum_e \alpha_e + a \sum_i \beta_i \le \upsilon\}$. With our ρ-approximate (weighted) oracle $\mathcal{A}$, given some υ and point (α, β, θ), we can certify that either $(\alpha, \rho\beta, \theta) \in \mathcal{P}(\upsilon, 1)$, or give a hyperplane certifying that $(\alpha, \beta, \theta) \notin \mathcal{P}(\upsilon, \rho)$, as follows. For each i, run $\mathcal{A}$ with element weights θ_e^i. If the returned set S has weight $\theta(S) > \beta_i$, then since $\theta^i(S) \ge (1/\rho) \max_{S' \in G_i} \theta^i(S')$, we return the constraint (5) corresponding to i, S as the separating hyperplane. The other constraints can be checked trivially. If no constraint is violated, we must have $(\alpha, \rho\beta, \theta) \in \mathcal{P}(\upsilon, 1)$, and so the ellipsoid algorithm will certify in polynomial time that either $\mathcal{P}(\upsilon, \rho) = \emptyset$, or give a point $(\alpha, \rho\beta, \theta) \in \mathcal{P}(\upsilon, 1)$. Note that $\mathcal{P}(OPT_{LP}, 1)$ defines the collection of optimum solutions for (DP), and so OPT_{LP} is the smallest υ such that $\mathcal{P}(\upsilon, 1) \neq \emptyset$; we can determine this value by binary search on υ.

Suppose we run the ellipsoid algorithm with input $OPT_{LP} - \epsilon$ for any $\epsilon > 0$. This yields a certificate showing $\mathcal{P}(OPT_{LP} - \epsilon, \rho) = \emptyset$, consisting of polynomially-many separating hyperplanes, including the inequality $\sum_e \alpha_e + \rho \sum_i \beta_i \leq OPT_{LP} - \epsilon$. Consider the dual polytope of $\mathcal{P}(\upsilon, a)$: $\mathcal{Q}(\upsilon, a) = \{(x, z) : (1), \sum_{S \in G_i} z_S \geq a, (3), \sum_{e,i} x_e^i \geq \upsilon\}$. By duality, the certificate corresponds to a point $(x, z) \in \mathcal{Q}(OPT_{LP} - \epsilon, \rho)$ with polynomially-many non-zero variables. Note that $(x/\rho, z/\rho)$ is a feasible (approximate) solution to (LP); further, (x, z) is *almost* a feasible solution with objective value $OPT_{LP} - \epsilon$ that only violates (2).[3] This property will be crucial to our rounding scheme.

2.1 Rounding a Solution

To round the solution $(x/\rho, z/\rho)$, we adapt the ideas used by Ageev and Svirendenko [1] for proving an integrality gap for the standard Maximum Coverage problem. Observe that (LP) is equivalent to the following linear program:

$$\max \sum_e \min\left(1, \sum_{S \ni e} z_S\right) \tag{LP2}$$

$$\text{s.t.} \sum_{S \in G_i} z_S \leq 1 \qquad \forall i \tag{6}$$

$$z_S \in [0, 1].$$

Constraints (1) and (3) have been rewritten as the minimum in the objective, and so given a fractional solution $(x/\rho, z/\rho)$ to (LP), we can obtain a solution z/ρ to (LP2) of equal objective value (*i.e.* at least OPT_{LP}/ρ). (LP2) is now in pipage rounding form as defined by [1], and so we can apply their pipage rounding algorithm to obtain an integer solution $\bar{z}$. We now just need to bound the integrality gap.

Let $L(z) = \sum_e \min(1, \sum_{S \ni e} z_S)$. We will define a function $F(z)$ that is both ϵ-convex on the input z/ρ (as defined in [1]) and satisfies the following F/L *lower bound condition.* Suppose that, for an optimal (fractional) solution $\check{z}$, our sub-optimal solution z/ρ has the property that $F(z/\rho) \geq L(\check{z})/\alpha$ for some α. Let F^* be the value of an optimal *integer* solution to (LP2); if L and F are coincident on binary inputs, then $L(\check{z}) \geq F^*$, and so this new condition would imply we have an α-approximation after pipage rounding. We claim that the function $F(z) = \sum_e (1 - \prod_{S \ni e}(1 - z_S))$ satisfies these conditions.

▶ **Lemma 6.** *$F(z)$ satisfies the F/L lower bound condition.*

▶ **Lemma 7.** *$F(z/\rho)$ is ϵ-convex.*

We can therefore apply pipage rounding on the fractional solution z/ρ, using the function F to guide the algorithm. This yields a deterministic $(1 - e^{-1/\rho})$-approximation to the MCG problem.

[3] We omit the ϵ for the remainder of this discussion for clarity.

3 Proof of Theorem 5

We present the framework by generalizing and modifying the combinatorial algorithm of Chekuri and Kumar [10] to suit our redefined problem. The key subroutine of their algorithm is an approximation for the MCG, which they use to determine a set of tours to "stitch" together for routing vehicles from each depot. Their algorithm uses an oracle as a black-box to solve an orienteering-style problem in order to find "good" tours to use in their MCG instance. In [10] these tours are built from an ℓ-MST, using the algorithm from [9]; we will instead use the user-provided black-box oracle for this task and show that we still obtain a good approximation.

Recall we are given as input a set of clients C, a set of k depots R, a vehicle initially located at each depot, and a metric distance function d. We wish to find $\mathcal{W}_i$-restricted walks for each vehicle i starting at their respective depots that collectively visit all clients, and minimize the total latency of all walks. The latency of a walk W that starts at root r and visits clients $c_1, c_2, \dots c_m$ is given by $\sum_{i=1}^{m} d_W(r, c_i)$, where d_W is the distance along the walk between two points.

The computation is split up into *phases*, with each phase given a budget with which to cover as many clients as possible. The latency of the clients we cover in this phase can then be bounded by the total budget we have spent in this phase and all prior phases. Let $j \geq 1$ be the current phase, and let C_j^u be the set of uncovered clients at the start of phase j. Let $\tau > 1$ be some global constant to be chosen later, $U \in [0, 1)$ be a number chosen uniform randomly, and $b = \tau^U$.

We define the *multi-depot group orienteering problem* (MD-GOP) as follows: given a subset of clients C' to be visited and a hard budget B, find for each depot $r_i \in R$ a *walk* of total length at most B such that all walks returned collectively cover as many (distinct) clients in C' as possible. We define $\mathcal{C}(C', B)$ to be some algorithm that solves the $\mathcal{W}_i$-restricted version of this problem approximately. $\mathcal{C}$ is a $\mathcal{W}_i$-restricted $(1/\rho, \gamma)$-flower approximation if it finds a collection of k *flowers* rooted at the depots r_i, such that each costs at most γB and together they cover at least a $\frac{1}{\rho}$-fraction of the vertices covered by an optimum MD-GOP (walk) solution. Note that for the case of uncapacitated vehicles, a flower is simply a single tour. Given this subroutine, the algorithm for phase j is as follows:

```
function Do-Phase($j$)
    Run $\mathcal{C}(C_j^u, b\tau^j)$ with clients $C_j^u$ and budget $b\tau^j$.
    Traverse the returned flower for each $r_i$ in either direction, chosen uniformly at random.
    Remove all covered clients from $C_j^u$.
end function
```

We build a bi-criteria $(1 - e^{-1/\rho}, \gamma)$-flower approximation algorithm $\mathcal{C}$, given a user-provided oracle $\mathcal{A}$ as per the Theorem, using our MCG approximation (Theorem 3). Let S_W be the set of vertices contained in the walk W. Let $\mathcal{W}_i^B$ be the set of walks in $\mathcal{W}_i$ of length at most B. Let $G_i = \{S_W : W \in \mathcal{W}_i^B\}$ be the group of all $\mathcal{W}_i$-restricted walks of total length at most B. This forms a valid MCG instance, whose solution yields a collection of k walks, each of cost at most B that collectively cover as many clients as possible.

This instance can be approximately solved as follows. Using $\mathcal{A}$, we can find flowers in G_i covering as many new clients as possible, relative to the optimal walk. Since $\mathcal{A}$ finds a flower covering at least a $1/\rho$-fraction of the optimal number of new clients, by Theorem 3 the final solution covers a $(1 - e^{-1/\rho})$-fraction of the optimal number of clients, exceeding the budget for each flower by a factor of γ. Thus, $\mathcal{C}$ is a $(1 - e^{-1/\rho}, \gamma)$-flower approximation.

3.1 Analysis

We now prove that we have a constant-factor approximation to the $\mathcal{W}_i$-restricted multi-depot k-TRP, thus completing the proof of Theorem 5. Fix an optimal solution OPT, and let O_j denote the set of clients in OPT that have latency $\leq b\tau^j$. Let C_j^v be the clients we have visited by the *end* of phase j. We define C_0^v to be the empty set.

▶ **Lemma 8.** *At the end of phase j, we have covered at least $(1 - e^{-1/\rho})|O_j - C_{j-1}^v|$ clients.*

Let n_j^{OPT} be the number of clients in OPT whose latency is *more* than $b\tau^j$, and let n_j be the number of clients that were left uncovered at the end of phase j. For $j \leq 0$, we define n_j^{OPT} and n_j to be n. Let B_j be the budget of phase j; for $j \geq 1$ this is $b\tau^j$, and for $j \leq 0$ we define it to be 0. For notational convenience, define $\Delta_j = B_j - B_{j-1}$.

▶ **Lemma 9.** *For all j, $n_j \leq e^{-1/\rho} n_{j-1} + (1 - e^{-1/\rho}) n_j^{OPT}$.*

▶ **Lemma 10.** *In expectation, the latency of our solution is at most:*

$$\frac{\gamma(\tau+1)}{2(\tau-1)} \sum_{j\geq 1} B_j (n_{j-1} - n_j) = \frac{\gamma(\tau+1)}{2(\tau-1)} \sum_{j \geq 1} n_{j-1} \Delta_j. \tag{OUR-UB}$$

▶ **Lemma 11.** *In expectation, the latency of OPT is at least:*

$$\frac{\ln \tau}{\tau - 1} \sum_{j \geq 1} n_{j-1}^{OPT} \Delta_j. \tag{OPT-LB}$$

Proof of Theorem 5. By summing Lemma 9 over all j, we see that

$$\begin{aligned} \sum_{j\geq 1} \Delta_j n_{j-1} &\leq e^{-1/\rho} \left(\sum_{j \geq 1} \Delta_j n_{j-2} + (e^{1/\rho} - 1) \sum_{j\geq 1} \Delta_j n_{j-1}^{OPT} \right) \\ &= \tau e^{-1/\rho} \sum_{j \geq 1} \Delta_j n_{j-1} + \frac{(1 - e^{-1/\rho})(\tau - 1)}{\ln \tau} \frac{\ln \tau}{\tau - 1} \sum_{j\geq 1} \Delta_j n_{j-1}^{OPT} \\ \implies \text{(OUR-UB)} &\leq \frac{\gamma(\tau+1)(1-e^{-1/\rho})}{2\ln(\tau)(1 - \tau e^{-1/\rho})} \text{(OPT-LB)}. \end{aligned}$$

Our algorithm is therefore a $\frac{\gamma(\tau+1)(1-e^{-1/\rho})}{2\ln(\tau)(1-\tau e^{-1/\rho})}$-approximation for any constant $1 < \tau < e^{1/\rho}$, satisfying the requirements of the theorem. ◀

3.2 An Uncapacitated Oracle

We now give a $(1, 2 + \epsilon)$-approximate oracle $\mathcal{A}$ for the uncapacitated multi-depot k-TRP (*i.e.* $\mathcal{W}_i$ is *all* possible walks from r_i). This oracle is used in [10] and earlier works for single-depot latency problems. First we describe an unweighted oracle (*i.e.* each node is assigned $\theta_e^i \in \{0, 1\}$); we later describe how to extend it to the weighted version.

Using the algorithm in [9] for finding an ℓ-MST, we find a tree that covers at least as many clients as the optimal r_i-rooted walk with budget B, and costs at most $(1 + \epsilon)$ times the optimal walk (see Theorem 1 in [9]). Since the optimal walk costs at most B, we find the largest ℓ such that the returned ℓ-MST has cost at most $(1 + \epsilon)B$. Such a tree will cover at least as many clients as the optimal walk. Double the edges of this tree, and convert to a tour by shortcutting past repeated vertices.

For the case that we have weights on the nodes, at a loss of at most $1-\epsilon'$ on the total weight of nodes we can cover, we can reduce the problem to the unweighted case by scaling and discarding nodes with very small weight (so that $\frac{\max_e \theta^i_e}{\min_{e'} \theta^i_{e'}} \in O(n^2)$) and then duplicating vertices. This gives a $(1-\epsilon', 2+\epsilon)$-approximate (weighted) oracle $\mathcal{A}$ (for any $\epsilon, \epsilon' > 0$).

This leads to the following result for the uncapacitated MD-kTRP, which matches the current-best given in [15].

▶ **Corollary 12.** *There is an* 8.497*-approximation to the uncapacitated multiple depot k-TRP* ($\tau \approx 1.405$)*.*

4 Capacitated Oracles and Proofs of Thms. 1 and 2

Previously, we showed that to solve the MD-CkTRP, we can use Theorem 5 and restrict $\mathcal{W}_i$ to only capacitated r_i-rooted walks. We thus need to find an oracle that can solve the related orienteering problem over this set of walks. Using standard techniques as before, we can reduce the weighted version of the problem (with weights on the nodes) to the unweighted version, which we present below.

The problem the oracle must solve is the following, which we call the *unsplit capacitated orienteering problem* (U-COP). We are given a collection of clients C, a root node r, a budget B, a vehicle capacity Q, a client demand function $w : C \to \mathbb{Z}^{>0}$, and an undirected distance metric d. We wish to find an r-rooted capacitated walk of total length at most B, where r must be re-visited after serving at most Q client demand, and we wish to cover as many clients as possible. Call the optimal number of clients ℓ^{OPT}, and let $d(W)$ denote the length of the walk W with respect to the metric d, and similarly for flowers and tours.

We give a $(1, 10+\epsilon)$-flower approximation algorithm, where the flower we find has total cost at most $(10+\epsilon)B$ and collectively covers ℓ^{OPT} clients, respecting the capacity constraint. We also consider a special case where $w(c) = 1$ for all $c \in C'$; we call this the *unit-demand capacitated orienteering problem* (1-COP). With this demand constraint, we can improve the above ratio to $(1, 6+\epsilon)$.

An optimum solution to either problem consists of a sequence of tours (each visiting at most Q demands) followed by at most one walk of total demand at most Q. If we convert that last walk to a tour by returning to the root, we obtain a capacitated flower of cost at most $2B$. We will restrict our attention to finding such flowers.

The algorithms for 1-COP and U-COP are very similar, so we describe both simultaneously. If there is a difference between the two algorithms, we place the difference for U-COP in (parentheses). It will be useful to consider the input metric as the complete graph $G = (V, E)$ with $V = C \cup \{r\}$ and edge costs $c_G(uv) = d(u, v)$ for all $uv \in E$.

1. Let G^* be a new graph obtained from G by adding a "terminal" client c' to each $c \in C$ and edge cc' between client c and its new terminal client; the cost of these new edges will be $\frac{1}{Q}d(r, c)w(c)$ (for U-COP, use $\frac{2}{Q}d(r, c)w(c)$). Let G^T be the "terminal" graph obtained from the metric completion of G^*, with all non-terminal client vertices removed.
2. Using the ℓ-MST approximation of Chaudhuri *et al.* [9], find a tree of cost at most $3B+\epsilon$ ($5B+\epsilon$) in G_T that covers as many terminals as possible. Doubling this tree produces a tour; call this tour O.
3. Convert O back into a tour in G^* that visits the same number of terminal clients of no greater cost (always possible since G^T is the metric completion of G^*). Prune away the terminals and short-cut to obtain a new tour O' in G.
4. Let G' be a unit-weighted complete graph containing r and $w(c)$ copies of each client $c \in C$; let Ω_c denote the copies of c in G' (so $|\Omega_c| = w(c)$). If clients u, v were distance

$c_G(uv)$ apart in G, then for all vertices $i \in \Omega_u, j \in \Omega_v$, $c_{G'}(ij) = c_G(uv)$. Define edge costs to r similarly. For each $i, j \in \Omega_c$, let $c_{G'}(ij) = 0$.[4]

5. Convert O' into a split-delivery, capacitated flower as follows. Map O' onto G' without increasing the cost while covering $\sum_{c \in O'} w(c)$ clients (possible by construction). Number the vertices of this tour in the order they are visited, and pick a random offset R in the range $[1, Q]$. Walk along the tour starting at R, and cut away a strip of the tour every Q vertices (short-cutting past r). Add an edge at each end of a strip back to r, to make each strip an r-rooted tour. [5]
6. For the 1-COP, return this capacitated flower. For the U-COP, we can "unsplit" our solution as follows. Note that if some client's delivery is split, it will be covered by at most two tours; remove any such c from both tours and place it in its own separate tour. Return the resulting capacitated flower.

We now prove the above procedure is in fact a good approximation for both problems. Consider a fixed optimal capacitated walk W^{OPT} of cost OPT which covers a set of clients C^{OPT}; let $\ell^{OPT} = |C^{OPT}|$. Let TSP^{OPT} be an optimal TSP tour that covers the clients C^{OPT}, and let F^{OPT} be an optimal capacitated flower; one must exist with cost at most $2OPT$.

We utilize two classic results in capacitated vehicle routing.

▶ **Lemma 13.** *The following inequalities hold for the 1-COP and U-COP:*

$$d(TSP^{OPT}) \leq d(F^{OPT}) \leq 2OPT \tag{7}$$

$$\frac{2}{Q} \sum_{c \in C^{OPT}} d(c, r) \leq d(F^{OPT}) \leq 2OPT. \tag{8}$$

(8) can be strengthened for the case where $w(c)$ is any integer ≥ 1:

▶ **Lemma 14.** *We have the following additional inequality for the U-COP:*

$$\frac{2}{Q} \sum_{c \in C^{OPT}} d(r, c)w(c) \leq d(F^{OPT}) \leq 2OPT. \tag{9}$$

For each set of clients H define $S_H = \frac{1}{Q} \sum_{c \in H} d(r, c)w(c)$; by (9), $S_{C^{OPT}} \leq B$. Note that W^{OPT} can be converted into a walk in G^* of cost at most $B + 2S_{C^{OPT}} \leq 3B$ (for U-COP, $B + 4S_{C^{OPT}} \leq 5B$) that visits C^{OPT} and the corresponding terminal clients. We can further convert W^{OPT} into a walk that visits *only* terminal clients (and so a walk in G^T), of no greater cost, that covers ℓ^{OPT} terminals. Thus, the ℓ-MST approximation of Chaudhuri *et al.* [9] will find a tree of cost at most $3B + \epsilon$ $(5B + \epsilon)$ in G_T that covers ℓ^{OPT} terminals.[6] From this and by construction, the tour O' must have cost at most $6B - 2S_{O'}$ $(10B - 4S_{O'})$.

The expected cost of the extra edges added in step 5 is $2S_{O'}$, so some offset R exists such that we pay at most this amount. Thus, we can cut up O' into smaller tours covering at most Q demand, with total cost $6B$ $(10B - 2S_{O'})$, yielding a $(1, 6 + \epsilon)$-approximation to the 1-COP. For the U-COP, the cost of the extra tours in step 6 is also $2S_{O'}$, yielding a $(1, 10 + \epsilon)$-approximation. Extending these results to the weighted case, we obtain a $(1 - \epsilon', 6 + \epsilon)$-approximation for 1-COP and $(1 - \epsilon', 10 + \epsilon)$-approximation for U-COP for any $\epsilon', \epsilon > 0$.

[4] This construction was first described in [2], and can be used to prove Inequality 9.

[5] If Q is not poly-bounded, note there is a simple poly-time algorithm to do this that avoids explicitly building the graph and trying more than $|V|$ values of R.

[6] We omit the ϵ from the rest of the discussion for clarity.

Proof of Theorems 1 and 2. Combining Theorem 5 with the $(1-\epsilon', 6+\epsilon)$-approximation for 1-COP yields a 25.49-approximation to the unit-demand MD-CkTRP; similarly, combining Theorem 5 with the $(1-\epsilon', 10+\epsilon)$-approximation for U-COP yields a 42.49-approximation to the MD-CkTRP ($\tau \approx 1.616$). ◀

5 Extensions to MD-CkTRP

We briefly consider two extensions to our problem - non-uniform vehicle capacities, and service delays. In the first case, suppose vehicle i has capacity Q_i. Adjust the definition of $\mathcal{W}_i$ to be all walks of capacity $\leq Q_i$ instead of Q; note that the approximation guarantees of our oracles do not depend on the capacity of the vehicle. Thus, our results extend to vehicles with non-uniform capacities.

To handle service delays, suppose each client c has a service time $\delta(c) \geq 0$, which adds to the time a vehicle must spend traversing its walk (we assume $\delta(r_i) = 0$ for each root r_i). We still wish to minimize the total latency of all clients visited. Define a new metric $d'(u, v) = d(u, v) + \frac{\delta(u)+\delta(v)}{2}$. Solve the MD-C$k$TRP for the new instance (with metric d'). The latency of each node u in the solution returned will be the sum of the edge-lengths, plus the sum of the delays of all the nodes visited before u, plus $\delta(u)/2$. Thus, at an extra loss of $+0.5$ in the approximation, the solution will be a solution for the corresponding problem with service delays.

6 Concluding Remarks

We presented a general framework to obtain a constant approximation algorithm for the capacitated multi-depot k-TRP, using bi-criteria approximation algorithms for orienteering style problems, giving the first constant approximations for MD-CkTRP. A consequence of this approach is if our oracles for single-depot (or multi-depot) orienteering are improved, we would have improved approximations for multi-depot (capacitated and uncapacitated) k-TRP.

References

1 A. Ageev and M. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

2 Kemal Altinkemer and Bezalel Gavish. Heursitics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

3 Aaron Archer and Anna Blasiak. Improved approximation algorithms for the minimum latency problem via prize-collecting strolls. *21st ACM SODA*, pages 429–447, 2010.

4 Aaron Archer, Asaf Levin, and David P Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM Journal on Computing*, 37(5):1472–1498, 2008.

5 A. Blum, P. Chalasani, B. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. *26th ACM STOC*, pages 163–171, 1994.

6 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

7 B. Carr and S. Vempala. Randomized meta-rounding. *In Proceedings of STOC*, 2000.

8 Deeparnab Chakrabarty and Chaitanya Swamy. Facility location with client latencies: Linear-programming based techniques for minimum-latency problems. *15th IPCO*, pages 92–103, 2011.

9 Kamalika Chaudhuri, Godfrey Brighten, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. *44th IEEE-FOCS*, pages 36–45, 2003.

10 Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques*, pages 72–83, 2004.

11 Jittat Fakcharoenphol, Chris Harrelson, and Satish Rao. The k-traveling repairman problem. *14th ACM-SIAM SODA*, pages 655–664, 2003.

12 Zachary Friggstad and Chaitanya Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. *In Proceedings of STOC*, pages 744–753, 2014.

13 Michel Goemans and Jon Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82(1-2):111–124, 1998.

14 Jens Lysgaard and Sanne Wohlk. A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem. *European Journal of Operational Research*, 236(3):800–810, 2014.

15 Ian Post and Chaitanya Swamy. Linear-programming based approximation algorithms for multi-vehicle minimum latency problems. *26th ACM-SIAM SODA*, pages 512–531, 2015.

16 Juan Carlos Rivera, H. Murat Afsar, and Christian Prins. A multistart iterated local search for the multitrip cumulative capacitated vehicle routing problem. *Computational Optimization and Applications*, 61(1):159–187, 2015.

17 René Sitters. The minimum latency problem is NP-hard for weighted trees. *IPCO*, 2337:230–239, 2002.

18 René Sitters. Polynomial time approximation schemes for the travelling repairman and other minimum latency problems. *25th ACM-SIAM SODA*, 2014.

Scaling and Proximity Properties of Integrally Convex Functions*

Satoko Moriguchi[1], **Kazuo Murota**[2], **Akihisa Tamura**[3], **and Fabio Tardella**[4]

1 **Department of Business Administration, Tokyo Metropolitan University, Hachioji, Japan**
satoko5@tmu.ac.jp

2 **Department of Business Administration, Tokyo Metropolitan University, Hachioji, Japan**
murota@tmu.ac.jp

3 **Department of Mathematics, Keio University, Yokohama, Japan**
aki-tamura@math.keio.ac.jp

4 **Department of Methods and Models for Economics, Territory and Finance, Sapienza University of Rome, Roma, Italy**
fabio.tardella@uniroma1.it

Abstract

In discrete convex analysis, the scaling and proximity properties for the class of L$^\natural$-convex functions were established more than a decade ago and have been used to design efficient minimization algorithms. For the larger class of integrally convex functions of n variables, we show here that the scaling property only holds when $n \leq 2$, while a proximity theorem can be established for any n, but only with an exponential bound. This is, however, sufficient to extend the classical logarithmic complexity result for minimizing a discretely convex function in one dimension to the case of integrally convex functions in two dimensions. Furthermore, we identified a new class of discrete convex functions, called directed integrally convex functions, which is strictly between the classes of L$^\natural$-convex and integrally convex functions but enjoys the same scaling and proximity properties that hold for L$^\natural$-convex functions.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Discrete optimization, discrete convexity, proximity theorem, scaling algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.57

1 Introduction

The proximity-scaling approach is a fundamental technique in designing efficient algorithms for discrete or combinatorial optimization. For a function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ in integer variables and a positive integer α, called scaling unit, the α-scaling of f is the function f^α defined by $f^\alpha(x) = f(\alpha x)$ $(x \in \mathbb{Z}^n)$. A proximity theorem is a result guaranteeing that a (local) minimum of the scaled function f^α is close to a minimizer of the original function f. The scaled function f^α is simpler in shape, and hence easier to minimize, whereas

* The research was initiated at the Trimester Program "Combinatorial Optimization" at Hausdorff Institute of Mathematics, 2015. This work is supported by The Mitsubishi Foundation, CREST, JST, and JSPS KAKENHI Grant Numbers 26350430, 26280004, 24300003, 16K00023.

the quality of the obtained minimizer of f^α as an approximation to the minimizer of f is guaranteed by a proximity theorem. The proximity-scaling approach consists in applying this idea for a decreasing sequence of α, often by halving the scale unit α. A generic form of a proximity-scaling algorithm may be described as follows, where K_∞ denotes the ℓ_∞-size of the effective domain of f, and $B(n, \alpha)$ denotes the proximity bound in ℓ_∞-distance.

S0: Find an initial vector x with $f(x) < +\infty$, and set $\alpha := 2^{\lceil \log_2 K_\infty \rceil}$.
S1: Find a vector y with $\|\alpha y\|_\infty \leq B(n, \alpha)$ that is a (local) minimizer of $\tilde{f}(y) = f(x + \alpha y)$, and set $x := x + \alpha y$.
S2: If $\alpha = 1$, then stop (x is a minimizer of f).
S3: Set $\alpha := \alpha/2$, and go to S1.

The algorithm consists of $O(\log K_\infty)$ scaling phases. This approach has been particularly successful for resource allocation problems [6, 7, 8, 13] and for convex network flow problems (under the name of "capacity scaling") [1, 11, 12]. Different types of proximity theorems have also been investigated: proximity between integral and real optimal solutions, among others.

In discrete convex analysis [15], a variety of discrete convex functions are considered. A function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ is called *integrally convex* if its local convex extension $\tilde{f} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is (globally) convex in the ordinary sense, where $\tilde{f}$ is defined as the collection of convex extensions of f in each unit hypercube $[\boldsymbol{a}, \boldsymbol{a} + \mathbf{1}]_{\mathbb{R}}$ with $\boldsymbol{a} \in \mathbb{Z}^n$; see Section 2 for precise statements.

For a function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$, $\mathrm{dom}\, f = \{x \in \mathbb{Z}^n \mid f(x) < +\infty\}$ is called the effective domain of f. Discrete midpoint convexity of f for $x, y \in \mathbb{Z}^n$ means

$$f(x) + f(y) \geq f\left(\left\lceil \frac{x+y}{2} \right\rceil\right) + f\left(\left\lfloor \frac{x+y}{2} \right\rfloor\right), \tag{1.1}$$

where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the integer vectors obtained by componentwise rounding-up and rounding-down to the nearest integers, respectively. For $x, y \in \mathbb{Z}^n$, $x \vee y$ and $x \wedge y$ denote the vectors of componentwise maximum and minimum of x and y, respectively.

A function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ is called *L$^\natural$-convex* if it satisfies one of the equivalent conditions (a) to (d) below:

(a) f is integrally convex and submodular: $f(x) + f(y) \geq f(x \vee y) + f(x \wedge y)$ $(x, y \in \mathbb{Z}^n)$.
(b) f satisfies discrete midpoint convexity (1.1) for all $x, y \in \mathbb{Z}^n$.
(c) f satisfies discrete midpoint convexity (1.1) for all $x, y \in \mathbb{Z}^n$ with $\|x - y\|_\infty \leq 2$, and the effective domain has the property: $x, y \in \mathrm{dom}\, f \Rightarrow \lceil (x+y)/2 \rceil, \lfloor (x+y)/2 \rfloor \in \mathrm{dom}\, f$.
(d) f satisfies translation-submodularity: $f(x) + f(y) \geq f((x - \mu\mathbf{1}) \vee y) + f(x \wedge (y + \mu\mathbf{1}))$ $(\mu \in \mathbb{Z}_+,\ x, y \in \mathbb{Z}^n)$, where $\mathbf{1} = (1, 1, \ldots, 1)$.

A function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ is called *M$^\natural$-convex* if it has the exchange property: For any $x, y \in \mathrm{dom}\, f$ and any $i \in \mathrm{supp}^+(x - y)$, there exists $j \in \mathrm{supp}^-(x - y) \cup \{0\}$ such that $f(x) + f(y) \geq f(x - \mathbf{1}_i + \mathbf{1}_j) + f(y + \mathbf{1}_i - \mathbf{1}_j)$, where $\mathrm{supp}^+(z) = \{i \mid z_i > 0\}$ and $\mathrm{supp}^-(z) = \{j \mid z_j < 0\}$ for $z \in \mathbb{Z}^n$, $\mathbf{1}_i$ denotes the i-th unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ if $1 \leq i \leq n$, and $\mathbf{1}_i = \mathbf{0}$ if $i = 0$.

Integrally convex functions constitute a common framework for discrete convex functions, including separable convex, L$^\natural$-convex and M$^\natural$-convex functions as well as L$^\natural_2$-convex and M$^\natural_2$-convex functions [15], and BS-convex and UJ-convex functions [3]. The concept of integral convexity is used in formulating discrete fixed point theorems [9, 19], and designing solution algorithms for discrete systems of nonlinear equations [17, 18]. In game theory the integral concavity of payoff functions guarantees the existence of a pure strategy equilibrium in finite symmetric games [10].

The scaling operation preserves L$^\natural$-convexity, that is, if f is L$^\natural$-convex, then f^α is L$^\natural$-convex. M$^\natural$-convexity is subtle in this respect: for an M$^\natural$-convex function f, f^α remains M$^\natural$-convex if $n \leq 2$, while this is not always the case if $n \geq 3$. However, nothing is known about scaling of integrally convex functions.

As for proximity theorems, the following facts are known for L$^\natural$-convex and M$^\natural$-convex functions.

▶ **Theorem 1.1** ([12, 14, 15]). *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$, $\alpha \in \mathbb{Z}_{++}$ (positive integer), and $x^\alpha \in \mathrm{dom}\, f$.*
(1) *Suppose that f is an* L$^\natural$*-convex function. If $f(x^\alpha) \leq f(x^\alpha + \alpha d)$ for all $d \in \{0,1\}^n \cup \{0,-1\}^n$, then there exists a minimizer x^* of f with $\|x^\alpha - x^*\|_\infty \leq n(\alpha - 1)$.*
(2) *Suppose that f is an* M$^\natural$*-convex function. If $f(x^\alpha) \leq f(x^\alpha + \alpha d)$ for all $d \in \{\mathbf{1}_i, -\mathbf{1}_i\ (1 \leq i \leq n),\ \mathbf{1}_i - \mathbf{1}_j\ (i \neq j)\}$, then there exists a minimizer x^* of f with $\|x^\alpha - x^*\|_\infty \leq n(\alpha - 1)$.*

Based on the above results, efficient algorithms for minimizing L$^\natural$-convex and M$^\natural$-convex functions have been successfully designed with the proximity-scaling approach (see [15]). Proximity theorems are also available for L$_2^\natural$-convex and M$_2^\natural$-convex functions [16] and L-convex functions on graphs [5]. However, no proximity theorem is proved for integrally convex functions.

The following are the new findings of this paper about integrally convex functions:

- A "box-barrier property" (Theorem 2.3), which allows us to restrict the search for a global minimum.
- Integral convexity is preserved under scaling if $n = 2$ (Theorem 3.1), but not when $n \geq 3$ (Example 3.3).
- A proximity theorem with an exponential bound $[(n+1)!/2^{n-1}](\alpha - 1)$ holds for all n (Theorem 4.3), but does not hold with the smaller bound $n(\alpha - 1)$ when $n \geq 3$ (Examples 4.1, 4.2).

Thus, to extend the known proximity and scaling results for L$^\natural$-convex functions to a wider class of functions, a novel concept of "directed integrally convex functions" is defined. For this new class of functions the following properties hold:

- The new class coincides with the class of integrally convex functions for $n \leq 2$, and is a proper subclass of this for $n \geq 3$ (Proposition 5.1 (1)).
- The new class is a proper superclass of L$^\natural$-convex functions for all $n \geq 2$ (Proposition 5.1 (2)).
- Directed integral convexity is preserved under scaling for all n (Theorem 5.6).
- A proximity theorem with bound $n(\alpha - 1)$ holds for all n (Theorem 5.7).

As a consequence of our proximity and scaling results, we derive that:

- When n is fixed, a (directed) integrally convex function can be minimized in $O(\log K_\infty)$ time by standard proximity-scaling algorithms, where K_∞ denotes the ℓ_∞-size of $\mathrm{dom}\, f$.

2 Integrally Convex Functions

For $x \in \mathbb{R}^n$ the integer neighborhood of x is defined as $N(x) = \{z \in \mathbb{Z}^n \mid |x_i - z_i| < 1\ (i = 1, \ldots, n)\}$. For a function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ the local convex extension $\tilde{f} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ of f is defined as the union of all convex envelopes of f on $N(x)$ as follows:

$$\tilde{f}(x) = \min\{\sum_{y \in N(x)} \lambda_y f(y) \mid \sum_{y \in N(x)} \lambda_y y = x,\ \sum_{y \in N(x)} \lambda_y = 1, \lambda_y \geq 0\ (\forall y \in N(x))\} \quad (x \in \mathbb{R}^n). \tag{2.1}$$

If $\tilde{f}$ is convex on $\mathbb{R}^n$, then f is said to be integrally convex. A set $S \subseteq \mathbb{Z}^n$ is said to be integrally convex if, for any $x \in \mathbb{R}^n$, $x \in \overline{S}$ implies $x \in \overline{S \cap N(x)}$, i.e., if the convex hull of S coincides with the union of the convex hulls of $S \cap N(x)$ for $x \in \mathbb{R}^n$.

Integral convexity can be characterized by a local condition. The following theorem is proved in [2] when the effective domain is an integer interval (discrete rectangle).

▶ **Theorem 2.1** ([2, Proposition 3.3]). *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be a function with an integrally convex effective domain. Then the following properties,* (a) *and* (b)*, are equivalent:* (a) *f is integrally convex.* (b) *For every $x, y \in \operatorname{dom} f$ with $\|x-y\|_\infty = 2$ we have*

$$\tilde{f}\left(\frac{x+y}{2}\right) \leq \frac{f(x)+f(y)}{2}. \tag{2.2}$$

▶ **Theorem 2.2** ([2, Proposition 3.1]; see also [15, Theorem 3.21]). *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be an integrally convex function and $x^* \in \operatorname{dom} f$. Then x^* is a minimizer of f if and only if $f(x^*) \leq f(x^*+d)$ for all $d \in \{-1, 0, +1\}^n$.*

The local characterization of global minima stated in Theorem 2.2 can be generalized to the following form, which we use in Section 5.4.

▶ **Theorem 2.3** (Box-barrier property). *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be an integrally convex function, and let $p \in (\mathbb{Z} \cup \{+\infty\})^n$ and $q \in (\mathbb{Z} \cup \{-\infty\})^n$, where $q \leq p$. Let $S = \{x \in \mathbb{Z}^n \mid q_i < x < p_i \ (\forall i)\}$, $W_i^+ = \{x \in \mathbb{Z}^n \mid x_i = p_i,\ q_j \leq x_j \leq p_j \ (j \neq i)\}$, $W_i^- = \{x \in \mathbb{Z}^n \mid x_i = q_i,\ q_j \leq x_j \leq p_j \ (j \neq i)\}$ $(i = 1, \ldots, n)$, $W = \bigcup_{i=1}^n (W_i^+ \cup W_i^-)$, and $\hat{x} \in S \cap \operatorname{dom} f$. If $f(\hat{x}) \leq f(y)$ for all $y \in W$, then $f(\hat{x}) \leq f(z)$ for all $z \in \mathbb{Z}^n \setminus S$.*

Proof. Let $U = \bigcup_{i=1}^n \{x \in \mathbb{R}^n \mid x_i \in \{p_i, q_i\},\ q_j \leq x_j \leq p_j \ (j \neq i)\}$, for which we have $U \cap \mathbb{Z}^n = W$. For $z \in \mathbb{Z}^n \setminus S$, the line segment connecting $\hat{x}$ and z intersects U at a point, say, $u \in \mathbb{R}^n$. Then $N(u)$ is contained in W. Since the local convex extension $\tilde{f}(u)$ is a convex combination of $f(y)$'s with $y \in N(u)$ and $f(y) \geq f(\hat{x})$ for every $y \in W$, we have $\tilde{f}(u) \geq f(\hat{x})$. On the other hand, it follows from integral convexity that $\tilde{f}(u) \leq (1-\lambda) f(\hat{x}) + \lambda f(z)$ for some λ with $0 < \lambda \leq 1$. Hence $f(\hat{x}) \leq \tilde{f}(u) \leq (1-\lambda) f(\hat{x}) + \lambda f(z)$, and therefore, $f(\hat{x}) \leq f(z)$. ◀

3 Scaling Operation for Integrally Convex Functions

For $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ and $\alpha \in \mathbb{Z}_{++}$, the α-scaling of f is the function $f^\alpha : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ defined by $f^\alpha(x) = f(\alpha x)$ $(x \in \mathbb{Z}^n)$. When $n = 2$, integral convexity is preserved under scaling.

▶ **Theorem 3.1.** *Let $f : \mathbb{Z}^2 \to \mathbb{R} \cup \{+\infty\}$ be an integrally convex function and $\alpha \in \mathbb{Z}_{++}$. Then the scaled function f^α is integrally convex.*

Proof. First note that a set $S \subseteq \mathbb{Z}^2$ is an integrally convex set if and only if it can be represented as $S = \{(x_1, x_2) \in \mathbb{Z}^2 \mid a_i x_1 + b_i x_2 \leq c_i \ (i = 1, \ldots, m)\}$ for some $a_i, b_i \in \{-1, 0, 1\}$ and $c_i \in \mathbb{Z}$ $(i = 1, \ldots, m)$. Hence,

$\operatorname{dom} f^\alpha = (\operatorname{dom} f \cap (\alpha\mathbb{Z})^2)/\alpha$ is an integrally convex set. By Theorem 2.1 we only have to check condition (2.2) for f^α with $x = (0, 0)$ and $y = (2, 0)$, $(2, 1)$, $(2, 2)$, i.e.,

$$f(0,0) + f(2\alpha, 0) \geq 2f(\alpha, 0),$$
$$f(0,0) + f(2\alpha, 2\alpha) \geq 2f(\alpha, \alpha),$$
$$f(0,0) + f(2\alpha, \alpha) \geq f(\alpha, \alpha) + f(\alpha, 0).$$

The first two inequalities follow easily from integral convexity of f, whereas the third inequality is a special case of "basic parallelogram inequality" (3.1) below with $a = b = \alpha$. ◀

▶ **Proposition 3.2.** *For an integrally convex function $f : \mathbb{Z}^2 \to \mathbb{R} \cup \{+\infty\}$ we have*

$$f(0,0) + f(a+b,a) \geq f(a,a) + f(b,0) \qquad (a,b \in \mathbb{Z}_+). \tag{3.1}$$

Proof. We may assume $a, b \geq 1$ and $\{(0,0),(a+b,a)\} \subseteq \operatorname{dom} f$, which implies $k(1,1) + l(1,0) \in \operatorname{dom} f$ for all (k,l) with $0 \leq k \leq a$, $0 \leq l \leq b$. We use notation $f_x(z) = f(x+z)$. For each $x \in \operatorname{dom} f$ we have $f_x(0,0) + f_x(2,1) \geq f_x(1,1) + f_x(1,0)$ by integral convexity of f. By adding these inequalities for $x = k(1,1) + l(1,0)$ with $0 \leq k \leq a-1$ and $0 \leq l \leq b-1$, we obtain (3.1). Note that all terms involved in these inequalities are finite. ◀

If $n \geq 3$, f^α is not always integrally convex, as is demonstrated by the following example.

▶ **Example 3.3.** Consider the integrally convex function $f : \mathbb{Z}^3 \to \mathbb{R} \cup \{+\infty\}$ defined on $\operatorname{dom} f = [(0,0,0),(4,2,2)]_{\mathbb{Z}}$ by

x_2	$f(x_1,x_2,0)$					
2	3	1	1	1	3	
1	1	0	0	0	0	
0	0	0	0	0	3	
	0	1	2	3	4	x_1

x_2	$f(x_1,x_2,1)$					
2	2	1	0	0	0	
1	1	0	0	0	0	
0	0	0	0	0	0	
	0	1	2	3	4	x_1

x_2	$f(x_1,x_2,2)$					
2	3	2	1	0	0	
1	2	1	0	0	0	
0	3	0	0	0	3	
	0	1	2	3	4	x_1

For the scaling with $\alpha = 2$, we have a failure of integral convexity: $f(0,0,0) + f(4,2,2) < \min[f(2,2,2) + f(2,0,0), f(2,2,0) + f(2,0,2)]$. The set $S = \operatorname{argmin} f = \{x \mid f(x) = 0\}$ is an integrally convex set, and $S^\alpha = \{x \mid \alpha x \in S\} = \{(0,0,0),(1,0,0),(1,0,1),(2,1,1)\}$ is not. ◀

Seeing that the class of L$^\natural$-convex functions is stable under scaling, while this is not true for the superclass of integrally convex functions, naturally leads to the question of finding an intermediate class of functions that is stable under scaling. An answer to this question will be given in Section 5.3.

4 Proximity Results for Integrally Convex Functions

In this section we show that a proximity theorem holds for integrally convex functions. More precisely, we show that if x^α is an α-local minimizer of an integrally convex function f, i.e., $f(x^\alpha) \leq f(x^\alpha + \alpha d)$ for all $d \in \{-1,0,+1\}^n$, then there exists a global minimizer x^* of f for which the ℓ_∞-distance from x^α is bounded by an appropriate function $B(n,\alpha)$. However, we first show that the bounding function $B(n,\alpha)$ must be at least quadratic in n, so that $B(n,\alpha) = n(\alpha - 1)$, which applies to L$^\natural$-convex functions, is not valid in this case.

4.1 Lower bounds for proximity distance

For integrally convex functions with $n \geq 3$, the bound $n(\alpha - 1)$ is not valid.

▶ **Example 4.1.** Consider the integrally convex function $f : \mathbb{Z}^3 \to \mathbb{R} \cup \{+\infty\}$ defined on $\operatorname{dom} f = [(0,0,0),(4,2,2)]_{\mathbb{Z}}$ by

x_2	$f(x_1,x_2,0)$					
2	5	1	0	0	4	
1	2	-1	-2	0	3	
0	0	-1	0	1	6	
	0	1	2	3	4	x_1

x_2	$f(x_1,x_2,1)$					
2	4	1	-2	-3	-1	
1	2	-1	-2	-3	-1	
0	2	-1	-2	0	5	
	0	1	2	3	4	x_1

x_2	$f(x_1,x_2,2)$					
2	6	3	0	-3	-4	
1	6	1	-2	-3	1	
0	6	2	0	3	6	
	0	1	2	3	4	x_1

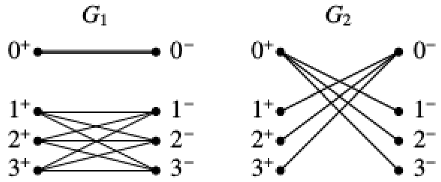


Figure 1 Example for $O(n^2)$ lower bound for proximity distance ($m = 3$).

and let $\alpha = 2$. For $x^\alpha = (0,0,0)$ we have $f(x^\alpha) = 0$, $f(x^\alpha) \leq f(x^\alpha + 2d)$ for $d = (1,0,0),(0,1,0), (0,0,1), (1,1,0),(1,0,1),(0,1,1),(1,1,1)$. Hence $x^\alpha = (0,0,0)$ is α-local minimal. A unique minimizer of f is located at $x^* = (4,2,2)$ with $f(x^*) = -4$ and $\|x^\alpha - x^*\|_\infty = 4$. The ℓ_∞-distance between x^α and x^* is strictly larger than $n(\alpha - 1) = 3$. ◀

▶ **Example 4.2.** For a positive integer $m \geq 1$, we consider two bipartite graphs G_1 and G_2 on vertex bipartition $(\{0^+,1^+,\ldots,m^+\},\{0^-,1^-,\ldots,m^-\})$; see Fig. 1. The edge sets of G_1 and G_2 are defined respectively as $E_1 = \{(0^+,0^-)\} \cup \{(i^+,j^-) \mid i,j = 1,\ldots,m\}$ and $E_2 = \{(0^+,j^-) \mid j = 1,\ldots,m\} \cup \{(i^+,0^-) \mid i = 1,\ldots,m\}$. Let $V^+ = \{1^+,\ldots,m^+\}$, $V^- = \{1^-,\ldots,m^-\}$, and $n = 2m+2$. Consider $X_1, X_2 \subseteq \mathbb{Z}^n$ defined by

$$X_1 = \left\{ \sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_{ij}(\mathbf{1}_{i^+} - \mathbf{1}_{j^-}) + \lambda_0(\mathbf{1}_{0^+} - \mathbf{1}_{0^-}) \;\middle|\; \begin{array}{l} \lambda_{ij} \in [0,\alpha-1]_{\mathbb{Z}}\ (i,j = 1,\ldots,m) \\ \lambda_0 \in [0,m^2(\alpha-1)]_{\mathbb{Z}} \end{array} \right\},$$

$$X_2 = \left\{ \sum_{i=1}^{m} \mu_i(\mathbf{1}_{i^+} - \mathbf{1}_{0^-}) + \sum_{j=1}^{m} \nu_j(\mathbf{1}_{0^+} - \mathbf{1}_{j^-}) \;\middle|\; \begin{array}{l} \mu_i \in [0,m(\alpha-1)]_{\mathbb{Z}}\ (i = 1,\ldots,m) \\ \nu_j \in [0,m(\alpha-1)]_{\mathbb{Z}}\ (j = 1,\ldots,m) \end{array} \right\},$$

where X_1 and X_2 represent the sets of boundaries of flows in G_1 and G_2, respectively. We define functions $f_1, f_2 : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ with $\operatorname{dom} f_1 = X_1$ and $\operatorname{dom} f_2 = X_2$ by

$$f_1(x) = \begin{cases} x(V^-) & (x \in X_1) \\ +\infty & (x \notin X_1), \end{cases} \qquad f_2(x) = \begin{cases} x(V^-) & (x \in X_2) \\ +\infty & (x \notin X_2) \end{cases} \qquad (x \in \mathbb{Z}^n),$$

where $x(U) = \sum_{u \in U} x_u$ for any set U of vertices. Both f_1 and f_2 are M-convex, and hence $f = f_1 + f_2$ is an M$_2$-convex function, which is integrally convex (see [15, Section 8.3.1]). We have $\operatorname{dom} f = X_1 \cap X_2$ and f is linear on $\operatorname{dom} f$. As is easily verified, f has a unique minimizer at x^* defined by

$$\begin{aligned} x_u^* &= m(\alpha-1)\ (u \in V^+), \\ &= -\,m(\alpha-1)\ (u \in V^-), \\ &= m^2(\alpha-1)\ (u = 0^+), \\ &= -\,m^2(\alpha-1)\ (u = 0^-), \end{aligned}$$

which corresponds to $\lambda_0 = m^2(\alpha-1),\ \ \lambda_{ij} = \alpha - 1,\ \ \mu_i = \nu_j = m(\alpha-1)\ \ (i,j = 1,\ldots,m)$.

Let $x^\alpha = \mathbf{0}$. This is α-local minimal, since $\operatorname{dom} f \cap \{-\alpha,0,\alpha\}^n = \{\mathbf{0}\}$, which can be verified easily. With $\|x^* - x^\alpha\|_\infty = m^2(\alpha-1) = (n-2)^2(\alpha-1)/4$, this example demonstrates a quadratic lower bound $(n-2)^2(\alpha-1)/4$ for the proximity distance for integrally convex functions. ◀

We have seen that the proximity theorem with linear bound $B(n,\alpha) = n(\alpha-1)$, which is valid for L$^\natural$-convex functions, does not hold for all integrally convex functions. Thus, we may ask if there is a subclass of integrally convex functions, including L$^\natural$-convex functions, that admits such a linear proximity bound. An answer to this question will be given in Section 5.4. Another question is whether we can establish a proximity theorem at all by enlarging the proximity bound. This question is answered next.

4.2 Theorem

▶ **Theorem 4.3.** *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be an integrally convex function, $\alpha \in \mathbb{Z}_{++}$, and $x^\alpha \in \mathrm{dom}\, f$.*

(1) *If $f(x^\alpha) \leq f(x^\alpha + \alpha d)$ $(\forall\ d \in \{-1, 0, +1\}^n)$, then $\mathrm{argmin}\, f \neq \emptyset$ and there exists $x^* \in \mathrm{argmin}\, f$ with $\|x^\alpha - x^*\|_\infty \leq \beta_n(\alpha - 1)$, where $\beta_1 = 1$, $\beta_2 = 2$; $\beta_n = \dfrac{n+1}{2}\beta_{n-1} + 1$ $(n = 3, 4, \ldots)$.*

(2) $\beta_n \leq \dfrac{(n+1)!}{2^{n-1}}$ $(n = 3, 4, \ldots)$.

To prove Theorem 4.3(1) we first note that it follows from its special case where $x^\alpha = \mathbf{0}$ and f is defined on a bounded set in the nonnegative orthant $\mathbb{Z}^n_+$. That is, the proof of Theorem 4.3(1) is reduced to proving the following proposition. We use notation $V = \{1, 2, \ldots, n\}$ and the characteristic vector of $A \subseteq V$ is denoted by $\mathbf{1}_A$.

▶ **Proposition 4.4.** *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be an integrally convex function such that $\mathrm{dom}\, f$ is a bounded subset of $\mathbb{Z}^n_+$ containing the origin $\mathbf{0}$. If*

$$f(\mathbf{0}) \leq f(\alpha \mathbf{1}_A) \qquad (\forall A \subseteq V), \tag{4.1}$$

then there exists $x^* \in \mathrm{argmin}\, f$ with $\|x^*\|_\infty \leq \beta_n(\alpha - 1)$.

4.3 Tools for the proof: f-minimality

In this section we introduce some technical tools that we use in the proof of Proposition 4.4.

For two nonnegative integer vectors $x, y \in \mathbb{Z}^n_+$, we write $y \preceq_f x$ if $y \leq x$ and $f(y) \leq f(x)$. Note that $y \preceq_f x$ if and only if $(y, f(y)) \leq (x, f(x))$ in $\mathbb{R}^{n+1}$. We say that $x \in \mathbb{Z}^n_+$ is *f-minimal* if there exists no $y \in \mathbb{Z}^n_+$ such that $y \preceq_f x$ and $y \neq x$. That is, x is f-minimal if and only if it is the unique minimizer of the function f restricted to the integer interval $[\mathbf{0}, x]_\mathbb{Z}$.

▶ **Lemma 4.5.** *Assume (4.1). For any $A\ (\neq \emptyset) \subseteq V$ and $\lambda \in \mathbb{Z}_+$ we have $(\alpha - 1)\mathbf{1}_A \preceq_f (\alpha - 1)\mathbf{1}_A + \lambda \mathbf{1}_A$.*

Proof. First note that $(\alpha - 1)\mathbf{1}_A \leq (\alpha - 1)\mathbf{1}_A + \lambda\mathbf{1}_A$ for all $\lambda \in \mathbb{Z}_+$. By integral convexity of f, $g(t) = f(t\mathbf{1}_A)$ is convex in $t \in \mathbb{Z}_+$, and therefore, $g(\alpha - 1) \leq [(\alpha - 1)g(\alpha) + g(0)]/\alpha$. On the other hand, $g(0) \leq g(\alpha)$ by α-local minimality (4.1). Hence $g(\alpha - 1) \leq g(\alpha)$. By convexity of g, this implies $g(\alpha - 1) \leq g((\alpha - 1) + \lambda)$ for all $\lambda \in \mathbb{Z}_+$, i.e., $f((\alpha - 1)\mathbf{1}_A) \leq f((\alpha - 1)\mathbf{1}_A + \lambda\mathbf{1}_A)$ for all $\lambda \in \mathbb{Z}_+$. ◀

▶ **Lemma 4.6.** *Let $x \in \mathbb{Z}^n_+$ and $A\ (\neq \emptyset) \subseteq V$, and assume $x \preceq_f x + \mathbf{1}_A$.*

(1) *For any $i \in V$ and $\lambda \in \mathbb{Z}_+$ we have $x + \mathbf{1}_A + \mathbf{1}_i \preceq_f (x + \mathbf{1}_A + \mathbf{1}_i) + \lambda\mathbf{1}_A$.*

(2) *For any $i \in A$ and $\lambda \in \mathbb{Z}_+$ we have $x + \mathbf{1}_A - \mathbf{1}_i \preceq_f (x + \mathbf{1}_A - \mathbf{1}_i) + \lambda\mathbf{1}_A$.*

Proof. (1) First note that $x + \mathbf{1}_A + \mathbf{1}_i \leq (x + \mathbf{1}_A + \mathbf{1}_i) + \lambda\mathbf{1}_A$ for all $\lambda \in \mathbb{Z}_+$. Define $g(\lambda) = f((x + \mathbf{1}_A + \mathbf{1}_i) + \lambda\mathbf{1}_A)$, which is a convex function in $\lambda \in \mathbb{Z}_+$ by integral convexity of f. We are to show $g(0) \leq g(\lambda)$ for all $\lambda \in \mathbb{Z}_+$, which is equivalent to $g(0) \leq g(1)$ by convexity of g. By integral convexity of f we have $f(x+2\mathbf{1}_A+\mathbf{1}_i)+f(x) \geq f(x+\mathbf{1}_A)+f(x+\mathbf{1}_A+\mathbf{1}_i)$, whereas $f(x) \leq f(x+\mathbf{1}_A)$ by the assumption. Hence $f(x+\mathbf{1}_A+\mathbf{1}_i) \leq f(x+2\mathbf{1}_A+\mathbf{1}_i)$, i.e., $g(0) \leq g(1)$. (2) Similarly. ◀

Repeated application of (1) and (2) of Lemma 4.6 yields the following general form.

▶ **Lemma 4.7.** *Let $x \in \mathbb{Z}^n_+$ and $A\ (\neq \emptyset) \subseteq V$, and assume $x \preceq_f x+\mathbf{1}_A$. For any $\lambda \in \mathbb{Z}_+$, $\mu_i^+, \mu_i^- \in \mathbb{Z}_+$ $(i \in A)$ and $\mu_i^\circ \in \mathbb{Z}_+$ $(i \notin A)$, we have*

$$\begin{aligned} & x+\sum_{i\in A}\mu_i^+(\mathbf{1}_A+\mathbf{1}_i)+\sum_{i\in A}\mu_i^-(\mathbf{1}_A-\mathbf{1}_i)+\sum_{i\notin A}\mu_i^\circ(\mathbf{1}_A+\mathbf{1}_i) \\ \preceq_f\ & x+\sum_{i\in A}\mu_i^+(\mathbf{1}_A+\mathbf{1}_i)+\sum_{i\in A}\mu_i^-(\mathbf{1}_A-\mathbf{1}_i)+\sum_{i\notin A}\mu_i^\circ(\mathbf{1}_A+\mathbf{1}_i)+\lambda\mathbf{1}_A. \end{aligned} \tag{4.2}$$

For $A\ (\neq \emptyset) \subseteq V$, let B_A denote the set of the generating vectors in (4.2) and C_A the set of their nonnegative integer combinations[1]:

$$B_A = \{\mathbf{1}_A\} \cup \bigcup_{i\in A}\{\mathbf{1}_A+\mathbf{1}_i, \mathbf{1}_A-\mathbf{1}_i\} \cup \bigcup_{i\notin A}\{\mathbf{1}_A+\mathbf{1}_i\}, \tag{4.3}$$

$$C_A = \{\lambda\mathbf{1}_A+\sum_{i\in A}\mu_i^+(\mathbf{1}_A+\mathbf{1}_i)+\sum_{i\in A}\mu_i^-(\mathbf{1}_A-\mathbf{1}_i)+\sum_{i\notin A}\mu_i^\circ(\mathbf{1}_A+\mathbf{1}_i) \mid \lambda, \mu_i^+, \mu_i^-, \mu_i^\circ \in \mathbb{Z}_+\}. \tag{4.4}$$

▶ **Lemma 4.8.** *Assume (4.1). If $y \in \mathbb{Z}^n_+$ is f-minimal, then $y \notin \alpha\mathbf{1}_A + C_A$ for any $A(\neq \emptyset) \subseteq V$.*

Proof. To prove the contraposition, suppose that $y \in \alpha\mathbf{1}_A + C_A$ for some A. Then

$$y = \alpha\mathbf{1}_A + \left(\mu\mathbf{1}_A+\sum_{i\in A}\mu_i^+(\mathbf{1}_A+\mathbf{1}_i)+\sum_{i\in A}\mu_i^-(\mathbf{1}_A-\mathbf{1}_i)+\sum_{i\notin A}\mu_i^\circ(\mathbf{1}_A+\mathbf{1}_i)\right)$$

for some $\mu, \mu_i^+, \mu_i^-, \mu_i^\circ \in \mathbb{Z}_+$. Or equivalently,

$$y = (\alpha-1)\mathbf{1}_A+\sum_{i\in A}\mu_i^+(\mathbf{1}_A+\mathbf{1}_i)+\sum_{i\in A}\mu_i^-(\mathbf{1}_A-\mathbf{1}_i)+\sum_{i\notin A}\mu_i^\circ(\mathbf{1}_A+\mathbf{1}_i)+(\mu+1)\mathbf{1}_A,$$

which is of the form of (4.2) with $x = (\alpha-1)\mathbf{1}_A$ and $\lambda = \mu+1$. Since $x = (\alpha-1)\mathbf{1}_A \preceq_f \alpha\mathbf{1}_A = x+\mathbf{1}_A$ by Lemma 4.5, Lemma 4.7 shows that y is not f-minimal. ◀

4.4 Proof of Proposition 4.4 for $n = 2$

We prove Proposition 4.4 for $n = 2$. Take $x^* = (x_1^*, x_2^*) \in \arg\min f$ that is f-minimal. We may assume $x_1^* \geq x_2^*$. Since x^* is f-minimal, Lemma 4.8 shows that x^* belongs to $X^* = \{(x_1, x_2) \in \mathbb{Z}^2_+ \mid x_1 \geq x_2\} \setminus ((\alpha\mathbf{1}_A + C_A) \cup (\alpha\mathbf{1}_V + C_V))$, where $A = \{1\}$ and $V = \{1, 2\}$. On noting $C_A = \{\mu_1(1,0) + \mu_{12}(1,1) \mid \mu_1, \mu_{12} \in \mathbb{Z}_+\}$, $C_V = \{\mu_1(1,0) + \mu_2(0,1) \mid \mu_1, \mu_2 \in \mathbb{Z}_+\}$, we see that X^* consists of all integer points contained in the parallelogram with vertices $(0,0)$, $(\alpha-1, 0)$, $(2\alpha-2, \alpha-1)$, $(\alpha-1, \alpha-1)$. Therefore, $\|x^*\|_\infty \leq 2(\alpha-1)$. Thus Proposition 4.4 for $n = 2$ is proved.

[1] It can be shown that B_A is a Hilbert basis of the convex cone generated by B_A.

4.5 Proof of Proposition 4.4 for $n \geq 3$

In this section we prove Proposition 4.4 for $n \geq 3$ by induction on n. Accordingly we assume that Proposition 4.4 is true for every integrally convex function in $n-1$ variables.

Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be an integrally convex function such that $\mathrm{dom}\, f$ is a bounded subset of $\mathbb{Z}^n_+$ containing the origin $\mathbf{0}$. Take $x^* = (x^*_1, x^*_2, \ldots, x^*_n) \in \arg\min f$ that is f-minimal. Then

$$[\mathbf{0}, x^*]_{\mathbb{Z}} \cap \arg\min f = \{x^*\}. \tag{4.5}$$

We may assume $x^*_1 \geq x^*_2 \geq \cdots \geq x^*_n$. For any $x \in \mathbb{Z}^n_+$ let $f_{[\mathbf{0},x]} : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ denote the restriction of f to the interval $[\mathbf{0}, x]_{\mathbb{Z}}$, that is, $f_{[\mathbf{0},x]}(y) = f(y)$ if $y \in [\mathbf{0}, x]_{\mathbb{Z}}$, and $= +\infty$ otherwise.

The following lemma reveals a key fact that will be used for induction on n. Note that, by (4.5), x^* satisfies the condition imposed on $x^\bullet$.

▶ **Lemma 4.9.** *Let $x^\bullet \in \mathrm{dom}\, f$ and assume $\arg\min f_{[\mathbf{0},x^\bullet]} = \{x^\bullet\}$. Then for any $i \in V$ there exists $x^\circ \in \mathrm{dom}\, f$ such that*

$$\mathbf{0} \leq x^\circ \leq x^\bullet, \quad \|x^\circ - x^\bullet\|_\infty = 1, \quad x^\circ_i = x^\bullet_i - 1, \quad \arg\min f_{[\mathbf{0},x^\circ]} = \{x^\circ\}.$$

Proof. Let x° be a minimizer of $f(x)$ among those x which satisfy the conditions: $\mathbf{0} \leq x \leq x^\bullet$, $\|x - x^\bullet\|_\infty = 1$, and $x_i = x^\bullet_i - 1$; in case of multiple minimizers, we choose a minimal minimizer. Then we can show $\arg\min f_{[\mathbf{0},x^\circ]} = \{x^\circ\}$. ◀

Lemma 4.9 can be applied repeatedly, since the resulting point x° satisfies the condition imposed on the initial point $x^\bullet$. Starting with $x^\bullet = x^*$ we apply Lemma 4.9 repeatedly with $i = n$. After x^*_n applications, we arrive at a point $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{n-1}, 0)$. We have $\arg\min f_{[\mathbf{0},\hat{x}]} = \{\hat{x}\}$ and

$$x^*_j - x^*_n \leq \hat{x}_j \qquad (j = 1, 2, \ldots, n-1). \tag{4.6}$$

We now consider a function $\hat{f} : \mathbb{Z}^{n-1} \to \mathbb{R} \cup \{+\infty\}$ defined by

$$\hat{f}(x_1, x_2, \ldots, x_{n-1}) = \begin{cases} f(x_1, x_2, \ldots, x_{n-1}, 0) & (0 \leq x_j \leq \hat{x}_j \ (j = 1, 2, \ldots, n-1)), \\ +\infty & (\text{otherwise}). \end{cases}$$

This function $\hat{f}$ is an integrally convex function in $n-1$ variables, and the origin $\mathbf{0}$ is α-local minimal for $\hat{f}$ and $\hat{x}$ is the unique minimizer of $\hat{f}$. By the induction hypothesis, we can apply Proposition 4.4 to $\hat{f}$ to obtain $\|\hat{x}\|_\infty \leq \beta_{n-1}(\alpha - 1)$. Combining this with (4.6) we obtain

$$x^*_1 - x^*_n \leq \beta_{n-1}(\alpha - 1). \tag{4.7}$$

We can also show

$$x^*_n \leq \frac{n-1}{n+1} x^*_1 + \frac{2(\alpha - 1)}{n+1} \tag{4.8}$$

from the f-minimality of x^*. It follows from (4.7) and (4.8) that

$$x^*_1 \leq \left(\frac{n+1}{2} \beta_{n-1} + 1 \right) (\alpha - 1) = \beta_n (\alpha - 1).$$

This completes the proof of Proposition 4.4, and hence that of Theorem 4.3 (1). The bound for β_n in Theorem 4.3 (2) is a simple calculus from the recurrence.

5 Directed Integrally Convex Functions

In this section we introduce a novel class of integrally convex functions, which admits the scaling operation and the proximity theorem with the linear bound $n(\alpha - 1)$.

5.1 Definition

We call a function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ *directed integrally convex* if $\mathrm{dom}\, f$ is an integer interval and discrete midpoint convexity (1.1) is satisfied by every pair $(x, y) \in \mathbb{Z}^n \times \mathbb{Z}^n$ with $\|x - y\|_\infty = 2$ (exactly equal to two).

▶ **Proposition 5.1.**

(1) *A directed integrally convex function is integrally convex.*

(2) *An* L$^\natural$*-convex function (defined on an integer interval) is directed integrally convex.*

Proof. (1) Let f be a directed integrally convex function. To use Theorem 2.1, take integer points $x, y \in \mathrm{dom}\, f$ with $\|x - y\|_\infty = 2$. For $u = (x + y)/2$, both $\lceil u \rceil$ and $\lfloor u \rfloor$ belong to $N(u)$, and therefore $2\tilde{f}(u) \leq f(\lceil u \rceil) + f(\lfloor u \rfloor) \leq f(x) + f(y)$, where the second inequality is midpoint convexity.

(2) By the characterizations of L$^\natural$-convex functions in Section 1. ◀

5.2 Parallelogram inequality

For directed integrally convex functions two special direction vectors play a crucial role:

$$d_1 = (\mathbf{1}^{m_1}, \mathbf{1}^{m_2}, -\mathbf{1}^{m_3}, \mathbf{0}^{m_4}, \mathbf{0}^{m_5}), \quad d_2 = (\mathbf{1}^{m_1}, \mathbf{0}^{m_2}, -\mathbf{1}^{m_3}, -\mathbf{1}^{m_4}, \mathbf{0}^{m_5}),$$

where $m_1, m_2, m_3, m_4, m_5 \geq 0$, $m_1 + m_2 + m_3 + m_4 + m_5 = n$, and $\mathbf{1}^m = (1, 1, \ldots, 1) \in \mathbb{Z}^m$ and $\mathbf{0}^m = (0, 0, \ldots, 0) \in \mathbb{Z}^m$ for $m \in \mathbb{Z}_+$ ($m = 0$ is allowed). For integers a and b we define

$$z(a, b) = a d_1 + b d_2 = \left(\ (a + b)\mathbf{1}^{m_1}, a\mathbf{1}^{m_2}, -(a + b)\mathbf{1}^{m_3}, -b\mathbf{1}^{m_4}, \mathbf{0}^{m_5}\ \right). \tag{5.1}$$

▶ **Lemma 5.2.** *Let a and b be integers.*

(1) $\|z(a + 1, b + 1) - z(a, b)\|_\infty = 2$ *if* $m_1 + m_3 \geq 1$.

(2) $\left\lceil \dfrac{z(a, b) + z(a + 1, b + 1)}{2} \right\rceil = z(a + 1, b), \qquad \left\lfloor \dfrac{z(a, b) + z(a + 1, b + 1)}{2} \right\rfloor = z(a, b + 1).$

▶ **Proposition 5.3** (Parallelogram inequality)**.** *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be a directed integrally convex function. For $x \in \mathrm{dom}\, f$ define $f_x(z) = f(x + z)$. If $m_1 + m_3 \geq 1$, then*

$$\begin{aligned} & f_x(\ \mathbf{0}^{m_1}, \mathbf{0}^{m_2}, \mathbf{0}^{m_3}, \mathbf{0}^{m_4}, \mathbf{0}^{m_5}\ ) + f_x(\ (a + b)\mathbf{1}^{m_1}, a\mathbf{1}^{m_2}, -(a + b)\mathbf{1}^{m_3}, -b\mathbf{1}^{m_4}, \mathbf{0}^{m_5}\ ) \\ & \geq f_x(\ a\mathbf{1}^{m_1}, a\mathbf{1}^{m_2}, -a\mathbf{1}^{m_3}, \mathbf{0}^{m_4}, \mathbf{0}^{m_5}\ ) + f_x(\ b\mathbf{1}^{m_1}, \mathbf{0}^{m_2}, -b\mathbf{1}^{m_3}, -b\mathbf{1}^{m_4}, \mathbf{0}^{m_5}\ ) \qquad (a, b \in \mathbb{Z}_+). \end{aligned} \tag{5.2}$$

Proof. Using notation (5.1) we can rewrite (5.2) as

$$f_x(z(0, 0)) + f_x(z(a, b)) \geq f_x(z(a, 0)) + f_x(z(0, b)). \tag{5.3}$$

We may assume $a, b \geq 1$ and $\{z(0, 0), z(a, b)\} \subseteq \mathrm{dom}\, f_x$, since otherwise the inequality (5.3) is trivially true. By directed integral convexity of f, we have

$$f_x(z(k, l)) + f_x(z(k + 1, l + 1)) \geq f_x(z(k + 1, l)) + f_x(z(k, l + 1)) \tag{5.4}$$

for $k, l \in \mathbb{Z}_+$. By adding these inequalities for (k, l) with $0 \leq k \leq a - 1$, $0 \leq l \leq b - 1$, we obtain (5.3). Note that all terms appearing in the above inequalities are finite. ◀

The parallelogram inequality with permutations of coordinates can be stated in an alternative form. Using notation $d_i = (d_{i1}, \ldots, d_{in})$ and $\Delta = \{(-1,-1),(0,0),(1,1),(0,-1),(1,0)\}$ we define

$$\mathcal{D} = \{(d_1, d_2) \mid (d_{1j}, d_{2j}) \in \Delta \ \ (j = 1,\ldots,n), \ \ \|d_1 + d_2\|_\infty = 2\}. \tag{5.5}$$

For $d_1 = (\mathbf{1}^{m_1}, \mathbf{1}^{m_2}, -\mathbf{1}^{m_3}, \mathbf{0}^{m_4}, \mathbf{0}^{m_5})$ and $d_2 = (\mathbf{1}^{m_1}, \mathbf{0}^{m_2}, -\mathbf{1}^{m_3}, -\mathbf{1}^{m_4}, \mathbf{0}^{m_5})$ we have $(d_1, d_2) \in \mathcal{D}$ as long as $m_1 + m_3 \geq 1$, and any $(d_1, d_2) \in \mathcal{D}$ can be put in this form through a suitable simultaneous permutation of coordinates of d_1 and d_2. Therefore, Proposition 5.3 can be rephrased as follows: If $(d_1, d_2) \in \mathcal{D}$, then $f(x) + f(x + ad_1 + bd_2) \geq f(x + ad_1) + f(x + bd_2)$ $(a, b \in \mathbb{Z}_+)$.

A generalized form of parallelogram inequality is given in the following proposition, where $\mathcal{D}^\top = \{(d_2, d_1) \mid (d_1, d_2) \in \mathcal{D}\}$.

▶ **Proposition 5.4.** *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be a directed integrally convex function, $x \in \mathrm{dom}\, f$, and $d_1^1, \ldots, d_1^K, d_2^1, \ldots, d_2^L \in \{-1, 0, +1\}^n$, where $K \geq 1$ and $L \geq 1$. If*

$$(d_1^k, d_2^l) \in \mathcal{D} \cup \mathcal{D}^\top \qquad (k = 1,\ldots,K;\ l = 1,\ldots,L), \tag{5.6}$$

$$\mathrm{supp}^+(d_1^k) \cap \mathrm{supp}^-(d_1^{k'}) = \emptyset \quad (k \neq k'), \qquad \mathrm{supp}^+(d_2^l) \cap \mathrm{supp}^-(d_2^{l'}) = \emptyset \quad (l \neq l'), \tag{5.7}$$

then

$$f(x) + f(x + \sum_{k=1}^{K} a^k d_1^k + \sum_{l=1}^{L} b^l d_2^l) \geq f(x + \sum_{k=1}^{K} a^k d_1^k) + f(x + \sum_{l=1}^{L} b^l d_2^l) \quad (a^k, b^l \in \mathbb{Z}_+). \tag{5.8}$$

Proof. With notation $x(k,l) = x + \sum_{i=1}^{k} a^i d_1^i + \sum_{j=1}^{l} b^j d_2^j$, the inequality (5.8) is rewritten as

$$f(x(0,0)) + f(x(K,L)) \geq f(x(K,0)) + f(x(0,L)). \tag{5.9}$$

We may assume $K, L \geq 1$ and $\{x(0,0), x(K,L)\} \subseteq \mathrm{dom}\, f$, since otherwise (5.9) is trivially true. Since $x(k,l) = x(k-1,l-1) + a^k d_1^k + b^l d_2^l$ and $(d_1^k, d_2^l) \in \mathcal{D} \cup \mathcal{D}^\top$ by assumption, we can apply the parallelogram inequality (5.2) to obtain

$$f(x(k-1,l-1)) + f(x(k,l)) \geq f(x(k,l-1)) + f(x(k-1,l)).$$

By adding these inequalities for (k,l) with $1 \leq k \leq K$ and $1 \leq l \leq L$, we obtain (5.9). Note that all terms appearing in the above inequalities are finite by (5.6) and (5.7). ◀

The assumptions in Proposition 5.4 are met in the following case.

▶ **Lemma 5.5.** *Conditions (5.6) and (5.7) are satisfied if $\{d_1^1, \ldots, d_1^K, d_2^1, \ldots, d_2^L\} \subseteq \{\mathbf{1}_{A_1} - \mathbf{1}_{B_1}, \mathbf{1}_{A_2} - \mathbf{1}_{B_2}, \ldots, \mathbf{1}_{A_s} - \mathbf{1}_{B_s}\}$ for nested families $A_1 \subseteq A_2 \subseteq \cdots \subseteq A_s$ and $B_1 \supseteq B_2 \supseteq \cdots \supseteq B_s$ of subsets of $\{1,\ldots,n\}$ such that $A_s \cap B_1 = \emptyset$ and $A_1 \cup B_s \neq \emptyset$.*

5.3 Scaling operation

Directed integrally convex functions are stable under scaling for arbitrary n, just as L$^\natural$-convex functions. Recall that the scaling operation preserves (general) integral convexity only when $n \leq 2$.

▶ **Theorem 5.6.** *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be a directed integrally convex function and $\alpha \in \mathbb{Z}_{++}$. Then the scaled function f^α is directed integrally convex.*

Proof. To show (1.1) for f^α, it suffices to consider the inequality for f with $x = \mathbf{0}$ and $y = (2\alpha\mathbf{1}^{m_1}, \alpha\mathbf{1}^{m_2}, -2\alpha\mathbf{1}^{m_3}, -\alpha\mathbf{1}^{m_4}, \mathbf{0}^{m_5}) \in \mathbb{Z}^n$, where $m_1 + m_3 \geq 1$. That is, we are to prove

$$\begin{aligned} &f(\ \mathbf{0}^{m_1}, \mathbf{0}^{m_2}, \mathbf{0}^{m_3}, \mathbf{0}^{m_4}, \mathbf{0}^{m_5}\ ) + f(\ 2\alpha\mathbf{1}^{m_1}, \alpha\mathbf{1}^{m_2}, -2\alpha\mathbf{1}^{m_3}, -\alpha\mathbf{1}^{m_4}, \mathbf{0}^{m_5}\ ) \\ &\geq f(\ \alpha\mathbf{1}^{m_1}, \alpha\mathbf{1}^{m_2}, -\alpha\mathbf{1}^{m_3}, \mathbf{0}^{m_4}, \mathbf{0}^{m_5}\ ) + f(\ \alpha\mathbf{1}^{m_1}, \mathbf{0}^{m_2}, -\alpha\mathbf{1}^{m_3}, -\alpha\mathbf{1}^{m_4}, \mathbf{0}^{m_5}\ ), \end{aligned}$$

which holds since it is a special case of the parallelogram inequality (5.2) with $a = b = \alpha$. ◀

5.4 Proximity theorem

The α-local proximity theorem with linear bound $n(\alpha-1)$ holds for directed integrally convex functions in n variables for all n. Recall that for (general) integrally convex functions the bound $n(\alpha-1)$ is valid only when $n \leq 2$, whereas it is valid for L$^\natural$-convex functions for all n.

▶ **Theorem 5.7.** *Let $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be a directed integrally convex function, $\alpha \in \mathbb{Z}_{++}$, and $x^\alpha \in \mathrm{dom}\, f$. If $f(x^\alpha) \leq f(x^\alpha + \alpha d)$ for all $d \in \{-1, 0, +1\}^n$, then there exists a minimizer $x^* \in \mathbb{Z}^n$ of f with $\|x^\alpha - x^*\|_\infty \leq n(\alpha - 1)$.*

To prove Theorem 5.7 we may assume $x^\alpha = \mathbf{0}$. Define $S = \{x \in \mathbb{Z}^n \mid \|x\|_\infty \leq n(\alpha-1)\}$, $W = \{x \in \mathbb{Z}^n \mid \|x\|_\infty = n(\alpha-1)+1\}$, and let μ be the minimum of $f(x)$ taken over $x \in S$ and $\hat{x}$ be a point in S with $f(\hat{x}) = \mu$. We shall show $f(y) \geq \mu$ for all $y \in W$. Then Theorem 2.3 (box-barrier property) implies that $f(z) \geq \mu$ for all $z \in \mathbb{Z}^n$.

▶ **Lemma 5.8.** *Each vector $y \in W$ can be represented as $y = (d_1^1 + d_1^2 + \cdots + d_1^{(n-1)(\alpha-1)}) + \alpha d_2$ with $(d_1^k, d_2) \in \mathcal{D} \cup \mathcal{D}^\top$ for $k = 1, 2, \ldots, (n-1)(\alpha-1)$.*

Proof. Fix $y = (y_1, \ldots y_n) \in W$ and put $m = \|y\|_\infty$, which is equal to $n(\alpha-1)+1$. With $A_k = \{i \mid y_i \geq m + 1 - k\}$, $B_k = \{i \mid y_i \leq -k\}$ $(k = 1, \ldots, m)$, we can represent y as $y = \sum_{k=1}^m (\mathbf{1}_{A_k} - \mathbf{1}_{B_k})$. We have $A_1 \subseteq A_2 \subseteq \cdots \subseteq A_m$, $B_1 \supseteq B_2 \supseteq \cdots \supseteq B_m$, $A_m \cap B_1 = \emptyset$, and $A_1 \cup B_m \neq \emptyset$.

Claim 1: $\exists\, k_0 \in \{1, 2, \ldots, m-\alpha+1\}$ s.t. $(A_{k_0}, B_{k_0}) = (A_{k_0+j}, B_{k_0+j})$ for $j = 1, 2, \ldots, \alpha-1$.

Proof of Claim 1. We may assume $A_1 \neq \emptyset$. Define $(a_k, b_k) = (|A_k|, n - |B_k|)$ for $k = 1, 2, \ldots, m$ and $s = |\mathrm{supp}^+(y)|$. The sequence $(a_k, b_k)_{k=1,2,\ldots,m}$ is nondecreasing in $\mathbb{Z}^2$, satisfying $(1, s) \leq (a_1, b_1) \leq (a_2, b_2) \leq \cdots \leq (a_m, b_m) \leq (s, n)$. Since $m = n(\alpha-1)+1$ and the length of a strictly increasing chain contained in the interval $[(1, s), (s, n)]$ in $\mathbb{Z}^2$ is bounded by n, the sequence $\{(a_k, b_k)\}_{k=1,2,\ldots,m}$ must contain a constant subsequence of length $\geq \alpha$. Hence follows the claim. ◀

With reference to the index k_0 in Claim 1 we define

$$d_2 = \mathbf{1}_{A_{k_0}} - \mathbf{1}_{B_{k_0}}, \qquad d_1^k = \begin{cases} \mathbf{1}_{A_k} - \mathbf{1}_{B_k} & (1 \leq k \leq k_0 - 1), \\ \mathbf{1}_{A_{k+\alpha}} - \mathbf{1}_{B_{k+\alpha}} & (k_0 \leq k \leq m - \alpha = (n-1)(\alpha-1)). \end{cases}$$

Then we have $y = \sum_{k=1}^m (\mathbf{1}_{A_k} - \mathbf{1}_{B_k}) = (d_1^1 + d_1^2 + \cdots + d_1^{(n-1)(\alpha-1)}) + \alpha d_2$. Moreover, we have $(d_1^k, d_2) \in \mathcal{D} \cup \mathcal{D}^\top$ $(k = 1, 2, \ldots, (n-1)(\alpha-1))$ by Lemma 5.5. ◀

By Lemma 5.8 and inequality (5.8) with $K = (n-1)(\alpha-1)$ and $L = 1$ we obtain

$$f(\mathbf{0}) + f(y) \geq f(d_1^1 + d_1^2 + \cdots + d_1^{(n-1)(\alpha-1)}) + f(\alpha d_2).$$

Here we have $d_1^1 + d_1^2 + \cdots + d_1^{(n-1)(\alpha-1)} \in S$ and hence $f(d_1^1 + d_1^2 + \cdots + d_1^{(n-1)(\alpha-1)}) \geq \mu$ by the definition of μ. We also have $f(\alpha d_2) \geq f(\mathbf{0})$ by α-local minimality of $\mathbf{0}$. Therefore,

$$f(y) \geq f(d_1^1 + d_1^2 + \cdots + d_1^{(n-1)(\alpha-1)}) + [f(\alpha d_2) - f(\mathbf{0})] \geq \mu + 0 = \mu.$$

This completes the proof of Theorem 5.7.

References

1 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin: *Network Flows—Theory, Algorithms and Applications*, Prentice-Hall, 1993.

2 P. Favati and F. Tardella: Convexity in nonlinear integer programming, *Ricerca Operativa*, **53** (1990), 3–44.

3 S. Fujishige: Bisubmodular polyhedra, simplicial divisions, and discrete convexity. *Discrete Optimization*, **12** (2014), 115–120.

4 S. Fujishige and K. Murota: Notes on L-/M-convex functions and the separation theorems, *Mathematical Programming*, **88** (2000), 129–146.

5 H. Hirai: L-extendable functions and a proximity scaling algorithm for minimum cost multiflow problem, *Discrete Optimization* **18** (2015), 1–37.

6 D. S. Hochbaum: Complexity and algorithms for nonlinear optimization problems, *Annals of Operations Research*, **153** (2007), 257–296.

7 D. S. Hochbaum and J. G. Shanthikumar: Convex separable optimization is not much harder than linear optimization, *Journal of the Association for Computing Machinery* **37** (1990), 843–862.

8 T. Ibaraki and N. Katoh: *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, 1988.

9 T. Iimura, K. Murota, and A. Tamura: Discrete fixed point theorem reconsidered. *Journal of Mathematical Economics*, **41** (2005), 1030–1036.

10 T. Iimura and T. Watanabe: Existence of a pure strategy equilibrium in finite symmetric games where payoff functions are integrally concave. *Discrete Applied Mathematics*, **166** (2014), 26–33.

11 S. Iwata, S. Moriguchi, and K. Murota: A capacity scaling algorithm for M-convex submodular flow. *Mathematical Programming*, **103** (2005), 181–202.

12 S. Iwata and M. Shigeno: Conjugate scaling algorithm for Fenchel-type duality in discrete convex optimization. *SIAM Journal on Optimization*, **13** (2003), 204–211.

13 N. Katoh, A. Shioura, and T. Ibaraki: Resource allocation problems. In: Pardalos, P. M., Du, D.-Z., Graham, R. L. (eds.) *Handbook of Combinatorial Optimization*, 2nd ed., Vol. 5, pp. 2897–2988, Springer, Berlin (2013).

14 S. Moriguchi, K. Murota, and A. Shioura: Scaling algorithms for M-convex function minimization. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E85-A** (2002), 922–929.

15 K. Murota: *Discrete Convex Analysis*, SIAM, 2003.

16 K. Murota and A. Tamura: Proximity theorems of discrete convex functions, *Mathematical Programming*, **99** (2004), 539–562.

17 G. van der Laan, D. Talman, and Z. Yang: Solving discrete systems of nonlinear equations. *European Journal of Operational Research*, **214** (2011), 493–500.

18 Z. Yang: On the solutions of discrete nonlinear complementarity and related problems. *Mathematics of Operations Research*, **33** (2008), 976–990.

19 Z. Yang: Discrete fixed point analysis and its applications. *Journal of Fixed Point Theory and Applications*, **6** (2009), 351–371.

Assigning Weights to Minimize the Covering Radius in the Plane*

Eunjin Oh[1] **and Hee-Kap Ahn**[2]

1 **Department of Computer Science and Engineering, POSTECH,**
77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea
jin9082@postech.ac.kr

2 **Department of Computer Science and Engineering, POSTECH,**
77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea
heekap@postech.ac.kr

Abstract

Given a set P of n points in the plane and a multiset W of k weights with $k \leq n$, we assign a weight in W to a point in P to minimize the maximum weighted distance from the weighted center of P to any point in P. In this paper, we give two algorithms which take $O(k^2n^2\log^4 n)$ time and $O(k^5 n\log^4 k + kn\log^3 n)$ time, respectively. For a constant k, the second algorithm takes only $O(n\log^3 n)$ time, which is near-linear.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Weighted center, facility location, weight assignment, combinatorial optimization, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.58

1 Introduction

Consider a set of robots lying at different locations in the plane. Each robot is equipped with a locomotion module so that it can move to a nearby facility to recharge its battery and return to its original location. We want to place a recharging facility for the robots such that the maximum travel time of them to reach the recharging facility is minimized. Then the Euclidean center of the robot locations may not be a good location for the recharging facility if the robots have huge differences in their speeds. For instance, consider three robots, each lying on a different corner of an equilateral triangle. If one of them has a much smaller speed compared to the speeds of the other two robots, the best location is very close to the corner where the low-speed robot lies. Hence, the recharging facility must be located at a *weighted center* of the robots by considering their speeds as weights of their placements.

In the weighted center problem, each input point $p \in P$ is associated with a positive weight and the *weighted distance* between an input point and a point of the plane is defined to be their distance divided by the associated weight of p. Then the point of the plane that minimizes the maximum weighted distance to input points is the center of the weighted input points, which we call the *weighted center*.

Dyer [8] studied the weighted center problem for a set of weighted points in the plane and gave a linear-time algorithm to compute their weighted center. Clearly, the weighted center coincides with the (unweighted) center if the associate weight is 1 for every input point.

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 58; pp. 58:1–58:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Imagine now that we are allowed to *reassign* the locomotion modules of the robots. Or, if the mobile robots are identical, except their speeds, we are allowed to *relocate* the robots. A relocation of robots (or a reassignment of locomotion modules) may change the weighted center and the maximum travel time for mobile robots to reach the weighted center. In other words, a clever assignment of robots (or their locomotion modules) to given locations may decrease the minimum of the objective function.

In this paper, we formally define this relocation problem and present algorithms for it. The *weight assignment problem* is defined as follows: given an input consisting of a set P of n points in the plane and a multiset $W = \{w_1, \ldots, w_k\}$ of k weights of positive real values with $k \leq n$, find an assignment of the weights in W to input points such that the maximum weighted distance from the weighted center to input points is minimized. We assume that every input point of P has the default weight 1. We assign the k weights to k points of P such that every weight is assigned to a point and each point gets one weight in W or the default weight 1, which we call an *assignment of weights* of W to P.

We regard an assignment of weights as a function. To be specific, for an assignment f of weights, $f(p)$ denotes the weight of W assigned to point $p \in P$. We use $c(f)$ to denote the weighted center of P with the assignment f, and call the maximum weighted distance from $c(f)$ to input points the *covering radius* of the assignment f and denote it by $r(f)$.

Obviously, there are $\binom{n}{k}$ different combinations of selecting k points from P and $k!$ different ways of assigning the k weights to a combination, and therefore there are $\Theta(n^k)$ different assignments of weights. Our goal is to find an assignment f of weights of W to the points of P that minimizes the covering radius $r(f)$ over all possible assignments of weights.

Related Work. As mentioned earlier, Dyer [8] studied the weighted center problem for a set of weighted points in the plane. He reformulated the problem as an optimization problem with linear inequalities and one quadratic inequality. Then he gave a linear-time algorithm to compute their weighted center using the technique by Megiddo [12]. Later, Megiddo [13] gave a linear-time algorithm for the same problem using a different technique.

In contrast, to our best knowledge, no algorithm is known for the weight assignment problem while there are works on several related problems. In the *inverse 1-center problem* on graphs, we are given a graph and a target vertex, and we are to increase or decrease the lengths of edges of the graph so that the target vertex becomes a center of the modified graph. The goal is to minimize the modification of the lengths. This means that we give additive weights to edges of the graph. Cai et al. [6] showed that this problem is NP-hard on a general directed graph. Recently, Alizadeh and Burkard [3] gave an $O(n^2 r)$-time algorithm for this problem on a tree, where r is the compressed depth of the tree. A variant of this problem is the *reverse 1-center problem*, in which we are to decrease the lengths of edges of the graph under a given budget. This problem is also known to be NP-hard even on a bipartite graph [5], and there is an $O(n^2 \log n)$-time algorithm on a tree by Zhang et al. [15].

Our weight assignment problem is closely related to the weight balancing problem which was studied by Barba et al. [4]. The input consists of a simple polygon, a target point inside the polygon, and a set of weights. The goal is to put the weights on the boundary of the polygon so that the barycenter (center of mass) of the weights coincides with the target point. They showed the existence of such a placement of weights under the condition that no input weight exceeds the sum of the other input weights. They also gave an algorithm to find such a placement in $O(k + n \log n)$ time, where k is the number of the weights and n is the number of the vertices of P. Our problem can be considered as a discrete version of this problem, but with a different criteria (minimizing the covering radius), in the sense that we place the weights on predetermined positions.

Our Result. In this paper, we present two algorithms that compute an assignment f of weights minimizing $r(f)$. The first algorithm returns an optimal assignment in $O(k^2n^2 \log^4 n)$ time using $O(kn)$ space. The second algorithm assumes that all weights in W are at most 1, and returns an optimal assignment in $O(k^5 n \log^4 k + kn \log^3 n)$ time using $O(kn)$ space, which is faster than the first algorithm when k is sufficiently small ($k = o(n^{1/3})$). Moreover, it takes only $O(n \log^3 n)$ time when k is a constant.

Another merit of our algorithms is that they are based on useful geometric intuition and are easy to implement though they use parametric search, the optimization technique developed by Megiddo [10]. The technique is an important tool for solving many geometric optimization problems efficiently, but algorithms based on it are often not easily implemented [2]. A main difficulty lies in computing the roots of the polynomials exactly whose signs determine the outcome of the comparisons made by the algorithm. However, as we will see later, in our algorithms, such a root is a covering radius of at most three weighted points and it can be computed without resorting to complicated methods.

2 Preliminaries

One way to deal with our problem is to find the weighted centers for all possible assignments of weights and choose the one with the minimum radius. Although there are $\Theta(n^k)$ different assignments of weights, there are only $O(k^3n^3)$ different weighted centers. This is because a weighted center is determined by at most three weighted points. That is, given an assignment of weights, there always exist at most three weighted points of P whose center coincides with the center of the whole weighted points. Thus the number of all possible weighted centers is at most the number of all possible 6-tuples $(p_1, p_2, p_3, w_1, w_2, w_3)$ such that $p_i \in P$ and $w_i \in W \cup \{1\}$ for $i = 1, 2, 3$, which is $O(k^3n^3)$. Moreover, this bound is asymptotically tight by the following lemma. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 1.** *There exist a set of n points and a set of k weights such that the number of all possible weighted centers is $\Omega(k^3n^3)$.*

2.1 Deciding Feasibility for a Weighted Center of Three Points

As noted above, the weighted center and the covering radius of a weight assignment are determined by at most three weighted points. But not every three weighted points define such a center and its covering radius. Here we show how to test this for three weighted points efficiently.

Let W' be the union of W and the multiset consisting of $n - k - 3$ numbers of weight 1. Consider three points from P and an assignment of three weights from W' to the points. We first test whether the three weighted points define a point in the plane at the same weighted distance from them. If such a point does not exist, there is no weighted center of the three weighted points. Otherwise, there is only one such point which is the weighted center of them. Let c denote the weighted center and r denote the covering radius of the three weighted points. Let $\langle p_1, \ldots, p_{n-3} \rangle$ be the sequence of the points in P, except the three points, in the increasing order with respect to the Euclidean distance from c. Let $\langle w_1, \ldots, w_{n-3} \rangle$ be the sequence of the weights in W', except the three weights, in the increasing order, The following lemma directly gives us an $O(n)$-time algorithm to decide whether the three weighted points determine the weighted center and the covering radius of an assignment of weights. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 2.** *There exists an assignment f of weights such that $c(f) = c$ and $r(f) = r$ if and only if $d(p_i, c)/w_i \leq r$ for $1 \leq i \leq n-3$.*

It is possible that two weighted points p and q determine the weighted center of the whole weighted points. In this case, the weighted center lies in the line passing through p and q. To handle this case, we consider two points from P and an assignment of two weights from W to the points. We compute the weighted center and decide whether the two weighted points determine the weighted center and the covering radius using Lemma 2.

We need to sort the points in P repeatedly for each weighted center determined by a combination of three points (or two points) from P and three weights (or two weights) from W while it suffices to sort the weights in W just once. Thus, the total running time is $O(k^3n^4 \log n)$.

Note that this algorithm returns all possible weighted centers. Thus it can be used for the problem with some different optimization criteria other than the minimization of the covering radius. For example, we can find the assignment f of weights such that the center $c(f)$ is the closest to a given point in the same time.

3 A Fast Algorithm using $O(kn)$ Space

In this section, we give an $O(k^2n^2 \log^4 n)$-time algorithm for finding an assignment f of weights that minimizes $r(f)$. This algorithm does not consider all possible weighted centers. Instead, it uses parametric search due to Megiddo [10]. To apply this technique, we need to devise a decision algorithm which is used as a subprocedure of the main algorithm.

3.1 A Decision Algorithm

Let r be an input of the decision algorithm. The decision algorithm decides whether there is an assignment f of weights with $r(f) \leq r$. In other words, it decides whether there are a point c and an assignment f of weights such that $d(p, c)/f(p) \leq r$ for all points $p \in P$. If this is the case, we call such a point c an *r-center* with respect to f.

Instead of considering all $\Theta(k^3n^3)$ combinations of three points in P and three weights in W, this algorithm considers all $O(kn)$ point-weight pairs. Our algorithm is based on the following lemma. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 3.** *For an assignment f of weights with $r(f) \leq r$, there is an r-center c with respect to f satisfying $d(p', c)/f(p') = r$ for some point $p' \in P$.*

By the above lemma, there is a point $p \in P$ and a weight $w \in W \cup \{1\}$ such that the circle centered at p with radius wr contains an r-center c with respect to some assignment f of weights if $r(f) \leq r$. Thus our strategy is to find an r-center, if it exists, lying on such a circle with respect to some weight assignment f satisfying that $f(p) = w$.

For each pair of a point $p \in P$ and a weight $w \in W \cup \{1\}$, we consider the circle C centered at p with radius wr. For a point $c \in C$, we check whether there exists an assignment f of weights such that the disk centered at q with radius $f(q)r$ contains c for all points $q \in P \setminus \{p\}$.

To this end, for each point $q \in P \setminus \{p\}$, we compute $k+1$ concentric circles centered at q with radius $w_1r, w_2r, \ldots, w_{k+1}r$, where $w_1, w_2, \ldots, w_{k+1}$ are weights in $W \cup \{1\}$. Then we compute the intersections of these circles with C and sort them along C in $O(kn \log(kn)) = O(kn \log n)$ time. (In a degenerate case, there can be more than one circle passing through the same

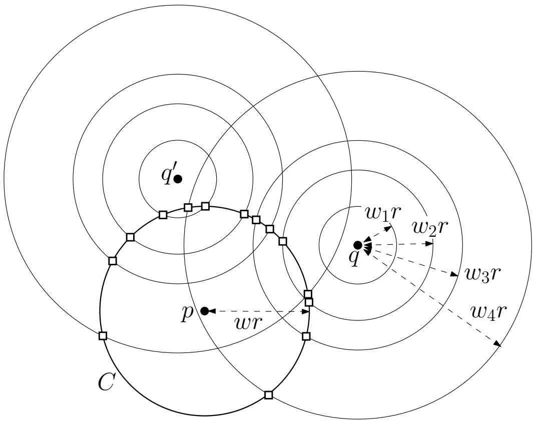


Figure 1 We compute the points marked with squares for all points in $q \in P \setminus \{p\}$ and sort them along C.

intersection point and we treat their intersection points as distinct points lying in the same position. Details can be found in the full version of this paper. In the following, we assume that exactly one circle passes through one intersection point.) See Figure 1 for an illustration.

Now, we have $O(kn)$ intersection points sorted along C. These intersection points subdivide the circle C into $O(kn)$ pieces, which we call *intervals* on C. We say an assignment f of weights is *feasible* for an interval if for a point c in the interval, $d(q,c)/f(q)$ is at most r for all points $q \in P$. Note that f is feasible for any point in the interval if f is feasible for a point in the interval. Therefore, for any point c lying in an interval, the set of feasible assignments of weights remains the same. We can test in $O(n \log n)$ time whether there exists a feasible assignment for an interval on C by slightly modifying Lemma 2.

Instead of applying this test repeatedly for each interval on C, which takes $O(kn^2 \log n)$ time in total, we can do this in $O(kn \log n)$ time in total for all intervals on C as follows. Consider the intervals one by one in clockwise order along C. Note that for any two consecutive intervals, there is only one disk centered at a point in P with radius rw for some $w \in W$ that contains one interval but does not contain the other interval. We use this fact in the following lemma.

▶ **Lemma 4.** *We can decide whether there is a feasible assignment of weights or not for every interval on C in $O(kn \log n)$ time in total.*

Proof. We first show how to check the existence of a feasible assignment for an interval μ in $O(kn)$ time. Then we show how we do this for all intervals on C efficiently.

We sort the weights in $W \cup \{1\}$ in the increasing order and denote the sorted list by $\langle w_1, \ldots, w_{k+1} \rangle$. Let ℓ_0 be the smallest index with $w_{\ell_0} = 1$, and c be a point in μ. For each point q in $P \setminus \{p\}$, let $\pi(q)$ be the smallest index such that $d(q,c)/w_{\pi(q)} \leq r$ and let $\pi(p)$ be the index indicating w. The indices $\pi(q)$ for all points q in P can be computed in $O(kn)$ time in total. Then, we sort the points in P in the increasing order with respect to $\pi(\cdot)$ and denote the sorted list by $\langle q_1, \ldots, q_n \rangle$.

Consider the simple case that $k = n$. Then there exists a feasible assignment for μ if and only if $\pi(q_\ell) \leq \ell$ for all indices $1 \leq \ell \leq n-1$. This is because for every point q in P, we have $d(q,c)/w_j \leq r$ for all indices $j \geq \pi(q)$. Thus, we can check the existence of a feasible assignment for μ by comparing $\pi(q_\ell)$ and ℓ for all indices $1 \leq \ell \leq n$, which can be done in $O(n)$ time.

For the case that $k < n$, a similar property holds: there exists a feasible assignment for μ if and only if

- $\pi(q_\ell) \leq \ell$ for all indices $1 \leq \ell \leq \ell_0 - 1$,
- $\pi(q_\ell) \leq \ell_0$ for all indices $\ell_0 \leq \ell \leq \ell_0 + n - k - 1$, and
- $\pi(q_\ell) \leq \ell - n + k + 1$ for all indices $\ell_0 + n - k \leq \ell \leq n$.

Thus, we can check the existence of a feasible assignment for μ in $O(kn)$ time.

To check the existence of a feasible assignment for the interval μ' next to μ in clockwise order along C, we do not need to compute all such indices and compare them again. Recall that there is exactly one disk C' centered at a point $q \in P \setminus \{p\}$ with radius rw for some $w \in W$ that contains either μ or μ'. If C' contains μ but does not contain μ', $\pi(q)$ increases by one. If C' contains μ' but does not contain μ, $\pi(q)$ decreases by one. Note that $\pi(q_\ell)$ remains the same for all points $q_\ell \in P \setminus \{q\}$ for both cases.

To use this property, we maintain $2(k+1)$ pointers $U_1, \ldots, U_{k+1}$ and $L_1, \ldots, L_{k+1}$. For an index $1 \leq i \leq k+1$, the pointer U_i points to q_ℓ with the largest index ℓ among the points such that $\pi(q_\ell)$ is equal to i. Similarly, for an index $1 \leq i \leq k+1$, the pointer L_i points to q_ℓ with the smallest index ℓ among the points such that $\pi(q_\ell)$ is equal to i.

Note that we already know whether $\pi(q)$ increases or decreases by one when we move from μ to μ'. Here, we have to update not only $\pi(q)$ but also the pointers. Moreover, we have to reorder $\langle q_1, \ldots, q_n \rangle$ in the increasing order with respect to $\pi(\cdot)$.

We show how to do this for the case that $\pi(q)$ increases by one. The other case can be handled analogously. We first find the point q_u that the pointer $U_{\pi(q)}$ points to. Then we swap the positions for q and q_u on the sequence $\langle q_1, \ldots, q_n \rangle$ and let $U_{\pi(q)}$ point to q. This does not violate the property that $\pi(q_\ell) \leq \pi(q_{\ell+1})$ for all indices $1 \leq \ell < n$ since $\pi(q) = \pi(q_u)$. Then we update $\pi(q)$ to $\pi(q) + 1$ and update the pointers accordingly.

Here, we do not need to compare $\pi(q_\ell)$ and the index again for a point q_ℓ in $P \setminus \{q, q_u\}$. Thus to check the existence of a feasible assignment for μ', it suffices to compare $\pi(q_\ell)$ and the index for $q_\ell = q, q_u$. This can be done in constant time, which implies that each update can be done in the same time. Since there are $O(kn)$ intervals on C, updating the information takes $O(kn)$ time. Therefore, the running time of the procedure is dominated by the time for sorting the intersection points along C and computing the intervals, which is $O(kn \log n)$. ◀

With the argument in this section, the following lemma holds.

▶ **Lemma 5.** *Given a radius $r > 0$, we can decide in $O(k^2 n^2 \log n)$ time using $O(kn)$ space whether there exists an assignment f of weights with $r(f) \leq r$ or not.*

3.2 An Overall Algorithm

For ease of presentation, we first show how to compute an optimal solution in $O(k^2 n^2 \log^3 n)$ time using $O(k^2 n^2)$ space. At the end of this section, we show how to reduce the space into $O(kn)$ at the expense of increased time complexity by an $O(\log n)$ factor.

To obtain an optimal solution, we apply parametric search [10] using the decision algorithm in Section 3.1. Let r^* be the minimum of $r(f)$ over all possible assignments f of weights.

We consider the arrangement $A(r)$ of the circles $C_{p,w}(r)$ for all point-weight pairs (p, w), where $C_{p,w}(r)$ is the circle centered at p with radius rw. Let $\mathcal{C}(r)$ be the set of such circles $C_{p,w}(r)$ over all point-weight pairs (p, w). Here, $r > 0$ is a variable.

As r becomes larger, the combinatorial structure of $A(r)$ changes. To be specific, the combinatorial structure of $A(r)$ changes $O(k^3 n^3)$ times. To see this, we observe that there exists a vertex of $A(r)$ which is an intersection of three circles $C_{p_i,w_i}(r)$ (or two circles) in $\mathcal{C}(r)$ for $i = 1, 2, 3$ when $A(r)$ changes. Moreover, for any three point-weight pairs (p_i, w_i) for

$i = 1, 2, 3$ (or $i = 1, 2$), there exist at most two radii r such that the common intersection of $C_{p_i,w_i}(r)$ is not empty. This is because the trajectory of the intersections between two increasing circles forms a hyperbolic curve, and two hyperbolic curves cross at most twice. (Note that C_{p_i,w_i} is a circle, not a disk.)

These radii partition the real value space $\mathbb{R}$ into intervals such that for any value r in the same interval the combinatorial structure of $A(r)$ remains the same. We search the interval where r^* lies using the decision algorithm in Section 3.1. There are $\Theta(k^3n^3)$ such radii, but we do not consider all of them. Instead, we search for the interval containing r^* in $O(\log n)$ iterations. In each iteration, we consider $O(k^2n^2)$ radii and reduce the search space (intervals). Moreover, in each iteration, we apply the decision algorithm $O(\log n)$ times, which leads to the running time of $O(k^2n^2 \log^3 n)$. Details are described in the following lemma and its proof.

▶ **Lemma 6.** *The combinatorial structure of $A(r^*)$ can be computed in $O(k^2n^2 \log^3 n)$ time using (k^2n^2) space.*

Proof. Given a radius $r > 0$, we first introduce a simple way to compute the arrangement $A(r)$. Later, this will be used for computing $A(r^*)$ efficiently. For a point $p \in P$ and a weight $w \in W \cup \{1\}$, we compute the intersections of $C_{p,w}(r)$ and $C_{p_i,w_i}(r)$ for all points p_i in $P \setminus \{p\}$ and all weights $w_i \in W \setminus \{w\} \cup \{1\}$. Each circle $C_{p_i,w_i}(r)$ intersects $C_{p,w}(r)$ at most twice for any fixed radius $r > 0$. Let $\mathcal{I}_{p,w}(r)$ be the set of all such intersection points. We sort the points in $\mathcal{I}_{p,w}(r)$ along $C_{p,w}(r)$. Once this is done for all points p and all weights $w \in W \cup \{1\}$, we can construct $A(r)$.

Here, computing the combinatorial structure of $A(r)$ is equivalent to sorting the points in $\mathcal{I}_{p,w}(r)$ along $C_{p,w}(r)$ for all point-weight pairs (p, w). We apply this procedure to compute the combinatorial structure of $A(r^*)$ without computing r^* explicitly. To sort the points in $\mathcal{I}_{p,w}(r^*)$ along $C_{p,w}(r^*)$, we compare the relative positions for two points in the set $O(kn \log n)$ times since the number of points in the set is $O(kn)$.

Suppose that we want to compare the relative positions for two points $u_1(r^*), u_2(r^*)$ in the set along $C_{p,w}(r)$. As r increases, the relative positions between $u_1(r)$ and $u_2(r)$ change at most once. Moreover, they change when $C_{p,w}(r)$ and the two circles defining $u_1(r)$ and $u_2(r)$ meet at exactly one point. This happens at most twice. Let r_1 and r_2 be such two radii with $r_1 \leq r_2$. Then we decide whether $r^* \leq r_1$, $r_1 \leq r^* \leq r_2$, or $r_2 \leq r^*$ using the decision algorithm in Section 3.1. With this, we can decide the relative positions between $u_1(r^*)$ and $u_2(r^*)$ without computing r^* explicitly. Thus, we can compare two points in $\mathcal{I}_{p,w}(r^*)$ in $O(k^2n^2 \log n)$ time. Since we do this $O(kn \log n)$ times, we can sort the points in $O(k^3n^3 \log^2 n)$ time. Since there are $O(kn)$ point-weight pairs (p, w), the total running time is $O(k^4n^4 \log^2 n)$.

Here, we apply the decision algorithm in Section 3.1 twice for each comparison, once with $r = r_1$ and once with $r = r_2$. We reduce the running time of the overall algorithm by reducing the number of executions of the decision algorithm. Suppose that we want to do m comparisons which are independent to each other. As we did in the previous procedure, we compute at most two radii from each comparison where the relative positions of the two points change. Then we have at most $2m$ radii. We sort them and apply binary search to compute the smallest interval containing r^*. After applying the decision algorithm $O(\log m)$ times, we can complete m comparisons.

In our problem, comparisons performed on two different circles are independent to each other. In each circle $C_{p,w}(r)$, we have $O(\log n)$ sets each of which consists of $O(kn)$ comparisons that are independent to each other. Indeed, Cole [7] gave a parallel algorithm to sort m elements in $O(\log m)$ time using $O(m)$ processors. Note that comparisons performed in different processors are independent to each other.

Thus, we sort the points in $\mathcal{I}_{p,w}(r^*)$ in $O(\log(kn)) = O(\log n)$ iterations and in each iteration we apply $O(kn)$ comparisons. We compute the whole combinatorial structure of $A(r^*)$ in $O(\log n)$ iterations and in each iteration we apply $O(k^2n^2)$ comparisons. This can be done in $O(T(k,n)\log^2 n + k^2n^2\log^2 n)$ time, where $T(k,n)$ is the running time of the decision algorithm. ◀

Now, we have the combinatorial structure of $A(r^*)$ while r^* is not known yet. The following lemma gives us a procedure to compute r^* in $O(k^2n^2\log^2 n)$ time.

▶ **Lemma 7.** *Given the combinatorial structure of $A(r^*)$, an optimal weight assignment and its covering radius r^* can be computed in $O(k^2n^2\log^2 n)$ time.*

Proof. In this proof, we use the notation defined in Lemma 6. We say that three circles $C_{p_i,w_i}(r)$ for $i = 1,2,3$ *define* a radius r' if they intersect at one point for $r = r'$. We already showed that there are at most three circles $C_{p_i,w_i}(r)$ for $i = 1,2,3$ that define r^*. Thus, in the set $\mathcal{I}_{p_1,w_1}(r^*)$ sorted along $C_{p_1,w_1}(r^*)$, a point corresponding to $C_{p_2,w_2}(r^*)$ and a point corresponding to $C_{p_3,w_3}(r^*)$ are consecutive. Note that when we sort the points in $\mathcal{I}_{p_1,w_1}(r^*)$, we cannot decide whether two points coincide or not. Instead, we give an arbitrary order for such points.

Let R be the set of radii r defined by three circles $C_{p_i,w_i}(r^*)$ for $i = 1,2,3$ such that the point corresponding to $C_{p_2,w_2}(r^*)$ and the point corresponding to $C_{p_3,w_3}(r^*)$ are consecutive in the set $\mathcal{I}_{p_1,w_1}(r^*)$. Since there are $O(k^2n^2)$ edges in the arrangement $A(r^*)$, there are the same number of such radii.

We sort the radii on R and apply binary search on it using the decision algorithm in Section 3.1 and find the smallest radius of R for which the decision algorithm returns "yes". Then the smallest radius is r^*. ◀

Since we maintain the whole combinatorial structure of the arrangement $A(r)$, we use $O(k^2n^2)$ space in the previous algorithm. We can reduce the space complexity to $O(kn)$ by computing only some partial information about the combinatorial structure of $A(r)$. However, this increases the running time of the algorithm by an $O(\log n)$ factor.

▶ **Lemma 8.** *We can compute an interval containing r^* such that the combinatorial structure of $A(r)$ remains the same for any r in the interval in $O(k^2n^2\log^4 n)$ time using $O(kn)$ space.*

Proof. Here, we modify the algorithm in Lemma 6 as follows. Recall that in each iteration of the previous algorithm, we consider all point-weight pairs (p,w) and perform $O(kn)$ comparisons for sorting the intersection points in each $\mathcal{I}_{p,w}$. In this algorithm, we do this in $O(\log n)$ subiterations of an iteration as follows.

We maintain an interval u containing r^*. Initially, u is set to $[-\infty, +\infty]$. As we apply the algorithm, the interval becomes a smaller subinterval. In each subiteration, we consider $O(kn)$ radii corresponding to each comparison for a point-weight pair (p,w) and discard the radii which lie outside the interval u. Then we choose the median of them. We do this for all point-weight pairs (p,w), and we have $O(kn)$ medians in total. Then we sort the medians and apply binary search to compute the interval between two consecutive medians containing r^* in $O(T(n,k)\log n)$ time, where $T(n,k)$ is the running time of the decision algorithm. In total, each subiteration takes $O(T(n,k)\log n + k^2n^2)$ time using $O(kn)$ space.

Now, we show that in $O(\log n)$ subiterations, we complete $O(k^2n^2)$ comparisons. In the ith subiteration, we discard $(1/2 + 1/4 + \ldots + 1/2^i)|R|$ radii in total, where $|R| = O(k^2n^2)$ is the number of radii. Note that we discard the radius corresponding to some comparison once we complete the comparison. In $O(\log n)$ subiterations, we discard all radii, which means that we complete $O(k^2n^2)$ comparisons.

In this algorithm, each iteration takes $O(T(n,k)\log^2 n)$ time. Since we have $O(\log n)$ iterations in total as the algorithm in Lemma 6, the running time of this algorithm is $O(T(n,k)\log^3 n)$. ◀

Once we have an interval u containing r^* such that the combinatorial structure of $A(r)$ remains the same for any r in the interval, the procedure described in the proof of Lemma 7 can also be improved.

▶ **Lemma 9.** *Given an interval containing r^* such that the combinatorial structure of $A(r)$ remains the same for any r in the interval, an optimal assignment and its covering radius r^* can be found in $O(k^2n^2\log n)$ time.*

Proof. We consider a point-weight pair (p,w) first. Instead of computing the whole arrangement, we compute the edges and the vertices lying on $C_{p,w}(r^*)$. This takes $O(kn\log n)$ time because we already have the smallest interval u containing r^*. Then we apply the algorithm in Lemma 7 on the edges and vertices lying on $C_{p,w}(r^*)$. The algorithm returns the minimum radius r in u such that the decision algorithm with input r answers "yes". We do this for all point-weight pairs (p,w). Then we have $O(kn)$ radii one of which is exactly r^*. We again apply binary search on these radii to find the minimum radius r over such radii that make the decision algorithm return "yes". Clearly, the minimum radius is exactly r^*. ◀

▶ **Theorem 10.** *Given a set P of n points in the plane and a multiset W of k weights with $k \le n$, an assignment f of weights that minimizes $r(f)$ can be found in $O(k^2n^2\log^4 n)$ time using $O(kn)$ space.*

4 A Faster Algorithm for a Small Set of Weights

We can improve the algorithm in Section 3 for the case that k is sufficiently small compared to n. In this algorithm, we restrict input weights to be at most 1.

Let f^* be an optimal assignment of weights, that is, $r(f^*) = r^*$, and let c^* be the weighted center of P with respect to f^*. A point p in P is called a *determinator* if $d(p,c^*)/f^*(p) = r^*$. We already observed that there exist two or three determinators for any point set P. Moreover, if there exist exactly two determinators, then the two determinators and c^* are collinear.

The algorithm in this section is based on the observation that if $f^*(p) = 1$ for a determinator p, then $d(p,c^*) \ge d(p',c^*)$ for any point p' in P. The following lemma provides a more general observation. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 11.** *Let f^* be an optimal assignment of weights with minimum number of determinators. If a determinator p is the ith closest point of P from c^*, it is assigned the ith smallest element in W', where W' is the union of W and the multiset consisting of $n-k$ numbers of weight 1.*

Combining this with Lemma 2, we have the following corollary.

▶ **Corollary 12.** *There exists an optimal assignment f^* of weights that maps the ith closest point of P from $c(f^*)$ to the ith smallest element in W', where W' is the union of W and the multiset consisting of $n-k$ numbers of weight 1.*

Lemma 11 reduces the number of candidates for a determinator with its weight compared to the algorithm in Section 3. Recall that the algorithm considers each of $O(kn)$ point-weight pairs as a determinator and its weight.

We consider three different cases. The first case is that every determinator is assigned weight 1. The second case is that every determinator is assigned a weight strictly less than 1. The third case deals with all remaining situations.

Case 1: Every determinator is assigned weight 1. By Lemma 11, the determinators are the farthest points of c^* among all points in P in this case. This means that the (unweighted) center of P coincides with the weighted center $c(f^*)$ with the assignment f^*. After computing the (unweighted) center of P in $O(n)$ time [11], it suffices to check whether the center is valid or not by using the procedure by Lemma 2. Therefore, this case can be handled in $O(n)$ time in total, excluding the time for sorting the weights in W.

Case 2: Every determinator is assigned a weight smaller than 1. By Lemma 11, a determinator is one of the k closest points from c^*. This is related to the concept of the *order-k Voronoi diagram.* The order-k Voronoi diagram is a generalization of the standard Voronoi diagram. It partitions the plane into regions such that every point in the same region has the same k closest sites. The complexity of the order-k Voronoi diagram of n point sites is $O(kn)$ [9]. There are a number of algorithms to compute the order-k Voronoi diagram with different running times [1, 14]. Among them we use the algorithm in [14], which runs in $O(n\log n + nk2^{c\log^* k}) \leq O(n\log n + nk\log k)$ time using $O(kn)$ space.

In terms of the order-k Voronoi diagram, Lemma 11 can be interpreted as follows. All determinators are sites corresponding to the cell of the order-k Voronoi diagram of P containing c^*. To use this observation, we consider each cell of the order-k Voronoi diagram of P. For each cell, we assign the k weights in W to the k sites corresponding to the cell by applying the algorithm in Section 3 to the sites. This takes $O(k^4\log^4 k)$ time. Then we check whether the weighted center is valid or not. To this end, it suffices to check the distance from the center to the farthest point of the center by Lemma 2. This can be done in $O(\log n)$ time once we have the farthest-point Voronoi diagram of P. In total, this takes $O(k^5 n\log^4 k + T(n,k) + kn\log n)$ time, where $T(n,k)$ is the running time for computing the order-k Voronoi diagram of P.

Case 3: The remaining cases. Here, we apply parametric search. We first apply the procedures that deal with Case 1 and Case 2. Let r_U be the minimum radius of the results of the two procedures. To handle Case 3, we give a decision algorithm that returns "yes" with input $0 < r \leq r_U$ if and only if there exist an assignment f of weights and a point $c \in \mathbb{R}^2$ such that $d(p,c)/f(p) \leq r$ for all points $p \in P$ and $d(q,c) = r$ for some point in $q \in P$. In other words, the decision algorithm returns "yes" if and only if there is an assignment with covering radius r one of whose determinators is assigned weight 1. For a radius r, we call such a point c a *center* with radius r.

The following lemma enables us to apply parametric search. A proof of this lemma can be found in the full version of this paper.

▶ **Lemma 13.** *If an optimal solution belongs to Case 3, then the decision problem for any input r with $r^* \leq r \leq r_U$ returns "yes."*

Decision Algorithm for Case 3. Given a covering radius r, the decision algorithm first computes the intersection I of the disks $D(p,r)$ for all points $p \in P$, where $D(p,r)$ is the disk centered at p with radius r. If the answer for the decision problem is "yes", then a center with r lies in the intersection I. Moreover, a center with r lies on the boundary of I by the definition.

Thus the decision algorithm searches the boundary of I and checks whether there exists a center on the boundary of I. Here, we follow the framework of the algorithm in Section 3. That is, we consider $O(kn)$ circles $C_{p,w}(r)$ for all points $p \in P$ and all weights $w \in W \cup \{1\}$, where $C_{p,w}(r)$ is the circle centered at p with radius rw. Then we compute $O(kn)$ intersection points of each of the circles with the boundary of I and sort them along the boundary of I in $O(kn \log n)$ time. We apply the procedure in the proof of Lemma 4, which checks whether there exists a center with radius r lying on the boundary of I in $O(kn)$ time. Thus the decision algorithm takes $O(kn \log n)$ time.

Overall Algorithm for Case 3. As we did in the decision algorithm, we first compute the intersection $I(r^*)$ of the disks $D(p, r^*)$ for all points $p \in P$. Here, we are not given r^*. Instead of computing the intersection explicitly, we compute its combinatorial structure.

▶ **Lemma 14.** *The combinatorial structure of the intersection of the disks $D(p, r^*)$ for all points $p \in P$ can be computed in $O(n \log n + T(n) \log n)$ time, where $T(n)$ is the running time of the decision algorithm.*

Proof. As we increase the radius r from 0 to r_U, the combinatorial structure of the intersection of the disks $D(p, r)$ may change. We have two types of events where the combinatorial structure changes: an arc of a disk starts to appear in the structure or an existing arc of a disk disappears from the structure. At both types of events, such an arc becomes a point which is a degenerate arc. Moreover, this point is a vertex of the farthest-point Voronoi diagram of P.

To use this fact, we compute the farthest-point Voronoi diagram of P in $O(n \log n)$ time. Then for each vertex v of the diagram, we compute the Euclidean distance between v and its farthest point in P. There are $O(n)$ distances, and we sort them in the increasing order.

Then we apply binary search on the distances using the decision algorithm to find the smallest interval containing r^*. This can be done in $O(T(n) \log n)$ time.

Therefore, for any radius on the interval, the combinatorial structure of the intersection of the disks remains the same. ◀

Now we have the combinatorial structure of the intersection $I(r^*)$. As we did in the algorithm of Section 3, we sort the intersections of $O(kn)$ circles $C_{p,w}(r^*)$ for all point $p \in P$ and all weight $w \in W$ with the boundary of $I(r^*)$ without explicitly computing r^*. This can be done in $O(kn \log n + T(n) \log^2 n)$ time, where $T(n) = O(kn \log n)$ is the running time of the decision algorithm, in a way similar to Lemma 6. Then we find an optimal solution in a way similar to Lemma 7 in $O(kn)$ time if it belongs to Case 3. In total, Case 3 can be dealt in $O(kn \log^3 n)$ time using $O(kn)$ space.

Combining the three cases, we have the following theorem.

▶ **Theorem 15.** *Given a set P of n points in the plane and a multiset W of k weights smaller than or equal to 1 with $k \leq n$, we can compute an assignment f of weights that minimizes $r(f)$ in $O(k^5 n \log^4 k + kn \log^3 n)$ time using $O(kn)$ space.*

5 Concluding Remarks

We would like to mention that the approach in this paper also works under any convex distance function, including the L_p metric for $p \geq 1$. For the L_1 or the L_∞ metric, the optimal weighted center is not necessarily unique though. If the weight assigned to an input point p is subtracted from the distance between p and a point of the plane, the running times of the algorithms can be improved by an $O(\log n)$ factor.

References

1 Pankaj K. Agarwal, Mark de Berg, Jiří Matoušek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM Journal on Computing*, 27(3):654–667, 1998.

2 Pankaj K. Agarwal and Micha Sharir. *Computer Science Today: Recent Trends and Developments*, chapter Algorithmic techniques for geometric optimization, pages 234–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

3 Behrooz Alizadeh and Rainer E. Burkard. The inverse 1-center location problem on a tree. Technical Report 2009-03, Graz University of Technology, 2009.

4 Luis Barba, Otfried Cheong, Jean-Lou De Carufel, Michael Gene Dobbins, Rudolf Fleischer, Akitoshi Kawamura, Matias Korman, Yoshio Okamoto, János Pach, Yuan Tang, Takeshi Tokuyama, Sander Verdonschot, and Tianhao Wang. Weight balancing on boundaries and skeletons. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry (SoCG 2014)*, pages 436–443, 2014.

5 Oded Berman, Divinagracia I. Ingco, and Amedeo Odoni. Improving the location of minimax facilities through network modification. *Networks*, 24(1):31–41, 1994.

6 Mao-Cheong Cai, X. G. Yang, and J. Z. Zhang. The complexity analysis of the inverse center location problem. *Journal of Global Optimization*, 15(2):213–218, 1999.

7 Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

8 Martin Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM Journal on Computing*, 13(1):31–45, 1984.

9 Der-Tsai Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE Trans. Computers*, 31(6):478–487, 1982.

10 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

11 Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

12 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

13 Nimrod Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(6):605–610, 1989.

14 Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry (SoCG 1999)*, pages 390–399, 1999.

15 Jianzhong Zhang, Zhenhong Liu, and Zhongfan Ma. Some reverse location problems. *European Journal of Operational Research*, 124(1):77–88, 2000.

A Near-Optimal Algorithm for Finding an Optimal Shortcut of a Tree*

Eunjin Oh[1] **and Hee-Kap Ahn**[2]

1 **Department of Computer Science and Engineering, POSTECH,**
77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea
jin9082@postech.ac.kr

2 **Department of Computer Science and Engineering, POSTECH,**
77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea
heekap@postech.ac.kr

Abstract

We consider the problem of finding a shortcut connecting two vertices of a graph that minimizes the diameter of the resulting graph. We present an $O(n^2 \log^3 n)$-time algorithm using linear space for the case that the input graph is a tree consisting of n vertices. Additionally, we present an $O(n^2 \log^3 n)$-time algorithm using linear space for a continuous version of this problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Network Augmentation, Shortcuts, Diameter, Trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.59

1 Introduction

Consider a graph $G = (V, E)$ with n vertices whose edges are assigned positive weights. The *length* of a path in G is the sum of the weights of the edges in the path. For any two vertices u and v in V, the *distance* between u and v in G is the length of the shortest path in G connecting u and v. We denote the distance between u and v by $d_G(u, v)$, or simply by $d(u, v)$ when it is understood in the context. The *diameter* of G is the maximum distance between any two vertices in G, that is, $\max_{u,v \in V} d_G(u, v)$.

In this paper, we consider *the diameter-optimal augmentation problem* for trees. We are to find a *shortcut* st that connects two vertices s and t of an input tree G and minimizes the diameter of $G + st = (V, E \cup \{st\})$. Here, the weight of a shortcut connecting two vertices is given in advance as an input. We assume that the weight of any shortcut can be determined in $O(1)$ time.

Related Work. For the case that the input graph is a path embedded in a metric space, Große et al. [6] gave an $O(n \log^3 n)$-time algorithm that computes an optimal shortcut by using parametric search [10]. Their decision algorithm uses the fact that for a fixed $s \in V$, the diameter of $T + st$ is the maximum of four monotone functions of vertex t moving along the path from s to an endpoint of the path.

Very recently, De Carufel et al. [4] studied the continuous version of this problem in which the input graph G is embedded in the Euclidean plane, the weight of an edge or a

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 59; pp. 59:1–59:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

shortcut is the Euclidean distance between its endpoints, and shortcuts are allowed to have their endpoints on any points of edges of G. Their goal is, by adding a shortcut to G, to minimize the *continuous diameter* of the resulting graph which is the maximum distance between any two *points* lying on edges and vertices. In this setting, they gave a linear-time algorithm for this problem when G is a path. They also studied the case that G is a cycle and showed that a single shortcut cannot decrease the continuous diameter. They gave a linear-time algorithm for computing an optimal pair of shortcuts when G is a convex cycle.

In graphs which are not necessarily embedded in a metric space, a similar problem was also studied. For the case that we are allowed to add more than one shortcut to an input graph, the problem of finding a diameter-optimal set of shortcuts becomes NP-hard [11]. For a path and a cycle, an upper and a lower bounds of the achievable diameter of the resulting graph were given by [2, 3]. For an outerplanar graph, Ishii [7] gave a constant-factor approximation algorithm for this problem.

Another variant of this problem is to find a shortcut that minimizes the dilation of a graph, and it was considered for graphs embedded in the Euclidean space [1, 5, 8] and in a metric space [9].

Our Results. We consider the problem of computing an optimal (discrete) shortcut of a tree and present an $O(n^2 \log^3 n)$-time algorithm for the problem. This problem is a generalization of the metric shortcut problem for paths [6] in the sense that the input graph is a tree and it is not necessarily embedded in a metric space. To achieve this running time, we traverse the tree in an Euler tour and efficiently compute the diameter of the tree with a shortcut using a data structure. As a byproduct, we present an $O(n \log n)$-time algorithm for computing the diameter of an edge-weighted graph containing exactly one cycle.

In addition, we consider its continuous version and present an algorithm for computing an optimal continuous shortcut when the input graph is a tree. Again the input graph is not necessarily embedded in a metric space, and therefore this problem is a generalization of the continuous Euclidean diameter problem [4]. Our algorithm takes $O(n^2 \log^3 n)$ time using $O(n)$ space.

A Lower Bound for Computation. In the discrete version, consider a path with n vertices whose diameter is $\lambda \in \mathbb{R}$. There are at most $\binom{n}{2}$ shortcuts[1] in the input graph. Consider the case that only one shortcut has a sufficiently small weight $\epsilon > 0$ and the other shortcuts have the weight larger than λ. In this case, the shortcuts with larger weights do not decrease the diameter while the shortcut with weight ϵ decreases the diameter. Thus an optimal shortcut is the one with weight ϵ. However, there is no way to find the optimal shortcut unless we check the weights of all shortcuts. So it is inevitable to spend $\Omega(n^2)$ time to compute an optimal shortcut even in a path. A similar argument also holds in the continuous diameter problem.

Note that the algorithms in [4] and [6] assume that an input graph is embedded in a metric space. This allows them to achieve a near-linear or a linear running time.

2 Computing the Diameter of a Unicyclic Graph

We say a graph is *unicyclic* if the graph contains exactly one cycle. Consider a unicyclic graph G, where each edge e is assigned a positive weight $\ell(e)$. Let γ denote the unique cycle of G.

[1] Precisely speaking, there are $\binom{n}{2} - (n-1)$ shortcuts for a tree.

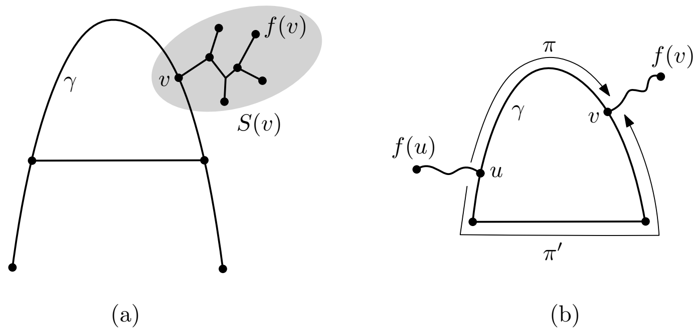


Figure 1 (a) The gray region indicates the tree $S(v)$ rooted at v in $G^{\setminus\gamma}$. (b) For any two vertices u and v on γ, the clockwise vertex-weighted distance is $\delta_{cw}(u,v) = w(u) + \sum_{e\in\pi} \ell(e) + w(v)$ and the counterclockwise vertex-weighted distance is $\delta_{ccw}(u,v) = w(u) + \sum_{e\in\pi'} \ell(e) + w(v)$.

In this section, we present an $O(n \log n)$-time algorithm for computing the diameter of a unicyclic graph. Recall that the diameter of a graph is the maximum distance between any two vertices of the graph.

We can compute an optimal shortcut in $O(n^3 \log n)$ time by applying this algorithm for every possible shortcut. In Section 3.1 and 3.2, we give a faster algorithm for computing an optimal shortcut. Although the faster algorithm does not directly use the algorithm in this section, we first introduce this algorithm because the faster algorithm uses a technique similar to the one used in this algorithm.

2.1 Computing the Weights of Cycle Vertices

Consider the subgraph $G^{\setminus\gamma}$ of G obtained by removing the edges on γ. Each connected component of $G^{\setminus\gamma}$ forms a tree and contains exactly one vertex on γ. We treat each connected component as a tree rooted at the vertex on γ. For each vertex v on γ, we use $S(v)$ to denote the connected component rooted at v in $G^{\setminus\gamma}$. See Figure 1(a). Note that $S(v)$ may consist of only one vertex v. In addition, we use $f(v)$ to denote the vertex in $S(v)$ farthest from v. The vertex $f(v)$ for every vertex v on γ can be computed in $O(n)$ total time.

Now we annotate a weight $w(v)$ to each vertex v on γ, where $w(v) = d_G(v, f(v))$. If $S(v)$ consists of v alone, we set $d_G(v, f(v)) = 0$.

Vertex-Weighted Distances for Cycle Vertices. There are two simple paths connecting two vertices on γ. To make the description easier, we assign an arbitrary orientation to γ and treat it as a *clockwise orientation*. Then the opposite orientation is a *counterclockwise orientation*.

We define three *vertex-weighted* distances from u to v for any two distinct vertices u and v in γ as follows. (For an illustration, see Figure 1(b).)

1. The clockwise vertex-weighted distance : $\delta_{cw}(u,v) = w(u) + \sum_{e\in\pi} \ell(e) + w(v)$, where π is the simple path from u to v in the clockwise orientation along γ.
2. The counterclockwise vertex-weighted distance : $\delta_{ccw}(u,v) = w(u) + \sum_{e\in\pi'} \ell(e) + w(v)$, where π' is the simple path from u to v in the counterclockwise orientation along γ.
3. The vertex-weighted distance : $\delta(u,v) = \min\{\delta_{cw}(u,v), \delta_{ccw}(u,v)\}$.

For any vertex u in γ, let $\delta(u,u) = \delta_{cw}(u,u) = \delta_{ccw}(u,u) = 0$.

2.2 Computing the Diameter of a Unicyclic Graph

In this section we give an $O(n \log n)$-time algorithm to compute the diameter of G. A diametral pair of a graph is a pair (x, y) of vertices of the graph such that the distance between x and y is the same as the diameter of the graph. There are two possible cases for a diametral pair (x, y) of the unicyclic graph G:

1. Both x and y are contained in $S(v)$ for some vertex v on γ.
2. Vertex x is contained in $S(u)$ and vertex y is contained in $S(v)$ for two distinct vertices u and v on γ.

For the first case, a simple path connecting x and y is unique. And it is contained in $S(v)$. Therefore we can handle this case by computing the diameter and the diametral pair of $S(v)$. For all vertices v on γ, the diameter of $S(v)$ can be computed in total linear time.

For the second case, there are two simple paths connecting x and y, one through the clockwise path from u to v and one through the counterclockwise path from u to v, and we observe that $d_G(x, y) = \delta(u, v)$. Therefore, it suffices to find the *vertex-weighted diameter* of γ, that is, $\max_{u,v \in \gamma} \delta(u, v)$.

To handle the second case, we first compute the weight $w(v)$ of every vertex v in γ and annotate it to v, which takes $O(n)$ time in total for all vertices in γ. To compute the vertex-weighted diameter of γ, we find the maximum vertex-weighted distance $\lambda_v = \max_{u \in \gamma} \delta(v, u)$ for each vertex v in γ. By definition, the vertex-weighted diameter of γ is the maximum of λ_v over all vertices v in γ.

Computing λ_v for a vertex v in γ. Let $\lambda > 0$ be any real number. For a vertex v in γ, let $U_\lambda(v)$ be the set of all vertices u in γ with $\delta_{cw}(v, u) > \lambda$. Note that the vertices in $U_\lambda(v)$ are not necessarily consecutive along γ. Let $g_\lambda(v)$ be the vertex of γ with the largest counterclockwise distance $\delta_{ccw}(v, g_\lambda(v))$ among all vertices in $U_\lambda(v)$. If $U_\lambda(v) = \emptyset$, we let $g_\lambda(v)$ be v. Then the following observation holds as $\delta_{cw}(v, u) > \lambda$ for any $u \in U_\lambda(v)$.

▶ **Observation 1.** *We have* $\delta_{ccw}(v, g_\lambda(v)) \leq \lambda$ *if and only if* $\lambda_v \leq \lambda$.

To use this observation, we map each vertex u in γ to the point $p_v(u)$ with x-coordinate $\delta_{cw}(v, u)$ and y-coordinate $\delta_{ccw}(v, u)$ in the xy-plane. Let P_v be the set of points $p_v(u)$ for all vertices u in γ. For an illustration, see Figure 2(a). To check if $\lambda_v \leq \lambda$, it is sufficient to find the point with largest y-coordinate among all points of P_v lying to the right of the vertical line $x = \lambda$. By definition, the point is $p_v(g_\lambda(v))$ if it exists. Otherwise, $g_\lambda(v) = v$.

We use a 1-dimensional range tree on (the x-coordinates of) P_v to compute $g_\lambda(v)$ efficiently for any $\lambda > 0$. To be specific, we construct the range tree on P_v with respect to the x-coordinates of the points of P_v. Each node in the range tree corresponds to an interval on $\mathbb{R}$. For each node z, we store the largest y-coordinate of the points in P_v whose x-coordinate is in the interval corresponding to z. We denote the value stored at node z by y_z. Once we have this range tree, we can check in $O(\log n)$ time whether $\lambda_v \leq \lambda$.

In addition, we compute λ_v using the range tree on P_v. Starting from the root of the range tree, we traverse the range tree to some leaf. Suppose that we reach an internal node z in the range tree. The interval corresponding to z is subdivided into two subintervals corresponding to its two children. Let $\lambda \in \mathbb{R}$ be a value separating the two subintervals. We check whether $\lambda_v \leq \lambda$ or not in constant time by using the value, y_z, stored in z: $\lambda_v \leq \lambda$ if and only if $y_z \leq \lambda$. If $\lambda_v \leq \lambda$, we move to its left child. Otherwise, we move to its right child. In each node, we spend $O(1)$ time. Thus in $O(\log n)$ time, we can compute two consecutive points p_1 and p_2 in P_v such that $\lambda_1 \leq \lambda_v \leq \lambda_2$, where λ_1 and λ_2 are the x-coordinates of p_1 and p_2, respectively. See Figure 2(a).

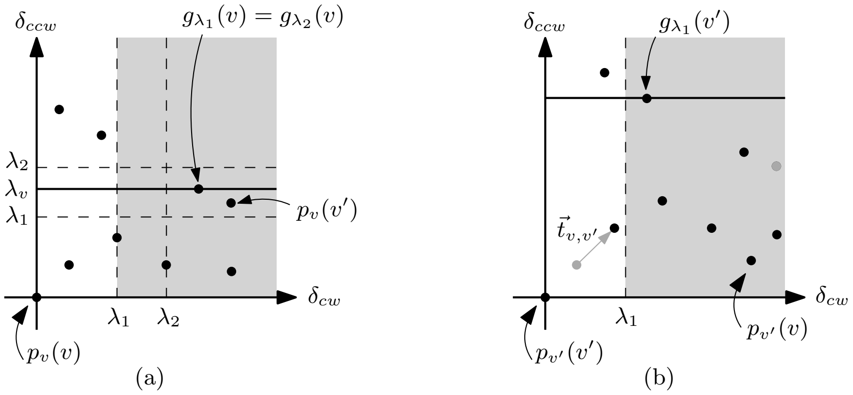


Figure 2 (a) The point set P_v obtained from the mapping of $\delta_{cw}(v,u)$ and $\delta_{ccw}(v,u)$ for each u in γ. We have $\lambda_1 < \lambda_v < \lambda_2$ because $\lambda_1 < \delta_{ccw}(v, g_{\lambda_1}(v))$ and $\delta_{ccw}(v, g_{\lambda_2}(v)) < \lambda_2$. Moreover, we have $\lambda_v = \delta_{ccw}(v, g_{\lambda_1}(v))$. (b) The point set $P_{v'}$ is a translated copy of P_v by $\vec{t}_{v,v'}$, with exceptions of $p_{v'}(v') \in P_{v'}$ and $p_v(v') \in P_v$.

Here, we have $g_{\lambda_1}(v) = g_{\lambda_2}(v)$. Moreover, λ_v is the y-coordinate of $g_{\lambda_1}(v)$. Therefore, we can compute λ_v in $O(\log n)$ time once we have the range tree on P_v.

▶ **Lemma 2.** *Once we have the 1-dimensional range tree on the x-coordinates of P_v for a vertex v in γ, the distance λ_v can be computed in $O(\log n)$ time.*

Computing λ_v for all vertices v in γ. Instead of computing the 1-dimensional range trees of P_v for all vertices v in γ repeatedly, we consider the vertices one by one in clockwise order along γ and make use of the 1-dimensional range tree on P_v in computing the 1-dimensional range tree on $P_{v'}$, where v' is the clockwise neighbor of v along γ.

Suppose that we have the 1-dimensional range tree on P_v. We show how to compute $P_{v'}$ using P_v for the clockwise neighbor v' of v. Consider a vertex $u \neq v, v'$ in γ. Recall that $p_v(u)$ is the point in the xy-plane with x-coordinate $\delta_{cw}(v,u)$ and y-coordinate $\delta_{ccw}(v,u)$. Let (x,y) be the x- and y-coordinates of $p_v(u)$. Then we have

$$p_{v'}(u) = (x - w(v) + w(v') - \ell(vv'),\ y - w(v) + w(v') + \ell(vv')).$$

Note that $w(v), w(v')$ and $\ell(vv')$ are independent to vertex u. This means that the point set $P_{v'} \setminus \{p_{v'}(v), p_{v'}(v')\}$ is a translated copy of the point set $P_v \setminus \{p_v(v), p_v(v')\}$ by the translation vector $\vec{t}_{v,v'} = (-w(v) + w(v') - \ell(vv'), -w(v) + w(v') + \ell(vv'))$. See Figure 2(b).

Thus, to compute $P_{v'}$, we remove $p_v(v')$ and $p_v(v)$ from P_v, translate the remaining points in P_v by $\vec{t}_{v,v'}$, and add $p_{v'}(v')$ and $p_{v'}(v)$ to the point set. The resulting point set is exactly $P_{v'}$. Both removing and adding the two points can be done in $O(\log n)$ time. The translation can be done in $O(1)$ time by moving the x-axis and the y-axis by $-\vec{t}_{v,v'}$, instead of translating the points.

The 1-dimensional range tree on $P_{v'}$ remains the same, except for the nodes for $p_{v'}(v'), p_{v'}(v)$ and the value each node stores. We first remove two points $p_{v'}(v'), p_{v'}(v)$ from the range tree on P_v. Then we update the values stored in the nodes of the range tree. These values increase or decrease by the same amount in the range tree of $P_{v'}$, and therefore we simply maintain a global value for the range tree, the *offset* of the values, instead of updating each value. We apply this offset to each value when it is used. After updating the offset, we add two points for v and v' to the 1-dimensional range tree in $O(\log n)$ time. The resulting tree is the range tree on $P_{v'}$.

Now, we have the 1-dimensional range tree on $P_{v'}$. By Lemma 2, we can compute $\lambda_{v'}$ in $O(\log n)$ time. By applying this procedure for all vertices one by one along γ, we have the following lemma.

▶ **Lemma 3.** *The distances λ_v for all vertices v in γ can be computed in $O(n \log n)$ time.*

Therefore, the following theorem holds.

▶ **Theorem 4.** *The diameter of a unicyclic graph G can be computed in $O(n \log n)$ time.*

3 The Diameter-Optimal Augmentation for a Tree

Let $T = (V, E)$ be an input tree, where each edge $e \in E$ is assigned a positive weight $\ell(e)$. In this section, we find a shortcut st connecting two vertices of T that minimizes the diameter of $T + st = (V, E \cup \{st\})$. We use λ^* to denote the diameter of $T + st$ for an optimal shortcut st. We assume that the weight of any shortcut can be determined in $O(1)$ time.

3.1 A Decision Algorithm for Computing a Shortcut

We first consider a decision problem and give an algorithm to decide whether $\lambda \geq \lambda^*$ or not in $O(n^2 \log n)$ time for a real number $\lambda > 0$. This decision algorithm is used as a subprocedure in the overall algorithm, which we will describe in Section 3.2.

Basically, we consider all possible shortcuts st and check whether the diameter of the unicyclic graph $T + st$ is at least λ. We spend $O(\log n)$ time for each shortcut.

3.1.1 The Euler Tour of the Input Tree

We fix a vertex s of T and consider it as an endpoint of a shortcut. In addition, we treat it as the root of T. An Euler tour of a graph is a path traversing each edge exactly once. For trees, we assume that each edge is bidirectional, so the Euler tour of a tree is the path through the tree that begins and ends at the root, traversing each edge exactly twice. In the Euler tour, we visit each vertex t of T exactly $\mathsf{deg}(t)$ times, where $\mathsf{deg}(t)$ is the degree of t in T.

Whenever we visit a vertex t in T, we compute the cycle of $T + st$ together with the weight $w(v)$ for each vertex v of the cycle, and check whether the diameter of $T + st$ is at least λ. Let $\gamma(s, t)$ be the cycle of $T + st$ for a vertex t in T.

When we move from a vertex t to one of its children t' in the Euler tour, the cycle changes locally. To be specific, assume that we already have $\gamma(s, t)$ and the weight $w(v)$ of each vertex v of $\gamma(s, t)$. As depicted in Figure 3, we construct $\gamma(s, t')$ by appending tt' and replacing st with st' to $\gamma(s, t)$. Then the weight of t may change and we need to compute $w(t')$ with respect to $\gamma(s, t')$, while the other vertices of $\gamma(s, t')$ have their weights unchanged. The weights, $w(t)$ and $w(t')$ with respect to $\gamma(s, t')$, can be computed in $O(1)$ time once we compute two distance values for every vertex in linear time in advance as follows.

For a vertex v of T rooted at s, let $h_1(v)$ be the distance between v and the descendant of v farthest from v. Let $h_2(v)$ denote the second largest value among values $h_1(u) + \ell(vu)$ for all children u of v. If v has exactly one child, we let $h_2(v) = 0$. We consider the vertices from the leaves one by one with respect to their depths, and compute $h_1(v)$ for all vertices v in total linear time. Then, we compute the distance $h_2(v)$ in $O(\deg(v))$ time by checking the values $h_1(u)$ for all children u of v.

The weight $w(t')$ with respect to $\gamma(s, t')$ is exactly $h_1(t')$, which can be found in constant time. For the weight $w(t)$ with respect to $\gamma(s, t')$, we have two cases: $w(t) = h_1(t)$ if

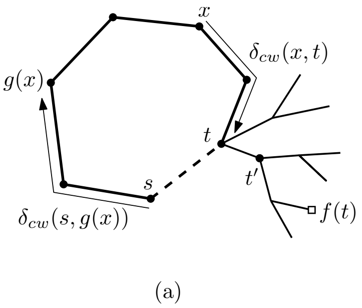


Figure 3 When we move from vertex t to its neighbor vertex t', tt' is appended to the cycle $\gamma(s,t)$ and the weight $w(t) = d(t, f(t))$ of t with respect to the new cycle $\gamma(s,t')$ changes accordingly. (a) $h(x) = \delta_{cw}(x,t) + \delta_{cw}(s, g(x))$ with respect to $\gamma(s,t)$. (b) $h(x) = \delta_{cw}(x,t') + \delta_{cw}(s, g(x))$ with respect to $\gamma(s,t')$.

$h_1(t) > h_1(t') + \ell(tt')$, and $w(t) = h_2(t)$ otherwise. In either case, we can compute $w(t)$ in constant time.

When we move from a vertex t to its parent t'', we can obtain $\gamma(s,t'')$ and the weights of the vertices on $\gamma(s,t'')$ in constant time analogously.

▶ **Lemma 5.** *The cycle $\gamma(s,t')$ and the weights $w(v)$ of vertices v of $\gamma(s,t')$ can be updated in $O(1)$ time for a traversal of edge tt' in the Euler tour once we compute $h_1(v)$ and $h_2(v)$ for every vertex v of T in linear time.*

3.1.2 Deciding the diameter of $T + st$ for λ

Without loss of generality, we assume that t is the counterclockwise neighbor of s in $\gamma(s,t)$. We show how to decide whether the diameter of $T + st$ is at least λ in $O(\log n)$ time. To do this efficiently, we maintain a list of distance values sorted in the increasing order.

The Sorted List $\mathcal{H}$ of distance values. For each vertex x in $\gamma(s,t)$, let $g(x)$ denote the vertex with largest $\delta_{cw}(s,y)$ among vertices y lying in between s and x in clockwise direction from s and satisfying $\delta_{ccw}(x,y) > \lambda$. If no such vertex exists, we let $g(x)$ be *null*. Note that we have $\delta_{cw}(x, g(x)) = w(x) + d_T(x,t) + \ell(st) + d_T(s, g(x)) + w(g(x))$. Moreover, if $\delta_{cw}(x, g(x))$ is at most λ or $g(x)$ is null, then we have $\delta(x,y) \leq \lambda$ for every vertex y in between s and x in clockwise direction from s. Let $h(x) = w(x) + d_T(x,t) + d_T(s, g(x)) + w(g(x))$ for a vertex x in $\gamma(s,t)$ such that $g(x)$ is not null. We maintain $h(x)$ for all vertices x in $\gamma(s,t)$ such that $g(x)$ is not null and sort them in decreasing order. We denote this sorted list by $\mathcal{H}$.

Once we have the sorted list $\mathcal{H}$, we can check whether the diameter of $T + st$ is at least λ in constant time. We choose the first element of $\mathcal{H}$, say $h(x)$. Then the answer for the decision problem is "yes" if and only if $\ell(st) + h(x) \leq \lambda$. This can be checked in constant time.

Now we show how to update the sorted list $\mathcal{H}$ in $O(\log n)$ time when we move from a vertex t to a vertex t' in the Euler tour in three phases: (1) update $\mathcal{H}$, except for $h(t')$ and $h(t)$, (2) remove $h(t)$ from $\mathcal{H}$, and (3) compute $h(t')$ and $h(t)$, and insert them to $\mathcal{H}$. We compute $h(t)$ again because the value $h(t)$ changes. Let $\mathcal{H}$ be the sorted list which is already computed for the vertex t. We are to update $\mathcal{H}$, that is, to construct the sorted list for t'. Recall that in $\gamma(s,t')$, only one vertex is added to or removed from $\gamma(s,t)$. Here, we consider the case that one vertex, which is t', is added to $\gamma(s,t)$. The other case is analogous to this case.

First observe that $h(x)$ increases by the same amount for all vertices x in $\gamma(s,t') \setminus \{t,t'\}$. Moreover, this amount is exactly $\ell(tt') - w(t) + w'(t')$, where $w(t)$ and $w'(t')$ denote the weights of t and t' in $\gamma(s,t)$ and $\gamma(s,t')$, respectively. This means that the order for the elements in $\mathcal{H}$ remains the same, except for $h(t)$ and $h(t')$. Thus, we can update $\mathcal{H}$, except for $h(t')$ and $h(t)$, in constant time by maintaining the offset.

For computing $h(t)$, we observe that $g(t)$ remains the same. However, since the weight of t changes, the value $h(t)$ changes accordingly. With the new weight $w(t)$, we can update $h(t)$ in constant time and insert it to $\mathcal{H}$ in $O(\log n)$ time.

The remaining procedure is to compute $h(t')$. This procedure is similar to the procedure for computing the diameter of a unicyclic graph in Section 2. To this end, we maintain a point set and its range tree in addition to the sorted list $\mathcal{H}$.

The Point Set P and Its Range Tree. We map each vertex x in $\gamma(s,t')$ to the point $p(x) \in \mathbb{R}^2$ with $p(x) = (\delta_{ccw}(t',x), \delta_{cw}(s,x))$. Let P denote the set of $p(x)$'s for every vertex x in $\gamma(s,t')$. We construct a 1-dimensional range tree on P with respect to the x-coordinates of the points in the set. This range tree can be updated in $O(\log n)$ time as we did in Section 2.

Once we have the range tree, we can compute $h(t')$ in $O(\log n)$ time as follows. We find the point with largest y-coordinate in P among all points of P lying to the right of the vertical line $x = \lambda$. Note that the point with largest y-coordinate is $g(t')$ by definition. Thus, $h(t')$ is the y-coordinate of $p(g(t'))$ since $h(t') = w(t') + d_T(t',t') + d_T(s,g(t')) + w(g(t'))$. By adding $h(t')$ to its proper position in $\mathcal{H}$, we have the sorted list $\mathcal{H}$ for t'.

This completes the procedure for checking if the diameter of $T + st$ is at least λ, which takes $O(\log n)$ time. With these arguments, the following lemma holds.

▶ **Lemma 6.** *Given a tree T and a real number $\lambda > 0$, we can decide in $O(n^2 \log n)$ time whether $\lambda^* \leq \lambda$ or not.*

3.2 An Algorithm for Computing the Optimal Shortcut

The optimal diameter λ^* is either the distance $d(u,v)$ of two vertices $u,v \in T$ or the sum of two distances $d(u,t)$ and $d(s,v)$ of $u,v,s,t \in T$ and the weight of shortcut st. We first apply binary search on the lengths of all paths in T and find an interval of distance as described in Section 3.2.1. Then we consider the second case in Section 3.2.2.

3.2.1 Binary Search on Distances of Two Vertices

We can compute the set $\mathcal{D}_1$ of distances of every two vertices in T in $O(n^2)$ time and apply binary search on the distances using the decision algorithm in Section 3.1. This procedure gives us an interval containing no value of $\mathcal{D}_1$ in its interior but containing λ^* in $O(n^2 \log^2 n)$ time. However, this procedure requires $\Omega(n^2)$ space.

We show how to do this in $O(n^2 \log^3 n)$ time using $O(n)$ space. We apply binary search in $O(\log n)$ rounds. In the first round, we do the following: for each vertex v of T, we compute the median of the distances of v to all other vertices in T. This gives us n medians in $O(n^2)$ time using $O(n)$ space. Then we sort these medians and apply binary search on them. Let η be the interval obtained from the binary search. Note that η contains λ^*.

In the next rounds, we repeat the following and refine the interval containing λ^*. For each vertex v of T, we consider the distances of v to all other vertices of T that lie in the interval η and choose the median of them. Then we apply binary search again on these medians and reduce η into the subinterval that contains λ^*.

In every round, for each vertex v of T, the number of distances of $\mathcal{D}_1$ lying in η decreases by a constant fraction. Therefore, after $O(\log n)$ rounds, the interval η contains no distance of $\mathcal{D}_1$ in its interior. This takes $O(n^2 \log^3 n)$ time using $O(n)$ space.

3.2.2 Computing the Diameter with the Optimal Shortcut

Now we consider the case that λ^* is the sum of two distances $d(u,t)$ and $d(s,v)$ of $u,v,s,t \in T$ and the weight of shortcut st. We observe that u and v are contained in two different subtrees of $T^{\setminus\gamma(s,t)}$ rooted at x and y on $\gamma(s,t)$, respectively, such that $y = g(x)$. Therefore, we have $\lambda^* = h(x) + \ell(st)$. Based on this observation, we consider $O(n^2)$ candidate values one of which is exactly λ^* and find λ^* among them. We use a technique similar to parametric search [10].

We simulate the decision algorithm with λ^* without explicitly knowing λ^*. Let $\mathcal{D}_2$ be the set of candidate values that we consider. Initially, $\mathcal{D}_2$ is empty. We fix a vertex s in T and traverse the vertices in an Euler tour of T as we did in the decision algorithm. Assume that we move from t to the next vertex t'. We compute the cycle $\gamma(s,t')$ and the weights of vertices in $\gamma(s,t')$ in constant time once we have two distance values, $h_1(v)$ and $h_2(v)$, for every vertex v of T by Lemma 5.

Then we update the sorted list $\mathcal{H}$ in three phases. In the first phase, we update the elements in $\mathcal{H}$, except for $h(t')$ and $h(t)$. In the second phase, we remove $h(t')$ from $\mathcal{H}$. These two phases are independent of input λ, so the first phase can be done even though we do not have the explicit value of λ^*.

In the third phase, we first update the element for $h(t)$. This update is independent of λ because we already have $g(t)$, thus it can be done in $O(\log n)$ time. Then we update the element for $h(t')$. To do this, we maintain a point set and its range tree. Since these structures are independent of input λ, we can update them as we did in the decision algorithm without the explicit value of λ^*. To compute $h(t')$, we find the point with largest y-coordinate in P among all points lying to the right of $x = \lambda$. This procedure is dependent of an input λ.

However, we already have the interval η from Section 3.2.1 that contains λ^* but does not contain any value of $\mathcal{D}_1$ in its interior. Moreover, the x-coordinate of each point in P is the distance between two vertices of T. Therefore, we can find the point with largest y-coordinate in P among all points lying to the right of $x = \lambda^*$ by choosing any distance $\lambda \in \eta$. The third phase of the update can be done in $O(\log n)$ time.

The next step in the decision algorithm is to check whether $\lambda \geq \ell(st') + h(x)$ or not for the first element $h(x)$ of $\mathcal{H}$. This means that the value $\ell(st') + h(x)$ is a *critical* point in a sense that the answer for the decision problem with input λ depends on whether λ is at least this value or not. Thus, we add $\ell(st') + h(x)$ to $\mathcal{D}_2$.

We do this for all vertices s in T, and then we have $O(n^2)$ critical points one of which is exactly λ^*. The running time of this procedure is the same as the decision algorithm, which is $O(n^2 \log n)$. Then we apply binary search on $\mathcal{D}_2$ and we compute λ^*.

Since we have $O(n^2)$ distances, this requires $\Omega(n^2)$ space. Instead of computing the distances in $\mathcal{D}_2$ simultaneously, we apply binary search in $O(\log n)$ rounds as we did in Section 3.2.1. Then we can compute λ^* in $O(n^2 \log^3 n)$ time using $O(n)$ space.

▶ **Theorem 7.** *Given a tree T, an optimal shortcut can be computed in $O(n^2 \log^3 n)$ time using $O(n)$ space.*

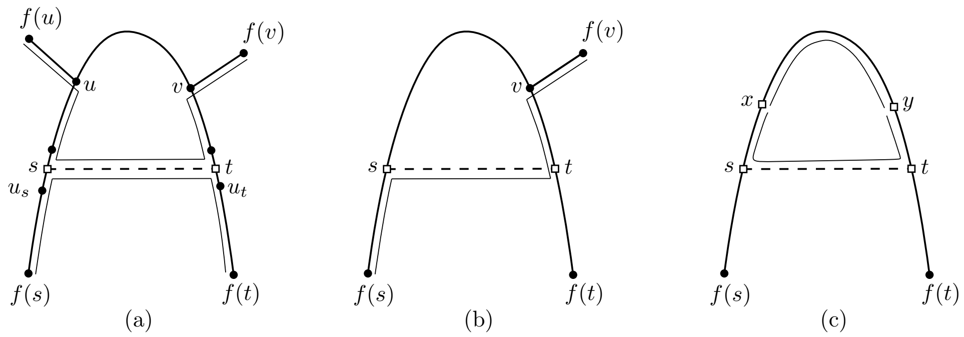


Figure 4 The points marked with disks are vertices of T while the points marked with squares are not necessarily vertices of T. (a), (b) Vertex-vertex pairs for a continuous diameter of a unicycle graph $T + st$. (c) A point-point pair (x, y).

4 The Continuous Diameter-Optimal Augmentation for a Tree

In this section, we consider a continuous version of the problem. We are given a tree $T = (V, E)$ with positive edge-weight $\ell(e)$ for each edge $e \in E$. In this problem, a subedge $e' \subset e$ also has its weights. For a point x in $e = ab$, the weight of the subedge ax is represented as an algebraic function with variable x, which is given as an input. Similarly, the weight of the subedge bx is given as an input. Since every edge has such two functions, we have $2n$ algebraic functions in total. With this weight, the length of the path between any two *points* of T is defined.

In addition, for two points $p \in e$ and $p' \in e'$, the weight of the shortcut pp' is also given as an input. We assume that for any two edges e and e', the weight of a shortcut pp' can be represented as an algebraic function with two variables $p \in e$ and $p' \in e'$. Since every pair of edges has such a function, we have at most $\binom{n}{2}$ algebraic functions representing the weight of a shortcut. In the following, we assume that we can compute the minimum of the upper envelope of any two algebraic functions with constant degree in constant time.

The continuous diameter is the maximum distance between any two points of T. Note that if (p, p') is a diametral pair for two points $p, p' \notin V$ then there are two simple paths connecting p and p' with the same length. The goal of this problem is to find two points s, t of T such that the continuous diameter of $T + st = (V, E \cup st)$ is minimized.

4.1 Characterization of the Continuous Optimal Shortcut

Let st be a continuous shortcut. Let $\gamma(s, t)$ be the unique cycle of $T + st$. For any vertex v on $\gamma(s, t)$, let $f(v)$ be the vertex of $S(v)$ farthest from v, where $S(v)$ is the connected component of $T^{\setminus \gamma(s,t)}$ containing v. Then there are two possible cases for a continuous diametral pair of the unicyclic graph $T + st$: (For an illustration, see Figure 4.)

- **vertex-vertex pair:** $(f(u), f(v))$ for two vertices $u, v \in \gamma(s, t)$ including s and t, and
- **point-point pair:** (x, y) for two points $x, y \in \gamma(s, t) \setminus V$.

For a point-point pair (x, y), there are two simple paths connecting x and y. Moreover, the two paths have the same length, which is half of the length of the cycle $\gamma(s, t)$.

4.2 A Decision Algorithm

We consider every pair (e_s, e_t) of edges of T and check whether there exists a shortcut st with $s \in e_s$ and $t \in e_t$ such that the diameter of $T + st$ is at most μ for a real number μ. If such a shortcut exists, we say that (e_s, e_t) is *feasible*. We say that a shortcut st is *in* (e_s, e_t) if $s \in e_s$ and $t \in e_t$. To solve the decision problem efficiently, we fix an edge e_s and treat an endpoint of e_s as the root of the tree. Then we compute an Euler tour of T from the root and visit every edge twice along the tour. Whenever we visit an edge e_t, we check whether (e_s, e_t) is feasible.

Checking Whether an Edge Pair is Feasible. Let u_s and u_t be the endpoints of e_s and e_t, respectively, that do not lie on the cycle $\gamma(s,t)$ for some shortcut st in (e_s, e_t). Assume that u_s is the clockwise neighbor of u_t in $T + u_s u_t$. See Figure 4(a).

Let $f(v)$ be the vertex of $S(v)$ farthest from v, where $S(v)$ is the connected component of $T^{\setminus\gamma(u_s,u_t)}$ containing v for a vertex v on $\gamma(u_s, u_t)$. The weight $w(v)$ of v on the cycle is the distance between $f(v)$ and v.

Assume that we have the followings:

- $d_T(u_s, u_t)$, and
- for every vertex v in $\gamma(u_s, u_t)$, the value $h(v)$ which is the maximum $w(u) + d_T(u, u_s) + d_T(u_t, v) + w(v)$ over all vertices u lying in the path between u_s and v in clockwise order from v on T with $d_T(u, v) + w(v) + w(u) > \mu$.

Then we can check whether (e_s, e_t) is feasible or not in constant time as follows. Let $f(s,t)$ be the weight of the shortcut st in (e_s, e_t), which is an algebraic function with variables s and t. Let $f_s(s)$ be the weight of the subedge $u_s s$, which is an algebraic function. Similarly, let $f_t(t)$ be the weight of the subedge $u_t t$. The continuous diameter of the cycle $\gamma(s,t)$ is $(d_T(u_s, u_t) - f_s(s) - f_t(t) + f(s,t))/2$, which is half of the length of the cycle and the diameter is an algebraic function with variables s and t.

For a vertex-vertex diametral pair, we consider $h(v)$ value of a vertex v in $\gamma(u_s, u_t)$. We find the maximum of $h(v)$'s over all vertices v in $\gamma(u_s, u_t)$. Once $h(v)$'s are sorted in decreasing order, we can choose the maximum h in constant time. If $h - f_s(s) - f_t(t) + f(s,t) \leq \mu$ for some shortcut st in (e_s, e_t), then the diameter is at most μ if a diametral pair is a vertex-vertex pair.

Thus, we check whether there is a shortcut st such that the maximum of $(d_T(u_s, u_t) - f_s(s) - f_t(t) + f(s,t))/2$ and $h - f_s(s) - f_t(t) + f(s,t)$ is at most μ. If so, we conclude that st is feasible. Otherwise, st is not feasible. With the argument in this section, we have the following lemma.

▶ **Lemma 8.** *Once we have $d_T(u_s, u_t)$ and $h(v)$ for every vertex v in $\gamma(u_s, u_t)$, we can check whether (e_s, e_t) is feasible or not in constant time.*

The distance $d_T(u_s, u_t)$ can be updated in constant time if we consider the edge e_t along the Euler tour. For the values $h(v)$'s, we can use the algorithm in Section 3.1. The algorithm in Section 3.1 computes exactly these values by maintaining a point set and a range tree. Therefore, we have the following.

▶ **Lemma 9.** *Given a tree T and a real number $\mu > 0$, we can decide in $O(n^2 \log n)$ time whether μ is at most the optimal solution or not.*

4.3 An Overall Algorithm

Let $\mathcal{D}_1$ be the set of distances between every pair of vertices in T. As we did in the algorithm for the discrete version, we compute the interval μ containing the optimal solution μ^* but containing no value of $\mathcal{D}_1$ in $O(n^2 \log^3 n)$ time.

Then we simulate the decision algorithm in Section 4.2 with an optimal solution. This can be done as follows without explicitly knowing an optimal solution. Since we have the interval μ, we can apply the procedures in Section 4.2, except for the last one that compares the maximum of the two algebraic functions with an input. Instead of applying the last procedure, we compute the maximum and put it into the set $\mathcal{D}_2$, which is set to be empty initially. Then one of the values in $\mathcal{D}_1 \cup \mathcal{D}_2$ is the optimal solution. Thus, we again apply binary search on the set $\mathcal{D}_1 \cup \mathcal{D}_2$. This is similar to the algorithm for the discrete version. This algorithm can also be analyzed analogously. Thus, we have the following theorem.

▶ **Theorem 10.** *Given a tree T, the continuous optimal shortcut can be computed in $O(n^2 \log^3 n)$ time using $O(n)$ space.*

References

1 Hee-Kap Ahn, Mohammad Farshi, Christian Knauser, Michiel Smid, and Yajun Wang. Dilation-optimal edge deletion in polygonal cycles. *International Journal of Computational Geometry & Applications*, 20(1):69–87, 2010.

2 Noga Alon, Andras Gyarfas, and Miklos Ruszinko. Decreasing the diameter of bounded degree graphs. *Journal of Graph Theory*, 35(3):161–172, 2000.

3 F. R. K. Chung and M. R. Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984.

4 Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, and Michiel Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. In *Proceedings of 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, pages 27:1–27:14, 2016.

5 Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38(1):226–240, 2008.

6 Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel Smid, and Fabian Stehn. Fast algorithms for diameter-optimally augmenting paths. In *Proceedings of Automata, Languages, and Programming: 42nd International Colloquium (ICALP 2015)*, pages 678–688, 2015.

7 Toshimasa Ishii. Diameter bounds for altered graphs. *Journal of Graph Theory*, 74(4):392–416, 2013.

8 Rolf Klein, Christian Knauer, Giri Narasimhan, and Michiel Smid. Exact and approximation algorithms for computing the dilation spectrum of paths, trees, and cycles. In *Proceedings of the 16th International Symposium on Algorithms and Computatiom, (ISAAC 2005)*, pages 849–858, 2005.

9 Jun Luo and Christian Wulff-Nilsen. Computing best and worst shortcuts of graphs embedded in metric spaces. In *Proceedings of the 19th International Symposium on Algorithms and Computatiom, (ISAAC 2008)*, pages 764–775, 2008.

10 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

11 Anneke A. Schoone, Hans L. Bodlaender, and Jan van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987.

Approximate Shortest Distances Among Smooth Obstacles in 3D

Christian Scheffer[1] **and Jan Vahrenhold**[2]

1 **Department of Computer Science, Technische Universität Braunschweig, Mühlenpfordtstraße 23, 38106 Braunschweig, Germany**
scheffer@ibr.cs.tu-bs.de

2 **Department of Computer Science, Westfälische Wilhelms-Universität Münster, Einsteinstr. 62, 48149 Münster, Germany**
jan.vahrenhold@uni-muenster.de

Abstract

We consider the classic all-pairs-shortest-paths (APSP) problem in a three-dimensional environment where paths have to avoid a set of smooth obstacles whose surfaces are represented by discrete point sets with n sample points in total. We show that if the point sets represent ε-samples of the underlying surfaces, $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximations of the distances between all pairs of sample points can be computed in $\mathcal{O}(n^{5/2} \log^2 n)$ time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Geodesic distances, approximation algorithm, epsilon sample

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.60

1 Introduction

Computing shortest distances between pairs of points is one of the classic problems in Computational Geometry. The problem is well-understood in (geometric) graphs and in the Euclidean plane [11]. Computing exact shortest paths among obstacles in three-dimensional Euclidean space, however, was shown to be $\mathcal{NP}$-hard [7]. Subsequently, authors have considered special cases such as exact distances on a convex polyhedron [16] or approximate distances on a general, possible weighted polyhedron [2], see also the survey by Bose et al. [6].

We consider a set of smooth obstacles in $\mathbb{R}^3$ given as an ε-sample, i.e., a point set on the union of the obstacles' boundaries locally dense enough to faithfully capture curvature and folding. As usual, ε is a sampling parameter unknown to the algorithm [4, 14]. In line with previous approaches (see [15] and the references therein), we assume that ε is upper-bounded by a constant $\varepsilon_0 > 0$ which only depends on the algorithm but neither on the input size nor on the curvature or folding of the underlying surface. We obtain the following result:

▶ **Theorem 1.** *There is a global and shape-independent constant $\varepsilon_0 > 0$ such that it holds for $\varepsilon \leq \varepsilon_0$: Given an ε-sample S of a set of smooth obstacles in $\mathbb{R}^3$, we can compute $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximations of all $\binom{n}{2}$ distances in $\mathcal{O}(n^{5/2} \log^2 n)$ time, where $n := |S|$.*

In general, shortest paths among obstacles alternate between geodesic subpaths on obstacles and straight-line segments in free space. The standard approach to computing such *free-space geodesics* would be to compute both geodesic distances and visibility edges between each pair of points, model these distances by a weighted graph G with vertices corresponding to the sample points on the obstacles, and then to combine the results using an all-pairs shortest path algorithm on G. Due to the complexity of visibility maps in three-dimensional

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 60; pp. 60:1–60:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

space, this approach would lead to an at least cubic runtime. We will alleviate this problem by simultaneously restricting the degree of G and locally bounding the length of the edges.

Related Work

A free-space geodesic is modeled by two types of edges in G that correspond to either geodesics on obstacles or straight-line segments in free space. For the geodesics on obstacles, we note that exact shortest path computations on general polyhedra are considered complex and challenging. We refer to recent surveys [3, 6, 11] for a detailed discussion and focus on two approximation algorithms: the best algorithm currently known for weighted polyhedra is the algorithm by Aleksandrov et al. [3], while the algorithm by Scheffer and Vahrenhold [14] works on (unweighted) 2-manifolds in $\mathbb{R}^3$. The efficiency of the algorithm by Aleksandrov et al. depends on the triangles obeying a "fatness" condition. In general, however, the aspect ratio and, hence, the runtime can be arbitrarily large. Both algorithms result in a shortest-path graph of quadratic size. Exploring this structure in combination with a visibility graph (see below) as part of a shortest-path algorithm leads to at least cubic runtime.

The second type of edges of a free-space geodesic corresponds to straight-line segments connecting points on obstacles by crossing the free space; these *bridge edges* are obtained by computing visibility information between points on the obstacles. Despite recent advances in algorithms for "realistic terrains" [12], the complexity of the visibility map of a three-dimensional surface is quadratic in the worst case—see [12] and the references therein. As discussed above, a visibility map of quadratic size leads to a cubic overall running time. A subquadratic complexity currently can be obtained only under standard assumptions about the fatness of the triangles and the (bounded) ratio of shortest and longest edges; in particular the latter assumption is infeasible in the case we are considering.

2 Outline of the Algorithm

To simplify the exposition, we will assume that the obstacles' boundary consists of exactly one surface Γ. Since even in that case straight-line segments, i.e., *bridge edges*, may be needed on a free-space geodesic to bridge cavities in the manifold $\mathcal{F}$ bounded by Γ, it is easy to see that our algorithm easily generalizes to multiple non-intersecting obstacles.

From a high-level perspective, our algorithm proceeds by first constructing a weighted graph $G := \left(S^{sub}, E_{loc} \cup E_{bri}\right)$ on a subset $S^{sub} \subseteq S$ of sample points where the edges in E_{loc}, called *local edges*, represent approximate free-space geodesic distances between points on Γ and the edges in E_{bri} represent straight-line segments avoiding Γ; in either case, the weight of an edge denotes the length of the respective connection. The algorithm then approximates the free-space geodesic distances $L^{\star}_{\Gamma}\left(s_1, s_2\right)$ for all $s_1, s_2 \in S^{sub}$ by computing exact all-pairs shortest paths in G and extends these results to compute approximate distances $\mathcal{L}\left(\cdot, \cdot\right)$ between all points in S. The algorithm is given below and will be discussed in the following.

As sketched above, the main challenge in obtaining an algorithm with subcubic running time lies in working with a shortest-path graph with bounded or at least sublinear degree. As we will detail below, we can obtain such a graph by first avoiding to compute approximate free-space geodesic distances between *all* pairs of points. Instead, we will compute such distances only for points within a locally bounded distance of each other. These distances will be represented by the set E_{loc} of local edges connecting points in a carefully coarsened subsample $S^{sub} \subseteq S$. We then compute a set E_{bri} of bridge edges such that E_{bri} is a superset of the visibility graph of S^{sub} w.r.t. (the sample points of) Γ. In both cases, we can simultaneously guarantee a sublinear node degree and a good approximation quality.

Algorithm 1 Approximating Geodesic Distances.

```
function APX3DGeodesicDistances($S$)
    $\psi(\cdot) \leftarrow$ ControlFunction($S$);    ▷ Setup approximation using Lemma 4 [9, 14]...
    $afs(\cdot) \leftarrow$ APXLocalFeatureSize($S$);    ▷ ...and algorithm by Aichholzer et al. [1]
    $\delta \leftarrow \max_{s \in S} \frac{\psi(s)}{afs(s)}$;    ▷ Lower-bound local feature size
    $S^{sub} \leftarrow$ CoarsenSample($S, \delta, afs(\cdot)$);    ▷ Use Algorithm 2
    $E_{loc} \leftarrow$ ComputeLocalEdges($S^{sub}, \delta, afs(\cdot)$);    ▷ Use Algorithm 3
    $E_{bri} \leftarrow$ ComputeBridgeEdges($S^{sub}, S, \delta, afs(\cdot)$);    ▷ Use Algorithm 5
    $G \leftarrow (S^{sub}, E_{loc} \cup E_{bri})$;    ▷ Assemble graph $G$
    return APXDistancesFromGraph($S$, $G$);    ▷ Expand result from $S^{sub}$ to $S$
```

In the remainder of this section, we consider both types of edges in turn and show how to efficiently compute them. We then discuss how to combine local and bridge edges into a graph G that is sparse enough to be traversed efficiently. Finally, we analyze the resulting algorithm w.r.t. to its running time and approximation quality.

2.1 Computing Local Edges

The situation we are facing is similar to the construction of spanner graphs approximating the full Euclidean graph. One way of constructing a spanner graph is by means of the *well-separated pair decomposition* [10]. Informally, this approach connects (representative points of) clusters that are "far away" from each other, where the notion of "far away" depends on the radius of the clusters. Doing so, the intra-cluster distances are approximated by the length of the edge connecting the representatives.

We proceed along similar lines: we consider only edges between points that are at bounded distance from each other where the notion of "bounded" depends on the sampling density and the curvature and folding of Γ at the points in question. This allows us to relate Euclidean and geodesic distances and thus to upper-bound the approximation quality of the geodesic distances computed. A naive implementation of this approach, however, might lead to a linear number of edges per point. To avoid such situations, we need to ensure that only few "short" edges are constructed per point; this will be done by applying a standard preprocessing step in which the sample is coarsened appropriately without affecting the sampling quality [14].

In contrast to the construction of the spanner graph, the construction steps sketched do not guarantee a linear number of edges to be sufficient to obtain a good approximation quality. What we can show, however, is that by considering only edges for points whose Euclidean distance is upper-bounded as sketched above, we obtain a graph with $\mathcal{O}(n^{3/2})$ edges which still results in the desired approximation quality.

2.1.1 Characterizing and Approximating the Sampling Density

To measure the density of a point sample, it is customary to consider the *local feature size* $lfs(\cdot)$ that is defined as the distance function to the medial axis of Γ [4]. To formalize notation, a discrete subset S of $\Gamma \subset \mathbb{R}^3$ is an *ε-sample* of Γ if for every point $x \in \Gamma$ there is a sample point $s \in S$ such that its distance $|xs|$ to s is upper-bounded by $\varepsilon \cdot lfs(x)$. The local feature size $lfs(\cdot)$ is *c-Lipschitz* for $c = 1$, i.e., $lfs(x) \leq lfs(y) + c|xy| = lfs(y) + |xy|$, $x, y \in \mathbb{R}^3$.

In [14], we show that $|x_1x_2|$ for $x_1, x_2 \in \Gamma$ is an $(1 + \mathcal{O}(\varepsilon))$-approximation of the geodesic distance $L_\Gamma(x_1, x_2)$ on Γ between x_1 and x_2 if $|x_1x_2| \leq \sqrt{\varepsilon} \cdot \min\{lfs(x_1), lfs(x_2)\}$. Put differently, if points are "close enough" to each other, their Euclidean distance approximates their geodesic distance. While we use this upper bound in [14] to prove the approximation quality of the geodesic distances computed, the discussion above suggests that our algorithm needs to actually evaluate these expressions to compute the radii of the locally bounded neighborhoods mentioned above and thus to be able to exclude points at larger distance from comparison. Unfortunately, neither ε nor $\mathit{lfs}(\cdot)$ can be computed exactly as Γ is unknown.

What we can do, however, is to compute an approximate upper bound $\mathit{afs}(\cdot)$, the *approximate local feature size*, for $\mathit{lfs}(\cdot)$ such that $\mathit{lfs}(s) \leq \mathcal{O}(1) \cdot \mathit{afs}(s)$ for all $s \in S$. Furthermore, we compute a so-called control function $\psi(\cdot)$ such that $\psi(s) \leq \mathcal{O}(\varepsilon) \cdot \mathit{lfs}(s)$ for all $s \in S$. Finally, we compute an approximate lower bound $\delta := \max_{s \in S}(\psi(s)/\mathit{afs}(s))$ for ε.

We follow Aichholzer et al. [1] and apply the distance from $s \in S$ to the closest *pole.*

▶ **Definition 2** ([4])**.** The *poles* of some $s \in S$ are the two vertices of the Voronoi cell $\mathit{Vor}_S(s)$ of s in the Voronoi diagram of S which are farthest from s, one on either side of Γ (note that Γ is the boundary of a 2-manifold, hence the inside and outside are well-defined). A pole p_s of s is called an *outer* pole if it lies outside Γ, and an *inner* pole otherwise.

Aichholzer et al. observed that this distance is the desired approximate upper bound $\mathit{afs}(s)$ for the local feature size $\mathit{lfs}(s)$, i.e., that $\mathit{lfs}(s) \leq 1.2802 \cdot \mathit{afs}(s)$ holds. Obviously, we can compute the (Voronoi diagram and the) poles and, hence, $\mathit{afs}(\cdot)$ in quadratic time. While the algorithm only needs to know the values of $\mathit{afs}(\cdot)$ for all points in S, the analysis will use a version of this function lifted to $\Gamma \subset \mathbb{R}^3$. For this analysis, we can assume that we can extend the domain of $\mathit{afs}(\cdot)$ to Γ– if needed, pointwise – such that $\mathit{lfs}(x) \leq 1.2802 \cdot \mathit{afs}(x)$ holds for all $x \in \Gamma$ (note that $\mathit{lfs}(\cdot)$ is defined for each point $x \in \Gamma$).

▶ **Observation 3.** *$\mathit{afs}(\cdot)$ is 1-Lipschitz and can be computed in $\mathcal{O}(n^2)$ time.*

We then obtain the approximate lower bound δ for ε using a *control function*:

▶ **Lemma 4** ([14])**.** *We can compute in time $\mathcal{O}(n^2)$ a control function $\psi : S \longrightarrow \mathbb{R}^+$ such that: (1) $\forall s \in S : \psi(s) \leq 1.19 \cdot \varepsilon \cdot \mathit{lfs}(s)$, (2) $\forall s \in S : \forall x \in \mathit{Vor}(s) \cap \Gamma : |xs| \leq \psi(s)$, and (3) ψ is $\frac{1}{18}$-Lipschitz.*

In line with the above argumentation, for $s_1, s_2 \in S$ we show $|s_1 s_2| \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1, s_2)$ if $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{\mathit{afs}(s_1), \mathit{afs}(s_2)\}$ holds, i.e., the above approximation scheme still yields meaningful results for sample points $s_1, s_2 \in S$ “close enough” to each other.

In our previous work [15], we proved:

▶ **Lemma 5** ([15, Lemma 23])**.** *There is a global and shape-independent constant ε_0 such that for all $\varepsilon \leq \epsilon_0$, the approximate lower bound δ for ε satisfies $\frac{1}{\delta} \in \mathcal{O}(\sqrt{n})$.*

2.1.2 Coarsening the Sample

It remains to discuss how to avoid high-degree nodes in the distance graph, or, equivalently, to avoid connecting points to “too many” other points that fulfill the above distance criterion. For this, we use the control function implied by Lemma 4 to compute a coarsened subsample $S^{sub} \subseteq S$ in which the following two conditions hold:

1. For each point $x \in \Gamma$, there is a sample point $s \in S^{sub}$, such that $|xs| \leq \mathcal{O}(\delta) \cdot \mathit{afs}(s)$.
2. For any two sample points $s \neq s' \in S^{sub}$, $|ss'| \geq \mathcal{O}(\delta) \cdot \mathit{afs}(s)$ holds.

Algorithm 2 Compute a coarsened subsample $S^{sub} \subseteq S$ (see [9, 14]).

```
function CoarsenSample($S$, $\delta$, $afs(\cdot)$)
    $S^{sub} \leftarrow \emptyset$; $\beta \leftarrow 0.1$;    ▷ Fix constant in “big-oh”-notation for later analysis
    while $S \neq \emptyset$ do
        $s \leftarrow$ arbitrary point in $S$;
        $S^{sub} \leftarrow S^{sub} \cup \{s\}$;
        $S \leftarrow S \backslash B_{\beta \cdot \delta \cdot afs(s)}(s)$;    ▷ $B_r(x)$: ball with radius $r$ centered at $x$
    return $S^{sub}$;
```

Algorithm 3 Compute the set E_{loc} of local edges.

```
function ComputeLocalEdges($S^{sub}$, $\delta$, $afs(\cdot)$)
    $E_{loc} \leftarrow \emptyset$;
    for all $(s_1, s_2) \in S^{sub} \times S^{sub}$ do
        if $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$ then
            $e \leftarrow (s_1, s_2)$; $weight(e) \leftarrow |s_1 s_2|$; $E_{loc} \leftarrow E_{loc} \cup \{e\}$;
    return $E_{loc}$;
```

2.1.3 Intermediate Summary: Computing Local Edges

Summarizing the above discussion, we first coarsen the subsample using Algorithm 2 and then construct local edges between all points that are close enough—see Algorithm 3.

▶ **Lemma 6.** *Let S be an ε-sample and E_{loc} the set of local edges computed for a coarsened subsample $S^{sub} \subseteq S$ according to Algorithm 2 and 3. Then, the following properties hold:*

(LE1) *The length of a local edge is a $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximation of the geodesic distance of its endpoints. More precisely, for all $s_1, s_2 \in S$ such that $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$ holds, we have $L^{\star}_{\Gamma}(s_1, s_2) \leq (1 + \mathcal{O}(\delta)) \cdot |s_1 s_2| \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot |s_1 s_2|$.*

(LE2) *Local edges can be asymptotically longer than the sampling density (described in terms of $afs(\cdot)$ and δ) by a factor of $\Theta(\sqrt{\delta})$, i.e., they connect points that are relatively “far away” from each other similar to cluster centers in a well-separated pair decomposition.*

(LE3) *Each sample point $s \in S^{sub}$ is incident to at most $\mathcal{O}(\sqrt{n})$ local edges.*

Proof.

(LE1) (Omitted due to space constraints.)

(LE2) Fix a point $s \in S^{sub}$ and let x be any point in $\mathrm{Vor}_{S^{sub}}(s)$. By Lemma 7 there is some $s' \in S^{sub}$ such that $|xs'| \leq 1.17 \cdot \delta \cdot afs(s')$. Since $afs(\cdot)$ is Lipschitz, it follows that $|xs'| \leq 1.2 \cdot \delta \cdot afs(x)$. Since $x \in \mathrm{Vor}_{S^{sub}}(s)$, we have $|xs| \leq |xs'| \leq 1.2 \cdot \delta \cdot afs(x)$. Again, since $afs(\cdot)$ is Lipschitz, we conclude that $|xs| \leq \Theta(\delta) \cdot afs(s)$. Now, consider any local edge $(s_1, s_2) \in S^{sub} \times S^{sub}$. By construction, the maximum length $\nu_{\max}$ of any such edge is $\nu_{\max} := \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$. Even if $afs(s_1) = \max\{afs(s_1), afs(s_2)\}$, the fact that $afs(\cdot)$ is Lipschitz, together with the small distance of s_1 and s_2, implies that $\nu_{\max} \geq afs(s_1) \cdot \Theta(\sqrt{\delta})$. Thus, $\nu_{\max} \cdot \Theta(\sqrt{\delta}) \geq |xs_1|$. The same argument applies to s_2.

(LE3) [Sketch] As $afs(\cdot)$ is 1-Lipschitz, we can show $afs(s') \geq (1 - \frac{1}{3} \cdot \delta) \cdot afs(s) \geq \frac{1}{2} \cdot afs(s)$ for $s' \in B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s)}(s)$. Algorithm 2 guarantees $|ss'| \geq 0.1 \cdot \delta \cdot afs(s)$ for all $s, s' \in S^{sub}$, $s \neq s'$. A standard packing argument yields $|B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s)}(s) \cap S^{sub}| \leq \frac{1}{\delta}$. As all sample points connected to s by local edges lie inside $B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s)}(s)$, we can upper-bound the number of local edges incident to s by $\frac{1}{\delta} \in \mathcal{O}(\sqrt{n})$; see Lemma 5. ◀

▶ **Lemma 7.** *For each $x \in \Gamma$, there is a $s' \in S^{sub}$ with $|xs'| \leq 1.17 \cdot \delta \cdot afs(s')$.*

Proof. Let $q \in S$ be the sample point closest to x. We distinguish between two cases:

1. $q \in S^{sub}$: We define $s' := q$. By definition, $\delta = \max_{p \in S} \frac{\psi(p)}{afs(p)} \geq \frac{\psi(s')}{afs(s')}$. Since, by Lemma 4, $|xs'| \leq \psi(s')$, we have $|xs'| \leq \frac{\psi(s')}{afs(s')} \cdot afs(s') \leq \delta \cdot afs(s')$.
2. $q \in S \setminus S^{sub}$: Define $s' \in S^{sub}$ to be the sample point that was processed by Algorithm 2 when q was excluded from further consideration (Line 6). With $\beta = 0.1$, this implies that $|s'q| \leq 0.1 \cdot \delta \cdot afs(s')$. As $afs(\cdot)$ is 1-Lipschitz, we get $afs(q) \leq (1 + 0.1 \cdot \delta) \cdot afs(s')$. Since q is the sample point closest to x, we have $|xq| \leq \delta \cdot afs(q)$ (see above). The triangle inequality implies then $|xs'| \leq |xq| + |qs'| \leq \delta \cdot afs(q) + 0.1 \cdot \delta \cdot afs(s') \leq 0.1 \cdot (1 + 0.1 \cdot \delta) \cdot \delta \cdot afs(s') + \delta \cdot afs(s') \leq 1.17 \cdot \delta \cdot afs(s')$ (since $\delta^2 < \delta$). ◀

2.2 Computing Bridge Edges

The second type of edges used in our construction is the set E_{bri} of bridge edges. While we would ideally compute the visibility graph of S^{sub} w.r.t. Γ, we cannot do so as the exact geometry of Γ is unknown. We thus compute E_{bri} as a superset of the edges in the visibility graph making sure that the additional edges that may intersect the interior of Γ do so not too deep; this will enable us to bound the approximation error.

2.2.1 Computing Approximate Visibility Information

As the exact nature of Γ is unknown, we cannot compute the visibility map of $s' \in S^{sub}$ w.r.t. Γ. Neither can we use a polyhedral reconstruction of Γ as the visibility map of s' w.r.t. such a reconstruction may have quadratic complexity [12, Sec. 2.1]. To circumvent this problem, we refrain from reconstructing Γ at all. Instead, we discretize Γ by a set of carefully constructed cubes corresponding to all points in S and compute the visibility maps w.r.t. theses cubes.

For this, we require that the visibility information obtained in this way approximates the true visibility information. First, we require that the cubes indeed cover Γ (recall that Γ is the boundary of a manifold $\mathcal{F}$, not $\mathcal{F}$ itself) such that no obstacles are ignored or holes appear, and, second, we require that the cubes do not cover too much of the space outside Γ in the sense that they block visibility rays that Γ does not block. To fulfill these requirements, we compute, for each $s'' \in S^{sub}$, a cube that is centered at s'' and contains a ball of radius $2 \cdot \delta \cdot afs(s'')$ (intuitively, this means that the cubes are large enough to overlap with "neighboring" cubes and thus cover Γ). We then push all cubes towards the interior of $\mathcal{F}$ such that they do not protrude from $\mathcal{F}$ and thus block visibility rays that $\mathcal{F}$ would not block. Based upon Amenta and Bern's [4] observation that for any point $s \in S$ the vector from s towards one of its poles approximates the respective surface normals in s, we push the cubes along the vector from s towards its inner pole—see Algorithm 4.

Algorithm 4 Compute centers of the cubes pushed towards the interior of the manifold $\mathcal{F}$.

```
function ComputeCubeCenters($S, \delta, afs(\cdot)$)
    for all $s \in S$ do
        $p_s \leftarrow$ inner pole of $s$;
        $s^{\downarrow} \leftarrow s + 15 \cdot \delta \cdot afs(s) \cdot \frac{\overrightarrow{sp_s}}{|sp_s|}$;
        $S^{\downarrow} \leftarrow S^{\downarrow} \cup \{s^{\downarrow}\}$;
    return $S^{\downarrow}$;
```

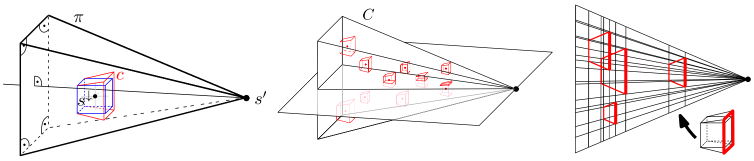


Figure 1 Left: Construction of a skewed cube $c \in \mathbb{C}(\pi, s')$ inside a pyramid. The base of c is the ball with center in s and a radius of $2 \cdot \delta \cdot \mathit{lfs}(s)$ w.r.t. the ℓ_∞-metric. Thus $B_{2\cdot\delta\cdot \mathit{lfs}(s)}(s) \subset c$. The sides of c are slanted outwards from the back face by the same angle as the aperture of π, see cube c. Finally, c is pushed in the interior of the solid $\mathcal{F}$ bounded by Γ by pushing s' into the direction of p_s by $15 \cdot \delta \cdot \mathit{afs}(s)$ where p_s is the *inner pole that corresponds to* s. Middle: Cross-section of the pyramid during the sweep. Right: Projection of the cubes.

For technical reasons, when computing the visibility information of a point $s' \in S^{sub}$, we cover the space with a constant number of pyramids with apex at s' and compute the visibility information for each pyramid separately. After we have identified all cubes intersecting a pyramid π with apex s' (this takes linear time per each of the $\mathcal{O}(1)$ pyramids), we perform a top-down sweep (in decreasing z-order) over all cubes crossing π and all sample points in S^{sub} inside the cone and maintain only the cross-section of the sweeping plane with the scene. Whenever we encounter a sample point, we locate this point in the cross-section and check its visibility from s'. To ensure that maintaining the cross-section is not too costly, we do not work with the axis-aligned cubes we just computed. Instead, we approximate the scene by a set of carefully skewed cubes such that their cross-section with the sweeping plane can be maintained efficiently and their geometry does not induce too many events where the combinatorial nature of the cross-section changes.

More precisely, for a a pyramid π with apex s', we construct for each sample point $s \in S^{sub} \setminus \{s'\}$ a skewed cube $c := c_s$ such that the following properties hold—see Figure 1:

1. The front and back face of the cube c are parallel to the base of π.
2. The sides of c are slanted outwards from the back face by the same angle as π's aperture. (Here, we need that we are working with a constant number of pyramids per point.)
3. The cube c is centered at $s^{\downarrow}$.

For fixed $s' \in S^{sub}$ and π, we denote the set of all skewed cubes intersecting π by $\mathbb{C}(\pi, s')$. We define the *visible neighborhood* $V(\pi, s')$ *of* s' *w.r.t.* π as the union of all points from $S^{sub} \cap \pi$ that are visible from s' w.r.t. $\mathbb{C}(\pi, s')$.

▶ **Lemma 8.** *For fixed* $s' \in S^{sub}$ *and* π*, we can compute* $V(\pi, s')$ *in* $\mathcal{O}(n \log^2 n)$ *time.*

Proof. We perform a standard space-sweep in which we process the points and the skewed cubes' front faces in radial order from the top to the bottom face of the pyramid—see Figure 1 (middle). The important fact to note is that, by construction, the visible silhouette of the set of these cubes is exactly the projection of the set of their front faces onto the base of the pyramid–see Figure 1 (right). The sweep-line structure maintained by the algorithm is a segment tree $\mathcal{T}$ over the x-coordinates of the projections of these front faces. Whenever we encounter the top edge of a front face f, we add f to the set of obstacles currently active but inserting its x-interval into the sweep-line structure. At each node of $\mathcal{T}$ whose extent is covered by f, we insert f into a list of faces sorted by their distance to s'. Analogously, we remove a face from $\mathcal{T}$ once we encounter its bottom edge. Because of the way the skewed cubes have been constructed, i.e., because the aperture of the pyramid and the slanting angle of the cube coincide, the intersection of the sweeping plane and the skewed cubes changes only at the top and bottom edges of the cubes. Whenever we encounter a sample point s, we query $\mathcal{T}$ with the x-coordinate of s. At each node of $\mathcal{T}$ visited, we check the sorted list of faces to see whether there is any face currently stored in $\mathcal{T}$ that blocks s from s'. If no such face is found along the root-to-leaf path in $\mathcal{T}$, s can be seen from s', otherwise s is blocked. The running time is easily seen to be $\mathcal{O}(n \log^2 n)$, as preprocessing takes $\mathcal{O}(n \log n)$ time and all update and query operations take at most $\mathcal{O}(\log n)$ time per node visited. ◀

Finally, we define the *visibility neighborhood* of s as $V(s') := \bigcup_{\pi \in \Pi} V(\pi, s')$.

▶ **Corollary 9.** *For each $s' \in S^{sub}$, we can compute $V(s')$ in $\mathcal{O}(n \log^2 n)$ time.*

2.2.2 Bounding the Degree of the Approximate Visibility Graph

Summarizing the above, we would like to connect each $s' \in S^{sub}$ with all $s \in V(s')$ by bridge edges. This approach can result in $|V(s')| \in \Theta(n)$. The final challenge thus is to compute an approximation of $V(s')$ that results in sublinear-degree vertices in the visibility graph.

▶ **Definition 10** ([13])**.** Let $X \subset \mathbb{R}^3$ be a discrete point set and let x be an arbitrary point in X. An approximate neighborhood $AH(x) := AH_\zeta(x)$ of x w.r.t. X is defined as a subset of $X \setminus \{x\}$, such that there exists a set of cones $\mathcal{C}(x) := \mathcal{C}_\zeta(x)$, with apex at x and an angular radius of ζ that covers $\mathbb{R}^3$, such that a point $x' \in X \setminus \{x\}$ belongs to the approximate neighborhood $AH(x) := AH_\zeta(x)$ iff there is a cone $C \in \mathcal{C}(x)$ such that x' is the point in $C \cap X$ minimizing the distance from its orthogonal projection onto the axis of C to x.

▶ **Lemma 11** ([13])**.** *For $\zeta > 0$, approximate neighborhoods, each one of size $\Theta\left(\zeta^{-2}\right)$, for all points from X can be computed in overall time $\mathcal{O}\left(|X|/\zeta^2 \cdot \log^2(|X|)\right)$*

Stated in terms of Lemma 11, we compute the set E_{bri} of bridge edges (see Section 2.2) in the weighted graph $G = (S^{sub}, E_{loc} \cup E_{bri})$ as follows: we iterate over all $s' \in S^{sub}$, compute $V(s')$ and then, for $\zeta := \sqrt{\delta} > 0$, approximate neighborhoods for the points in $V(s')$.

As a technicality, we wish to guarantee that bridge edges are not of local nature. Hence, we just consider edges that are longer than local edges (see Section 2.1.3). Thus, for $s' \in S^{sub}$, we define $A(s')$ as the $\sqrt{\delta}$-approximate neighborhood of s' w.r.t. $V(s') \setminus B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s')}(s')$.

▶ **Corollary 12.** *For $s' \in S^{sub}$, we can compute $A(s')$ in $\mathcal{O}(\max\{n \log^2 n, n/\delta \log^2 n\})$ time.*

Algorithm 5 Compute the set E_{bri} of bridge edges.

```
function ComputeBridgeEdges($S^{sub}, S, \delta, afs(\cdot)$)
    $E_{bri} \leftarrow \emptyset$;
    $S^{\downarrow} \leftarrow$ ComputeCubeCenters($S, \delta, afs(\cdot)$);                ▷ Use Algorithm 4
    for $s' \in S^{sub}$ do
        $\Pi \leftarrow$ ComputePyramids($s', S^{\downarrow}$);                         ▷ See Figure 1
        $V(s') \leftarrow$ ComputeVisibleNeighborhood($s'$, $\pi$, $S^{sub}$, $S^{\downarrow}$);   ▷ Use Corollary 9
        $A(s') \leftarrow$ APXVisibleNeighborhood($s', \delta$, $V(s')$);             ▷ Use Lemma 11
        for $x \in A(s')$ do
            $e \leftarrow (s', x)$; $weight(e) \leftarrow |s'x|$; $E_{bri} \leftarrow E_{bri} \cup \{e\}$;
    return $E_{bri}$;
```

We show that E_{bri} fulfils the requirements outlined at the beginning of Section 2.2.1:

▶ **Lemma 13.** *Let S be an ε-sample and E_{bri} the set of edges computed for a coarsened subsample $S^{sub} \subseteq S$ according to Algorithm 2 and 5. Then, the following properties hold:*

(BE1) *E_{bri} is a superset of the visibility edges of S^{sub} w.r.t. Γ: for $s \in S^{sub}$, E_{bri} contains all edges (s, s') such that $s' \in S^{sub}$ and $ss' \cap \mathcal{F}^\circ = \emptyset$ ($\mathcal{F}$ is the solid bounded by Γ).*

(BE2) *Let $(s, s') \in E_{bri}$ such that $ss' \cap \mathcal{F}^\circ \neq \emptyset$. The intersection is not too deep, hence, the shortcut taken not too short. More formally, for each edge $(s, s') \in E_{bri}$ and for any point $x \in ss' \cap \mathcal{F}$, there is a sample point $s_x \in S^{sub}$ such that $|xs| \leq 18 \cdot \delta \cdot \min\{afs(x), afs(s_x)\}$.*

(BE3) *Each sample point $s \in S^{sub}$ is incident to at most $\mathcal{O}(\sqrt{n})$ bridge edges.*

Proof.

(BE1) We guarantee that each skewed cube c_s lies inside the solid $\mathcal{F}$ bounded by Γ. In order to do this, we first show that c lies inside a ball with radius $3.82 \cdot \delta \cdot afs(s)$ and centered in $s^\downarrow$, where s denotes the sample point corresponding to c. Furthermore, we show $s^\downarrow \in \mathcal{F}$ and that the distance between $s^\downarrow$ and Γ is lower-bounded by $9 \cdot \delta \cdot afs(s^\downarrow)$. Finally, the triangle inequality implies (BE1). We omit the details due to space constraints.

(BE2) See Section 3.1.2 for the proof.

(BE3) For each $s' \in S^{sub}$, i.e., for each node of G, we use the algorithm by Ruppert and Seidel [13], to compute an approximate $\sqrt{\delta}$-neighborhood for s' w.r.t. $V(s')$ and connect s' to one representative per cone. With $\zeta := \sqrt{\delta}$, the nodes of G thus are incident to $\mathcal{O}(\zeta^{-2}) = \mathcal{O}(\delta^{-1})$ edges. Using again Lemma 5, we observe that $\delta^{-1} \in \mathcal{O}(\sqrt{n})$. ◀

Combining Lemma 6 and Lemma 13, we see that the weighted graph $G = (S^{sub}, E_{loc} \cup E_{bri})$ has a sublinear node degree and is well-suited to approximate the sought geodesic distances since the edges are either bridge edges, i.e., (approximate) visibility edges, or local edges, whose lengths are approximations of the geodesic distances of their endpoints.

2.3 Approximating All Distances / Runtime Analysis

We have described how to construct a weighted graph $G := \left(S^{sub}, E_{loc} \cup E_{bri}\right)$ on the coarsened set of sample points. While we can now use Dijkstra's algorithm to compute shortest distances in this graph, i.e., between points in S^{sub}, we also need to discuss how to compute distances between all sample points in S and not only between those in S^{sub}.

The main idea is borrowed from the construction of spanner graphs based upon well-separated pair decompositions [10]. If a sample point s has been excluded from S^{sub} because it was found to lie inside a ball $B_{\beta \cdot \delta \cdot afs(s')}(s')$ of some sample point $s' \in S^{sub}$, the distances to/from s' are good enough approximations of the distances to/from s as long as the destination is "far away"; otherwise, we use the Euclidean distance as an approximation.

Algorithm 6 Deriving an approximation $\mathcal{L}(\cdot,\cdot)$ of $L^\star_\Gamma(\cdot,\cdot)$ from G.

```
function APXDISTANCESFROMGRAPH($S$, $G$)
    Compute shortest path distances $L_G(s_1, s_2)$ for all $s_1, s_2 \in S^{sub}$.
                                        ▷ Use Dijkstra's algorithm from each point in $S^{sub}$;
    for all $s \in S$ do
        $\nu_s \leftarrow$ sample point in $S^{sub}$ closest to $s$;          ▷ $\nu_s = s$ for all $s \in S^{sub}$
    for all $s_1, s_2 \in S$ do
        if $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$ then
            $\mathcal{L}(s_1, s_2) \leftarrow |s_1 s_2|$;                ▷ Approximate by Euclidean distance
        else
            $\mathcal{L}(s_1, s_2) \leftarrow L_G(\nu_{s_1}, \nu_{s_2})$;       ▷ Approximate using closest points in $S^{sub}$
    return $\mathcal{L}(\cdot,\cdot)$;
```

We now have collected all ingredients needed to analyze the running time of Algorithm 1.

▶ **Lemma 14.** *Algorithm 1 has a running time of $\mathcal{O}(n^{5/2}\log^2 n)$.*

Proof. To recall Algorithm 1: we first compute a control function (Lemma 4) and approximate the local feature size (Observation 3). Based upon these results, we can (asymptotically) lower-bound the local feature size. Since we need the poles for this, we construct the Voronoi diagram of all points, which can be done in $\mathcal{O}(n^2)$ time. In the next phase of the algorithm, we work with a coarsened subsample S^{sub} which can be constructed in $\mathcal{O}(n^2)$ time as well (Algorithm 2). Since we compute the set of local edges by iterating over all pairs of points in S^{sub} (Algorithm 3), this step takes $\mathcal{O}(n^2)$ time as well. Computing the set of bridge edges takes $\mathcal{O}(n \cdot \max\{n\log^2 n, n/\delta \log^2 n\}) \leq \mathcal{O}(n^{5/2}\log^2 n)$ time (Corollary 12, Lemma 5). The running time of the final step (Algorithm 6) is dominated by the $\Theta(n)$-fold invocation of Dijkstra's algorithm on a graph with $\mathcal{O}(n)$ vertices and $\mathcal{O}(n^{3/2})$ edges (Lemma 6 (LE3), Lemma 13 (BE3)). Using an efficient priority queue implementation, the running time for this step is $\mathcal{O}(n^{5/2}\log n)$. Hence, the overall running time of Algorithm 1 is $\mathcal{O}(n^{5/2}\log^2 n)$. ◀

3 Analysis of the Approximation Quality

▶ **Lemma 15.** *$\mathcal{L}(\cdot,\cdot)$ is an $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximation of $L^\star_\Gamma(\cdot,\cdot)$.*

To prove Lemma 15, we first relate the value of δ to ε (Lemma 16). In Subsection 3.1, we then show $\mathcal{L}(\cdot,\cdot) \geq (1-\mathcal{O}(\sqrt{\varepsilon}))\cdot L^\star_\Gamma(\cdot,\cdot)$. Finally, we show $\mathcal{L}(\cdot,\cdot) \leq (1+\mathcal{O}(\sqrt{\varepsilon}))\cdot L^\star_\Gamma(\cdot,\cdot)$ (see Subsection 3.2). This concludes the proof of Lemma 15 and, hence, of Theorem 1.

▶ **Lemma 16.** *$\delta \in \mathcal{O}(\varepsilon)$.*

Proof. We combine Lemma 4 and Aichholzer et al.'s observation that $\mathit{lfs}(s) \leq 1.2802 \cdot \mathit{afs}(s)$ holds for all $s \in S$ [1, Lemma 5.1]: $\delta = \max_{s\in S}\frac{\psi(s)}{\mathit{afs}(s)} \leq \max_{s\in S}\frac{\varepsilon/(1-\varepsilon)\cdot \mathit{lfs}(s)}{1.2802^{-1}\cdot \mathit{lfs}(s)} \leq \mathcal{O}(\varepsilon)$. ◀

3.1 Lower-Bounding the Approximation Quality

To ensure $\mathcal{L}(s_1,s_2) \geq (1-\mathcal{O}(\sqrt{\varepsilon}))L^\star_\Gamma(s_1,s_2)$ for all $s_1,s_2 \in S$, we consider a shortest path ϕ in the distance graph G between ν_{s_1} and ν_{s_2}. We then construct a curve γ between s_1 and s_2 in the *free space* $\Lambda := \overline{\mathbb{R}^3 \setminus \mathcal{F}}$ such that $|\gamma| \leq (1+\mathcal{O}(\sqrt{\varepsilon}))\cdot|\phi|$, or, equivalently, $(1-\mathcal{O}(\sqrt{\varepsilon}))\cdot|\gamma| \leq |\phi|$ holds. To show that $|\gamma| \leq (1+\mathcal{O}(\sqrt{\varepsilon}))\cdot|\phi|$ holds, we separately consider the local edges and the bridges edges on ϕ.

3.1.1 Lower-Bounding the Approximation Quality of Local Edges

Lemma 19 gives the lower bound for the length of local edges. In a previous paper [14], we proved a corresponding lower bound for edges whose lengths are related to ε and $\mathit{lfs}(\cdot)$:

▶ **Lemma 17** ([14, Lemma 20])**.** *For $x,y \in \Gamma$ with $|xy| \leq \sqrt{\varepsilon}\cdot\min\{\mathit{lfs}(x),\mathit{lfs}(y)\}$, we have $L_\Gamma(x,y) \leq (1+\mathcal{O}(\varepsilon))\cdot|xy|$, where $L_\Gamma(x,y)$ is the geodesic distance of x and y on Γ.*

Lemma 17 cannot be applied directly to a local edge $(p,q) \in E_{loc}$, since $|pq|$ depends on δ and $\mathit{afs}(\cdot)$ instead of ε and $\mathit{lfs}(\cdot)$. To extend this result to local edges, we can show that a similar statement also applies to free-space geodesic distances in our case:

▶ **Lemma 18.** *For $x,y \in \Lambda$ with $|xy| \leq \sqrt{\varepsilon}\cdot\min\{\mathit{lfs}(x),\mathit{lfs}(y)\}$, we have $L^\star_\Gamma(x,y) \leq (1+\mathcal{O}(\varepsilon))\cdot|xy|$.*

Lemma 18 allows us to iteratively construct the curve γ discussed above by connecting points on Γ and in Λ. We use γ to prove Lemma 19 (proof omitted due to space constraints):

▶ **Lemma 19.** *For $s_1, s_2 \in S$ with $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$, we have $L^{\star}_{\Gamma}(s_1, s_2) \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot |s_1 s_2|$, i.e., $\mathcal{L}(s_1, s_2) \geq (1 - \mathcal{O}(\sqrt{\varepsilon})) \cdot L^{\star}_{\Gamma}(s_1, s_2)$.*

3.1.2 Lower-Bounding the Approximation Quality of Bridge Edges

Lemma 26 gives the lower bound for the length of bridge edges. In a nutshell, we ensure that, given some $(s, q) \in E_{bri}$, for each $x \in \overline{sq} \cap \digamma$ there is a $s_x \in S^{sub}$ such that $|x s_x| \leq \mathcal{O}(\varepsilon) \cdot \min\{afs(s), afs(q)\}$. Applying Lemma 19 multiple times yields $L^{\star}_{\Gamma}(s, q) \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot |sq|$.

To ensure the existence of s_x for $x \in \overline{sq} \cap \digamma$, we ensure that x lies not "too deep" inside $\digamma$, see (BE2). For this, we consider the restricted Delaunay triangulation of S w.r.t. Γ.

▶ **Definition 20** ([5])**.** Let t be the triangle induced by three sample points $s_1, s_2, s_3 \in S$. t is an element of the *restricted Delaunay triangulation* T iff $\mathrm{Vor}_S(s_1) \cap \mathrm{Vor}_S(s_2) \cap \mathrm{Vor}_S(s_3) \cap \Gamma \neq \emptyset$.

▶ **Theorem 21** ([5, Theorem 19])**.** *T is homeomorphic to Γ for $\varepsilon < 0.06$.*

▶ **Definition 22.** For $t \in T$ with corners $s_1, s_2, s_3 \in S$ let $t^{\downarrow}$ be the triangle that is induced by $s_1^{\downarrow}$, $s_2^{\downarrow}$, and $s_3^{\downarrow}$ (see Algorithm 4 for a definition of $\cdot^{\downarrow}$). We define $T^{\downarrow} := \{t^{\downarrow} \mid t \in T\}$.

For each constructed visibility edge, we can show that the skewed cubes cover $T^{\downarrow}$:

▶ **Lemma 23.** *For each $s \in S$ and $\pi \in \Pi$, we have $T^{\downarrow} \subset \bigcup_{c \in \mathbb{C}(\pi, s)} c$.*

We formalize the space Δ "between" $T^{\downarrow}$ and Γ as follows: For each $t = \triangle(s_1, s_2, s_3) \in T$ with $s_1, s_2, s_3 \in S$ and $\zeta \in [0, 1]$, we define $t_{\zeta} := \triangle(s_1 + \zeta(s_1^{\downarrow} - s_1), s_2 + \zeta(s_2^{\downarrow} - s_2), s_3 + \zeta(s_3^{\downarrow} - s_3))$. Also, for $x \in \Gamma$ and $\zeta \in [0, 1]$, we define $x_{\zeta} := x + \zeta(\mu_1(x) - x)$. Finally, we denote $\Delta := \left(\bigcup_{t \in T, \zeta \in [0,1]} t_{\zeta}\right) \cup \left(\bigcup_{x \in \Gamma, \zeta \in [0,1]} x_{\zeta}\right)$.

Assume now that there were some $x \in \overline{sq} \cap \digamma$ not "between" $T^{\downarrow}$ and Γ, i.e., $\overline{sq}$ were penetrating $\digamma$ "too deeply". Theorem 21 and the construction of $T^{\downarrow}$ then would imply the existence of some intersection point y of $\overline{sq}$ and some $t \in T^{\downarrow}$. Lemma 23 would then imply y to lie in one of the skewed cubes used during the construction of the visibility edge between s and q—a contradiction to the correctness of the space-sweep algorithm.

More formally, Theorem 21 implies there is a homeomorphism $\mu_1 : \Gamma \to T$. By construction, there is a continuous and surjective function $\mu_2 : T \to T^{\downarrow}$. Thus, $\mu := \mu_2 \circ \mu_1$ is surjective and continuous. This construction of μ implies that $T^{\downarrow}$ has the same genus as Γ, i.e., has no extra holes. Note that it is not guaranteed that triangles from $T^{\downarrow}$ are intersection free. Using elementary manipulations, we can show:

▶ **Lemma 24.** *For each $x \in \Delta$, there is an $s_x \in S^{sub}$ such that $|x s_x| \leq 18 \cdot \delta \cdot afs(s_x)$.*

As μ is surjective and continuous, $\digamma \setminus \Delta$ is bounded by a subset of $T^{\downarrow}$. Combining this with Lemmas 24 and 23 implies that $\overline{sq}$ does not penetrate $\digamma$ "too deeply":

▶ **Lemma 25.** *For each $x \in \overline{sq} \cap \digamma$ there is a $s_x \in S^{sub}$ with $|x s_x| \leq 18 \cdot \delta \cdot afs(x)$.*

Proof. Assume that there is some $x \in \overline{sq} \cap \digamma$ such that there is no $s_x \in S^{sub}$ with $|x s_x| \leq 18 \cdot \delta \cdot afs(s_x)$. The contraposition of Lemma 24 implies $x \in \digamma \setminus \Delta$. As $\mu : \Gamma \to T^{\downarrow}$ is surjective and continuous, $T^{\downarrow}$ has the same genus as Γ, i.e., has no holes. Thus, there exists some $y \in \overline{sq} \cap T^{\downarrow}$. This implies for all $\pi \in \Pi$, there is some cube $c \in \mathbb{C}(\pi, s)$ such that $\overline{sq} \cap c \neq \emptyset$. Analogously, we obtain for all $\pi \in \Pi$, there is some cube $c \in \mathbb{C}(\pi, q)$ such that $\overline{sq} \cap c \neq \emptyset$. As $(s, q) \in E_{bri}$, this is a contradiction to the correctness of the space-sweep algorithm. ◀

▶ **Lemma 26.** *For $(s,q) \in E_{bri}$, we have $L^\star_\Gamma(s,q) \le (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot |sq|$.*

Combining Lemmas 19 and 26 yields the lower bound for all edges.

3.2 Upper-Bounding the Approximation Quality

To ensure $\mathcal{L}(s_1,s_2) \le (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$ for all $s_1, s_2 \in S$, we again distinguish whether $|s_1s_2| \le \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{\mathit{afs}(s_1), \mathit{afs}(s_2)\}$ holds. If this is the case, we have $\mathcal{L}(s_1,s_2) = |s_1s_2|$, which is trivially upper-bounded by $(1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.

If $|s_1s_2| > \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{\mathit{afs}(s_1), \mathit{afs}(s_2)\}$, Lemma 7 yields $|s_1\nu_{s_1}| \le \mathcal{O}(\delta) \cdot \mathit{afs}(\nu_{s_1})$ and $|s_2\nu_{s_2}| \le \mathcal{O}(\delta) \cdot \mathit{afs}(\nu_{s_2})$. This implies $L^\star_\Gamma(\nu_{s_1},\nu_{s_2}) \le (1+\mathcal{O}(\sqrt{\delta})) \cdot L^\star_\Gamma(s_1,s_2)$. As $\nu_{s_1}, \nu_{s_2} \in S^{sub}$, we can show the required upper bound for $\mathcal{L}(s_1,s_2)$ by applying Lemma 25.

▶ **Lemma 27.** *For all $s_1, s_2 \in S^{sub}$, $\mathcal{L}(s_1,s_2) \le (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.*

Thus, $\mathcal{L}(s_1,s_2) \le (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot (1+\mathcal{O}(\sqrt{\delta})) \cdot L^\star_\Gamma(s_1,s_2) \le (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.

▶ **Corollary 28.** *For all $s_1, s_2 \in S$, $\mathcal{L}(s_1,s_2) \le (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.*

In conclusion, the discussion in this section consitutes a proof of Lemma 15, i.e., we have shown that $\mathcal{L}(\cdot,\cdot)$ is a $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximation of $L^\star_\Gamma(\cdot,\cdot)$. Together with Lemma 14, where we showed the running time of our algorithm to be in $\mathcal{O}(n^{5/2}\log^2 n)$, this constitutes a proof of our main result (Theorem 1).

References

1 Oswin Aichholzer, Franz Aurenhammer, Thomas Hackl, Bernhard Kornberger, Simon Plantinga, Günter Rote, Astrid Sturm, and Gert Vegter. Recovering Structure from r-Sampled Objects. *Computer Graphics Forum*, 28(5):1349–1360, 2009.

2 Lyudmil Aleksandrov, Hristo Djidjev, Anil Maheshwari, and Jörg-Rüdiger Sack. An approximation algorithm for computing shortest paths in weighted 3-d domains. *Discr. Comp. Geom.*, 50(1):124–184, 2013.

3 Lyudmil Aleksandrov, Hristo N. Djidjev, Hua Guo, Anil Maheshwari, Doron Nussbaum, and Jörg-Rüdiger Sack. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. *Discr. Comp. Geom.*, 44(4):762–801, 2010.

4 Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discr. Comp. Geom.*, 22(4):481–504, 1999.

5 Nina Amenta, Sunghee Choi, Tamal K. Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. *Intl. J. Comp. Geom. Appl.*, 12(1–2):125–141, 2002.

6 Prosenjit Bose, Anil Maheshwari, Chang Shu, and Stefanie Wuhrer. A survey of geodesic paths on 3D surfaces. *Comp. Geom.: Theory & Appl.*, 44(9):486–498, 2011.

7 John F. Canny and John H. Reif. New lower bound techniques for robot motion planning problems. In *Proc. Symp. Found. Comp. Sci.*, pp. 49–60, 1987.

8 Mark de Berg, Herman J. Haverkort, and Constantinos P. Tsirogiannis. Visibility maps of realistic terrains have linear smoothed complexity. *J. Comp. Geom.*, 1(1):57–71, 2010.

9 Stefan Funke and Edgar A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. 13th Symp. Discr. Alg.*, pp. 781–790, 2002.

10 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric graphs. In *Proc. 13th Symp. Discr. Alg.*, pp. 828–837, 2002.

11 Joseph S. B. Mitchell. Shortest paths and networks. In Jacob Eli Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, Discrete Mathematics and its Applications, chapter 27, pp. 607–641. CRC Press, 2nd ed., 2004.

12 Esther Moet, Marc J. van Kreveld, and A. Frank van der Stappen. On realistic terrains. *Comp. Geom.: Theory & Appl.*, 41(1–2):48–67, 2008.

13 Jim Ruppert and Raimund Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proc. 3rd Conf. Comp. Geom.*, pp. 207–210, 1991.

14 Christian Scheffer and Jan Vahrenhold. Approximating geodesic distances on 2-manifolds in $\mathbb{R}^3$. *Comp. Geom.: Theory & Appl.*, 47(2):125–140, 2014.

15 Christian Scheffer and Jan Vahrenhold. Approximating geodesic distances on 2-manifolds in $\mathbb{R}^3$: The weighted case. *Comp. Geom.: Theory & Appl.*, 47(8):789–808, 2014.

16 Yevgeny Schreiber and Micha Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. *Discr. Comp. Geom.*, 39(1–3):500–579, 2008.

An Improved Tax Scheme for Selfish Routing*

Te-Li Wang†1, Chih-Kuan Yeh2, and Ho-Lin Chen‡3

1 **Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan**
b01901022@ntu.edu.tw

2 **Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan**
b01901163@ntu.edu.tw

3 **Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan**
holinchen@ntu.edu.tw

Abstract

We study the problem of routing traffic for independent selfish users in a congested network to minimize the total latency. The inefficiency of selfish routing motivates regulating the flow of the system to lower the total latency of the Nash Equilibrium by economic incentives or penalties. When applying tax to the routes, we follow the definition of [8] to define ePoA as the Nash total cost including tax in the taxed network over the optimal cost in the original network. We propose a simple tax scheme consisting of step functions imposed on the links. The tax scheme can be applied to routing games with parallel links, affine cost functions and single-commodity networks to lower the ePoA to at most $\frac{4}{3} - \epsilon$, where ϵ only depends on the discrepancy between the links. We show that there exists a tax scheme in the two link case with an ePoA upperbound less than 1.192 which is almost tight. Moreover, we design another tax scheme that lowers ePoA down to 1.281 for routing games with groups of links such that links in the same group are similar to each other and groups are sufficiently different.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases selfish routing, price of anarchy, tax

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.61

1 Introduction

We study the problem of routing traffic for independent selfish users in a congested network to minimize the total cost (latency). In many settings, it is very expensive or impossible to regulate the traffic precisely. In the absence of regulation, users usually only focus on minimizing his own cost measured by the total time needed to traverse his chosen route. Many works focus on the degradation in network performance measured by comparing the cost of the Nash equilibrium flow and the cost of the optimal setting. The ratio of total cost of Nash Equilibria to the minimum possible cost is defined to be the Price of Anarchy (PoA). Therefore one could consider PoA as an index of the inefficiency of the lack of regulation in a network of selfish behavior. In [25], it is proven that the PoA is $\leq \frac{4}{3}$ for

* Research supported by MOST grant number 104-2221-E-002-045-MY3 and MOST grant number 104-2815-C-002-086-E.

† The first two authors contributed equally to the paper and are listed in alphabetical order.

‡ Corresponding author.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 61; pp. 61:1–61:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

affine latency functions, and the upper bound $\frac{4}{3}$ is tight in a well-known example called the Pigue's example [20]. There are many well-known works on the selfish routing game, such as [22, 23, 24, 21, 6, 18].

The inefficiency of selfish routing motivates regulating the flow of the system to lower the total latency of the Nash Equilibria by economic incentives or penalties. Marginal cost pricing is an ancient idea proposed in [20]. Marginal cost taxes may induce the minimum-latency flow as a flow at a Nash equilibrium, assuming all network users choose the routes to minimize the sum of latency and tax [2]. One major research is to lower price of anarchy to 1 for users having different sensitivity to tax in a single-commodity network [10], with an upper bound of tax with complexity $O(n^3)$. Several further researches improved the result above, such as generalizing the result for single commodity to multi-commodity [12, 15] and generalizing the result for giving an tax upperbound with complexity $O(n)$ [11]. In [4], optimal tax with constraints can be derived in certain circumstances. Another similar concept is the coordination mechanisms introduced in [7]. Coordination Mechanisms have been used to improve the PoA in scheduling problems for parallel and related machines [7, 14, 17] as well as for unrelated machines [1, 5].

In the above researches, the system is efficient only if the tax is returned to the users, otherwise dis-utility for users due to large tax may exist. In [16], an PoA upperbound of 2 is given if tax is included as a part of the cost. The bound becomes 5/4 particularly for affine latency case. On the other hand, it has been proven [9] that marginal tax could not help reduce total cost if tax is considered as a part of the cost for affine cost functions. It is also proven [8] that continuous tax functions yield no improvement to the total latency.

In the above modelings, the total flow r is specified as a part of the game. However, there are situations that the total flow is unknown beforehand, thus finding a good tax scheme becomes more difficult. Christodoulou *et al.* [8] studied this type of problem for single-commodity routing games with affine cost functions. They designed a tax scheme such that the PoA is at most $\frac{4}{3} - \epsilon$ over all possible amount of flow, where ϵ is a constant that approaches 0 when the number of links go to infinity. In this work, arbitrary tax function is allowed as along as the sum of tax and the original cost (latency) function is monotone increasing.

In our work, we focus on step-function congestion tolls. This type of tax scheme has been studied by transport economists to model the effects of the traffic lights on traffic regulations [13, 19]. Compared to arbitrary tax schemes, the step-function congestion tolls is more feasible in transportation regulations. This motivates us to investigate the possibility of improving ePoA using only step-function congestion tolls in settings similar to [8].

Our Result: We provide a simple tax scheme consisting of step functions imposed on the links. The tax scheme is applied to routing games with parallel links, affine cost function, single-commodity networks to lower the ePoA below $\frac{4}{3}-\epsilon$, where ϵ depends on the discrepancy between the links but not the number of links. Moreover, we consider a special case in which all links can be clustered into several groups. The latency function is similar among links in the same group and are sufficiently different between links in different groups. Each group could be seen as different transportation methods. For example, all freeway may belong to one group, and all local roads and railroad may each belong to another group. In this case, we propose a tax scheme which reduces ePoA to 1.281.

The rest of the paper is organized as follows. In Section 2 we describe the basic routing game model and the type of tax scheme that we will use. In Section 3 we define the parameters in the tax scheme more formally and prove some essential results on the relationship between the ePoA and the imposed tax. In Section 4 we show that step function tolls perform equally well to previous optimal (arbitrary) tax scheme for networks with two parallel links. In Section 5 we propose a tax scheme for $\frac{4}{3} - \epsilon$ ePoA. In Section 6 we give an 1.281 upperbound of ePoA for networks with groups of similar links.

2 Model

We consider single-commodity congestion games on networks, defined by a directed traffic network $G = (V, E, l)$, with vertex set V, edge set E, and cost function (or latency function such as [25]) set l. l is the set of cost function l_e for each link $e \in E$. There is only a start node and an end node in V, while each link $e \in E$ connects the start node directly to the end node, and we denote all links $E = \{e_1, e_2, \ldots, e_m\}$. r is defined to be the rate of traffic or the total flow, which is independent of the network G. Unlike some previous works, r is not part of the game. We aim to lower the ePoA for any value of r, instead of choosing a different tax scheme for different value at r. A flow is a function that maps every link $e \in E$ to a non-negative real number. Given G and r, we call a flow feasible if $\Sigma_{e \in E} f_e = r$.

l_e is the cost function of link $e \in E$, which is non-decreasing, non-negative and affine. Therefore we order the links by an increasing order of the constant of the latency of the links. Without loss of generality, we let $l_{e_i}(f) = a_i \cdot f + b_i$ and $b_i \leq b_j$ for any $i, j > 0$ such that $i < j$.

The concept of User Equlibrium [3] is adopted as Nash Equilibrium in this work. Formally, a flow f feasible for traffic network G and total rate r, is at User Equilibrium if and only if for every $e_1, e_2 \in E$ with $f_{e_1} > 0$, $l_{e_1}(f) \leq \lim_{\epsilon \to 0} l_{e_2}(f + \epsilon \mathbf{1_{e2}} - \epsilon \mathbf{1_{e1}})$. It has been proven that for the case where all discontinuity is lower semicontinuous, the User Equilibrium exists as a theorem in [3]. The definition follows an equivalent definition in [25] when the latency function is continuous. We call the flow at Nash Equilibrium, or User equilibrium simply the Nash flow in the rest of the paper. The cost of flow f in traffic network G is $C(f, G) = \Sigma_{e \in E} f_e \cdot l_e(f_e)$. We use $C_{opt}(r, G)$ to denote the minimum cost of any flow feasible at rate r, or the cost of the optimal flow. Therefore the optimal flow is the flow that minimizes the cost of flow for given (G, r), which would be referred to as OPT. Moreover, we say that a flow uses j links when there are j links with non-zero flow-value. We use $C_N(r, G)$ to denote the cost of the Nash flow at rate r, while the uniqueness of the Nash equilibrium is guaranteed in theorem in [3]. When the context is clear, we may omit G, using $C(f)$ for $C(f, G)$, $C_{opt}(r)$ for $C_{opt}(r, G)$, $C_N(r)$ for $C_N(r, G)$. The Price of Anarchy is defined as $PoA(r) = \frac{C_N(r)}{C_{opt}(r)}$, and $PoA = \max_{r>0} PoA(r)$. It should be noted that the PoA defined here is not a function of r as in most previous works. The PoA in our work is the worst case of $PoA(r)$ among any r-value for a particular network G. In the remainder of the paper, we focus on single commodity, parallel-link networks $G = (V, E, l)$, where E consists of m links $\{e_1, \cdots, e_m\}$, and cost function of link e_i is of the form $l_{e_i}(f) = a_i \cdot f + b_i$.

2.1 Tax

On each edge, the original cost function before imposing the tax is $l_{e_i}(f) = a_i \cdot f + b_i$, the tax-modified cost function becomes $\hat{l}_{e_i}(f) = \hat{a}_i \cdot f + \hat{b}_i$, and $\hat{a} = a$. The tax scheme used in our work adds tax $\hat{b}_j - b_j$ to the cost for users using link j, where $\hat{b}_j$ is a function of total flow r.

$$\hat{b_j} = b_j + \sum_{i>j} (b_i - b_{i-1}) \cdot h_i \cdot u(r - w_i), \tag{1}$$

where $h_i < 1$ and w_i are constants to be chosen, and u is the unit step function. Note that to guarantee the existence of User Equilibrium, the unit step function is defined to be lower-semicontinuity. One point to be noted is that under this form of tax, the Nash flow accounting tax on any link is non-decreasing while total rate r increases. This is a desired property, which makes taxing feasible and efficient, since rerouting existing traffic when total traffic increases may be very costly if at all possible.

For the taxed network, we consider adding tax to be a modification to the original network. Therefore, we call $\hat{G}$ the tax-modified network obtained by imposing tax on G. All notations for the taxed network $\hat{G}$ is denoted with a hat, such as the expression $\hat{b}_j$ defined above. We specify that the $\hat{C}_N(r)$ is the total cost of the Nash equilibrium flow of the tax modified network at rate r, where the cost of each edge and the Nash flow are both affected by the tax. We formally define $\mathrm{ePoA} = \max_{r>0}\mathrm{ePoA}(r) = \max_{r>0}\hat{\mathrm{PoA}}(r) = \max_{r>0}\frac{\hat{C}_N(r)}{C_{opt}(r)}$.

3 Useful Inequalities on the PoA and Tax

Before proving the main results, we need to prove some lemmas on the cost of the Nash equilibrium and OPT.

▶ **Definition 1.** We follow notations in previous works. Given a traffic network G, let $\lambda_j = 1/a_j$, $\gamma_j = b_j/a_j$, $\Lambda_j = \Sigma_{i=1}^{j}\lambda_i$, $\Gamma_j = \Sigma_{i=1}^{j}\gamma_j$ and $r_j = \Sigma_{i=1}^{j-1}(b_{i+1} - b_i)\Lambda_i$. We also define $u_j = r_j/r_{j-1}$ and $v_j = \Lambda_j/\Lambda_{j-1}$.

Intuitively, r_j is the amount of flow at which the $(j+1)$-th edge starts to have non-zero Nash flow. PoA is locally maximized at each r_j. The tax schemes we design also seeks to reduce PoA near these values.

Cost of the Nash flow and the OPT on this type of traffic network has been well studied, and closed-form expressions were given [8]. We restate some essential results in Lemma 2.

▶ **Lemma 2** ([8]). *The Nash flow uses link j for $r > r_j$ and the OPT uses link j for $r > r_j/2$. If the OPT uses exactly j links at rate r then*

$$C_{opt}(r) = \frac{1}{\Lambda_j}(r^2 + \Gamma_j r) - C_j, \quad \textit{where} \quad C_j = \Big(\sum_{h=1}^{j}\sum_{i=1}^{h}(b_h - b_i)^2\lambda_h\lambda_i\Big)/(4\Lambda_j).$$

If the Nash flow uses exactly j links at rate r then

$$C_N(r) = \frac{1}{\Lambda_j}(r^2 + \Gamma_j r).$$

If $s < r$ and OPT uses exactly j links at s and r then

$$C_{opt}(r) = C_{opt}(s) + \frac{1}{\Lambda_j}((r-s)^2 + (\Gamma_j + 2s)(r-s)).$$

If $s < r$ and the Nash flow uses exactly j links at s and r then

$$C_N(r) = C_N(s) + \frac{1}{\Lambda_j}((r-s)^2 + (\Gamma_j + 2s)(r-s)).$$

Directly from Lemma 2, we know that both the OPT and the Nash flow start to use links with the same b-value simultaneously because $r_i = r_j$ if $b_i = b_j$.

▶ **Lemma 3.** *Given a traffic network G, if there exists an index i such that $b_i = b_{i+1}$, we can find a network G' having one less link than G such that $C_N(r, G) = C_N(r, G'), C_{opt}(r, G) = C_{opt}(r, G')$ for all r.*

Using Lemma 3, given a traffic network G, we can replace all links with the same b values by one link and let the cost of the Nash and the OPT remain the same. Furthermore, if we apply tax in the new game, we can apply the same tax on every corresponding links in the old game, as a result, we only consider traffic networks such that $b_i \neq b_j, \forall i \neq j$ in the rest of the paper.

Informally, the tax scheme we design works in the following way. For every flow value r_i which corresponds to a local maximum in the PoA-r curve, we add a set of step functions which reduces the tax in the flow range $[\alpha r_i, \beta r_i]$ if the original PoA at flow r_i is greater than a certain threshold. This set of step functions has no effect on PoA when the total flow is less than αr_i but increases PoA marginally when the total flow is greater than βr_i. A tax scheme can be described by a set of parameters (T, A, B), where T is the threshold, $A = \{\alpha_1, \cdots, \alpha_m\}$, $B = \{\beta_1, \cdots, \beta_m\}$ describes the range of flow in which PoA is supressed. When the tax is imposed on a flow value r_i, a step function are added onto the original cost functions for the first i links, where the heights and positions of those step functions are chosen such that the Nash flow on these i links stop increasing when the total flow r is between $[\alpha r_i, \beta r_i]$, causing the Nash flow to use new links. The detailed definition of the tax scheme being used is the following:

▶ **Definition 4.** Given a traffic network G, let G_j be an identical network of G with links e_1 to e_{j-1} removed. Let f_j be the Nash flow on a given a network G_j and rate r, let $C_{Nj}(r)$ be the cost of f_j on G_j.

▶ **Definition 5.** Given a traffic network G, constants T, α_i and β_i such that $\alpha_i < 1$, $\beta_i > 1$ for $1 \leq i \leq m$, Let $A = \{\alpha_1, \cdots, \alpha_m\}$, $B = \{\beta_1, \cdots, \beta_m\}$, $S(T)$ be the set of all index i such that PoA$(r_i) > T$ and $\hat{G}$ be the network obtained from applying tax(T, A, B) to G. The parameters h_j and w_j in equation (1) (Section 2.1), which correspond to the heights and the locations of the step functions are chosen as following,

$$h_j = \begin{cases} \left(\frac{C_{Nj}((\beta_j - \alpha_j)\cdot r_j)}{(\beta_j - \alpha_j)\cdot r_j} - \frac{\hat{C_N}(\alpha_j \cdot r_j)}{\alpha_j \cdot r_j}\right)/(b_j - b_{j-1}), & \text{if } j \in S(T) \\ 0, & \text{otherwise.} \end{cases}$$

$$w_j = \alpha_j \cdot r_j.$$

We also set two parameters, $h_{max} = \text{max}_i h_i$, $v_{min} = \text{min}_{i \in S(T)} v_i$.

Follow the definition, we can describe the cost of the Nash flow on $\hat{G}$ with Lemma 6.

▶ **Lemma 6.** *Given a traffic network G, constants T, α_i and β_i such that $\alpha_i < 1$, $\beta_i > 1$ for $1 \leq i \leq m$, and tax(T, A, B) imposed on G,*

$$\hat{C_N}(r) = \hat{C_N}(\alpha_j \cdot r_j) + C_{Nj}(r - \alpha_j \cdot r_j) \quad \textit{for } r \in [\alpha_j \cdot r_j, \beta_j \cdot r_j] \textit{ and } j \in S(T).$$

If the Nash flow uses j links on $\hat{G}$ at rate r,

$$\hat{C_N}(r) = \frac{1}{\Lambda_j}(r^2 + r \cdot \hat{\Gamma}_j(r)) \quad \textit{for } r \notin (\alpha_j \cdot r_j, \beta_j \cdot r_j) \forall j \in S(T),$$

where $\hat{\Gamma}_j(r) = \Sigma_{i=1}^{j} \hat{b}_i(r)/\hat{a}_i = \Sigma_{i=1}^{j} \hat{b}_i(r)/a_i$. If the Nash flow uses exactly n links on G_j at rate r,

$$C_{Nj}(r) = \frac{1}{\Lambda_{j+n-1} - \Lambda_{j-1}}(r^2 + (\Gamma_{j+n-1} - \Gamma_{j-1}) \cdot r).$$

Proof. Equations can be derive directly from Lemma 2. ◀

In this paper, all tax schemes are designed in a way that after the tax is being applied, the ePoA is determined by the Nash/OPT costs at total flow $\alpha_i r_i$ or $\beta_i r_i$ for some i. In order to have a good estimate of the ePoA, we first derive Theorem 7 which gives us a good estimate of the original PoA at total flow $\alpha_i r_i$ and $\beta_i r_i$. In this theorem, the first inequality gives a good upper bound on PoA at $\beta_i r_i$ and the second inequality gives a good upper bound on the PoA at $\alpha_i r_i$. All upper bounds are described using parameters Λ_j since these values play an important role in determining the PoA [8].

▶ **Theorem 7.** *If the Nash flow uses exactly j links and the OPT uses exactly h links at rate r then*

$$PoA(r) \leq max\Big\{\frac{4r}{4r - r_{j-1}}, \frac{r^2\Lambda_j^{-1} + r \cdot r_j(\Lambda_{j-1}^{-1} - \Lambda_j^{-1})}{r^2\Lambda_{j-1}^{-1} - \Sigma_{i=j}^{h}(r - r_i/2)^2 \cdot (\Lambda_{i-1}^{-1} - \Lambda_i^{-1})}\Big\}.$$

If the Nash flow uses exactly j-1 links and the OPT uses exactly h links at rate r then

$$PoA(r) \leq max\Big\{\frac{4r}{4r - r_{j-1}}, \frac{r^2\Lambda_{j-1}^{-1}}{r^2\Lambda_{j-1}^{-1} - \Sigma_{i=j}^{h}(r - r_i/2)^2 \cdot (\Lambda_{i-1}^{-1} - \Lambda_i^{-1})}\Big\}.$$

The proof is omitted due to space constraints.

In the PoA-r curve, local maximum only exists at $r = r_j$. The following lemma gives an upper bound on PoA which will be used to show that PoA in the region $[\beta_i r_i, \alpha_{i+1} r_{i+1}]$ is bounded by the PoA of this region's two endpoints.

▶ **Lemma 8.** *Given a traffic network G, if the Nash flow uses exactly j links at rate s and t for $s < t$, then*

$$PoA(r) \leq max\{PoA(s), PoA(t)\}, \forall r \in [s, t]\,.$$

Most of our proof relies on Theorem 7 and Lemma 8, first with Lemma 8 to bound the PoA for total flow far away from the peak values r_i, then with Theorem 7 to provide a good bound for total flow close to these peak values.

As previously mentioned, the step functions that decrease PoA near r_i will increase PoA when the total flow is greater than $\beta_i r_i$. Lemma 9 shows that our tax will only increase the total cost by a constant factor.

▶ **Lemma 9.** *Given a traffic network G, constants T, α_i and β_i such that $\alpha_i < 1$, $\beta_i > 1$ for $1 \leq i \leq m$, let $\hat{G}$ be the traffic network obtained by imposing tax(T, A, B) on G, then*

$$\frac{\hat{C_N}(r)}{C_N(r)} \leq 1 + \frac{h_{max}}{v_{min}}, \quad \textit{for} \quad r \quad \textit{such that} \quad r \notin (\alpha_j \cdot r_j, \beta_j \cdot r_j) \quad \textit{for all} \quad j \in S(T).$$

Proof. For total flow $r \in [r_{j-1}, r_j]$ and $r \notin (\alpha \cdot r_i, \beta \cdot r_i)$ for all $i \in S(T)$, let k be the largest $i \in S(T)$ such that $i < j$, from Lemma 2,

$$\frac{\hat{C_N}(r)}{C_N(r)} \leq \frac{r + \hat{\Gamma}_{j-1}(r)}{r + \Gamma_{j-1}} = 1 + \frac{\sum_{i=1}^{j-1}(\hat{b}_i(r) - b_i)\lambda_i}{r + \Gamma_{j-1}}.$$

The largest possible tax added to a link when $r \in [r_{j-1}, r_j]$ is $h_{max} \cdot b_k$, and only link 1 to k-1 have non-zero tax added rate r,

$$\frac{\hat{C_N}(r)}{C_N(r)} \leq 1 + \frac{\sum_{i=1}^{k-1} h_{max} \cdot b_k \lambda_i}{r + \Gamma_{j-1}} \leq 1 + \frac{h_{max} \cdot b_k \Lambda_{k-1}}{r_{j-1} + \Gamma_{j-1}} = 1 + \frac{h_{max} \cdot b_k \Lambda_{k-1}}{b_{j-1}\Lambda_{j-1}}.$$

Since $k < j$, $b_k \leq b_{j-1}$ and $\Lambda_k \leq \Lambda_{j-1}$,

$$\frac{\hat{C_N}(r)}{C_N(r)} \leq 1 + \frac{h_{max}}{\Lambda_k/\Lambda_{k-1}} = 1 + \frac{h_{max}}{v_k} \leq 1 + \frac{h_{max}}{v_{min}}.$$ ◀

4 The ePoA for Two-Link Networks

In this section, we study the networks with two parallel links. In this special case, we give an upperbound of ePoA for the step function tolls which is 1.192. This result shows that applying step function tolls is as powerful as arbitrary tax scheme proposed in [8]. In fact, when the total flow is between 0 and βr_1, our step function tax is exactly identical to the tax scheme in [8]. When the total flow is greater than βr_1, the previous tax scheme remove the previously added step-function tax and does not impose tax on any link. In this paper, removing the step functions is not allowed. We prove that even though these step functions only increase PoA when the total flow is greater than βr_1, the influence is marginal and the maximum value always happen at total flow βr_1. The proof is omitted due to space constraints.

▶ **Theorem 10.** *Given a two link traffic network G, there always exist a pair of $\alpha \in (\frac{1}{2}, 1), \beta \in (1, \infty)$ such that if $tax(T = 1.192, \{\alpha\}, \{\beta\})$ is imposed, then $ePoA \leq 1.192$.*

5 Upperbound of the ePoA for Multiple Parallel-Link Networks

In this section we consider parallel-link networks. Given a traffic netowrk G, we consider that ratio between two adjacent peak values $\frac{r_i}{r_{i-1}}$. Let $\epsilon = \min(\min_{i>1} u_i, 2) - 1 = \min(\min_{i>1} \frac{r_i}{r_{i-1}}, 2) - 1$. We prove that the ePoA has an upper bound less than $\frac{4}{3} - \frac{1}{3}(\frac{\epsilon}{3})^3$. Notice that in this case, ϵ only depends on the discrepancy between the links and is independent of the number of links in the network. The main result of this section is the following theorem.

▶ **Theorem 11.** *Given a traffic network G, and $tax(T = \frac{4}{3} - (\frac{\epsilon}{3})^3, \{\alpha_1 = \cdots = \alpha_m = 1 - 2(\frac{\epsilon}{3})^3\}, \{\beta_1 = \cdots = \beta_m = 1 + 3(\frac{\epsilon}{3})^3\})$ is imposed. Then*

$$ePoA < \frac{4}{3} - \frac{1}{3}(\frac{\epsilon}{3})^3.$$

Notice that $\frac{r_j}{r_{j-1}}$ is less than $\frac{b_j}{b_{j-1}}$ and increases when the difference between a_j and a_{j-1} increases, and thus is a good indicator of the discrepancy between the links.

In order to prove Theorem 11, we need the following lemmas. Intuitively, we first use Lemma 14 and 15 to prove that the PoA of the original network is at most T when the total flow is αr_j of βr_j. Combining with Lemma 8 and 9, we know that ePoA$\leq T(1 + \frac{h_{max}}{v_{min}})$ for all $r \notin (\alpha r_j, \beta r_j)$. Lemma 16 shows that when the total flow is between αr_j and βr_j, the ePoA is also bounded. Plug in the value of h_{max} and v_{min} from Lemma 12 and 13 to finish the proof. For the constants T, α_i, β_i chosen, we can bound all related parameters needed in Theorem 11 with some straightforward calculations.

▶ **Lemma 12.** *Given a traffic network G, a constant T such that $T > \frac{4+4\epsilon}{3+4\epsilon}$.*

$$v_{min} = min_{i \in S(T)} v_i = min_{i \in S(T)} \frac{\Lambda_i}{\Lambda_{i-1}} > \frac{(2\epsilon - \epsilon^2)T}{4 - 3T}.$$

Proof. By definition of set S, $\mathrm{PoA}(r_i) > T$ for all $i \in S(T)$. From Theorem 7,

$$\mathrm{PoA}(r_j) \leq \max\Big\{\frac{4r_j}{4r_j - r_{j-1}}, \frac{r_j^2 \Lambda_{j-1}^{-1}}{r_j^2 \Lambda_{j-1}^{-1} - \Sigma_{j \leq i \leq h}(r_j - r_i/2)^2 \cdot (\Lambda_{i-1}^{-1} - \Lambda_i^{-1})}\Big\}.$$

Since $r_h > r_{h-1} > \cdots > r_{j+1} \geq (1+\epsilon)r_j \geq (1+\epsilon)^2 r_{j-1}$ and $\Lambda_h^{-1} > 0$,

$$\mathrm{PoA}(r_j) \leq \max\Big\{\frac{4+4\epsilon}{3+4\epsilon}, \frac{4}{3 + (2\epsilon - \epsilon^2)\Lambda_{j-1}/\Lambda_j}\Big\}.$$

From condition of T,

$$T < \mathrm{PoA}(r_j) \leq \frac{4}{3 + (2\epsilon - \epsilon^2)\Lambda_{j-1}/\Lambda_j} \quad \forall j \in S(T).$$ ◀

▶ **Lemma 13.** $h_j \leq \Big(\frac{1}{v_j - 1}(\beta_j - \alpha_j) + (1 - \alpha_j)\Big) \cdot \frac{r_j}{r_j - r_{j-1}}$.

▶ **Lemma 14.** *Given a traffic network G, constants T and α such that $\alpha \leq \frac{T + (T^2 - T)^{\frac{1}{2}}}{2}$, $\frac{4\alpha \cdot r_j}{4\alpha \cdot r_j - r_{j-1}} \leq T$ and $1 < T < \frac{4}{3}$, then $PoA(\alpha \cdot r_j) \leq T$.*

▶ **Lemma 15.** *Given a traffic network G, constants T and β such that $2 > \beta \geq \frac{T}{4(T-1)}$ and $1 < T < \frac{4}{3}$, then $PoA(\beta \cdot r_j) \leq T$.*

▶ **Lemma 16.** *Given a traffic network G, constants T, α_i and β_i such that $\alpha_i < 1$, $\beta_i > 1$ for $1 \leq i \leq m$, and $tax(T, A, B)$ imposed on G,*

$$ePoA(r) \leq max\Big\{ePoA(\alpha_j \cdot r_j), (\beta_j - \alpha_j)\frac{\Lambda_j}{\Lambda_j - \Lambda_{j-1}} + (1 - \beta_j + \alpha_j)\Big\}$$

$$for\ r \in [\alpha_j \cdot r_j, \beta_j \cdot r_j] \text{ and } j \in S(T).$$

The Proof of Lemma 13 to 16 are omitted due to space constraints.

Proof of Theorem 11. Let $\alpha = \alpha_1 = \cdots = \alpha_m = 1 - 2(\frac{\epsilon}{3})^3$, $\beta = \beta_1 = \cdots = \beta_m = 1 + 3(\frac{\epsilon}{3})^3$. First consider the case when total flow $r \notin (\alpha \cdot r_j, \beta \cdot r_j) \quad \forall j \in S(T)$. Since $\beta \cdot r_j < \alpha \cdot r_{j+1} \quad \forall j$, we can apply the result of Lemma 8,

$$\mathrm{PoA}(r) \leq \max\Big\{\max_{i \notin S(T)} \mathrm{PoA}(r_i), \max_{i \in S(T)} \mathrm{PoA}(\alpha \cdot r_i), \max_{i \in S(T)} \mathrm{PoA}(\beta \cdot r_i)\Big\}.$$

From Lemma 14, 15 and the definition of $S(T)$, all terms above are bounded by the threshold T,

$$\mathrm{PoA}(r) \leq T \text{ for } r \notin (\alpha \cdot r_j, \beta \cdot r_j) \quad \forall j \in S(T).$$

$\mathrm{ePoA}(r)$ is bounded by $\mathrm{PoA}(r)$ times the ratio between cost of the Nash flow on $\hat{G}$ and G, From Lemma 9,

$$\mathrm{ePoA}(r) = \mathrm{PoA}(r) \cdot \frac{\hat{C_N}(r)}{C_N(r)} \leq T(1 + \frac{h_{max}}{v_{min}}) \quad \text{for} \quad r \notin (\alpha \cdot r_j, \beta \cdot r_j) \quad \forall j \in S(T). \tag{2}$$

We then consider $\mathrm{ePoA}(r)$ when total flow $r \in [\alpha \cdot r_j, \beta \cdot r_j]$, and $j \in S(T)$. From Lemma 16,

$$\mathrm{ePoA}(r) \leq \max\Big\{\mathrm{ePoA}(\alpha \cdot r_j), (\beta - \alpha)\frac{\Lambda_j}{\Lambda_j - \Lambda_{j-1}} + (1 - \beta + \alpha)\Big\}.$$

For the second term above, since $j \in S(T)$, the ratio of Λ_j and Λ_{j-1} is bounded, from Lemma 12,

$$\begin{aligned}
&(\beta-\alpha)\frac{\Lambda_j}{\Lambda_j-\Lambda_{j-1}}+(1-\beta+\alpha)\\
\leq\,&(\beta-\alpha)\frac{v_{min}}{v_{min}-1}+(1-\beta+\alpha)\\
\leq\,&(5(\frac{\epsilon}{3})^3)\cdot\frac{(2\epsilon-\epsilon^2)(\frac{4}{3}-(\frac{\epsilon}{3})^3)}{(2\epsilon-\epsilon^2)(\frac{4}{3}-(\frac{\epsilon}{3})^3)-3(\frac{\epsilon}{3})^3)}+(1-5(\frac{\epsilon}{3})^3)\\
<\,&\frac{4}{3}-(\frac{\epsilon}{3})^3=T.
\end{aligned}$$

From previous case, we know that $\text{ePoA}(\alpha\cdot r_j)\leq T(1+\frac{h_{max}}{v_{min}})$, therefore

$$\text{ePoA}(r)\leq T(1+\frac{h_{max}}{v_{min}})\quad\text{for}\quad r\in[\alpha\cdot r_j,\beta\cdot r_j]\quad\text{if}\quad j\in S(T). \tag{3}$$

Combine (2) and (3), we have an upperbound of $\text{ePoA}(r)$ for all $r>0$,

$$\text{ePoA}\leq T(1+\frac{h_{max}}{v_{min}}).$$

From Lemma 12 and 13,

$$\text{ePoA}<\frac{4}{3}-\frac{\epsilon^3}{27}\Big(1-(\frac{2(2\epsilon-\epsilon^2)(\frac{4}{3}-\frac{\epsilon^3}{27})+3\epsilon^3/9}{(2\epsilon-\epsilon^2)(\frac{4}{3}-\frac{\epsilon^3}{27})-\epsilon^3/9})\cdot\frac{\epsilon+\epsilon^2}{9(2-\epsilon)}\Big)<\frac{4}{3}-\frac{\epsilon^3}{81}. \qquad ◀$$

6 Networks with Groups of Similar Links

In previous sections, we have given an upperbound of ePoA when it is strictly less than $\frac{4}{3}$. In this section, we study a special case in which the links can be classified int many groups. Links in the same group all have similar r_i and thus similar cost functions. This special case is closely related to the case in which there are many types of transportation methods, or just many types of roads (such as freeways and local roads). We give an upper bound of ePoA for a specific case of groups of similar link defined below.

▶ **Definition 17.** A traffic network G_c is a *network with clustered latencies* if and only if there exists N intervals $[L_1,R_1],\ldots,[L_N,R_N]$, and $\frac{R_i}{L_i} <= 1.05$ for $i\in[1,2,\ldots,N]$, and $\frac{L_{i+1}}{R_i}\geq 20$ and any r_j for $j\geq 2$ is in one of the intervals $[L_i,R_i]$.

The main result of this section is ePoA≤ 1.281 for a traffic network G_c with clustered latencies. Before proving the main result, we introduce the following transformation, and several lemmas.

▶ **Definition 18.** Given any traffic network G_c with clustered latencies, we define the *aggregated network of* G_c, G_a as the following. For all r_i in G_c, inside a certain interval $[L_k,R_k]$, we re-label the index i to be $k_1,k_2,\ldots,k_{n_k}$ so that $L_k\leq r_{k_1}\leq r_{k_2}\leq\cdots\leq r_{k_{n_k}}\leq R_k$. An intermediate network G_{temp} is obtained by increasing the constant of the cost functions b_{k_i} to $\tilde{b_{k_i}}=b_{k_{n_k}}$ for all $i<n_k$. Now all links e_i with r_i in the same interval in G_c has the same b-value, which is $b_{k_{n_k}}$. Thus, by Lemma 3, these links can be merged through a transformation of graph without changing either the Nash flow or the OPT. After the merge, the resulting network is G_a. The transformation Tr is the combination of increasing the constants of links in G_c to get G_{temp}, and merging edges of G_{temp} to get G_a.

▶ **Lemma 19.** *In a traffic networks G_1 with r_{j-1} and r_j where , b_{j-1} is increased to b_j, and the two links are merged to index i_{new}, as stated in Definition 18 then the position of $r_{i_{new}}$ is betweeen (r_{j-1}, r_j).*

Proof. By the basic equation in Definition 1, $r_j = \Sigma_{i=1}^{j-1}(b_{i+1} - b_i)\Lambda_i$.

$$\begin{aligned} r_j - r_{i_{new}} &= (b_j - b_{j-1})(\Lambda_{j-1} - \Lambda_{j-2}) > 0 \\ r_{i_{new}} - r_{j-1} &= (b_j - b_{j-1})(\Lambda_{j-1}) > 0 \end{aligned}$$

Thus, $r_{j-1} \leq r_{i_{new}} \leq r_j$. ◀

Following Definition 18, with Lemma 19 used recursively, we see that the resulting r_k after merging all links in section k lies in $[L_k, R_k]$. Therefore, after the transformation, the resulting traffic network G_a has min $\frac{r_{j+1}}{r_j} \geq 20$, which is directly from the fact that $\frac{L_{i+1}}{R_i} \geq 20$. The ratio of the optimal cost between the network after the transformation and before the transformation is less than the ratio of the largest $\frac{\hat{b_i}}{b_i} \leq 1.05$. The formal lemma and proof are below.

▶ **Lemma 20.** *For any traffic network G_c with clustered latencies and its corresponding aggregated network G_a, $\frac{C_{opt}(r,G_a)}{C_{opt}(r,G_c)} \leq 1.05$.*

The following lemma is similar to Lemma 9, for a slightly different situation.

▶ **Lemma 21.** *In a traffic network G with rate r and $\frac{r_{i+1}}{r_i} \geq 20$, where constants T, $\alpha_i < 1$, $\beta_i > 1$. When $tax(T, A, B)$ is imposed on the network G. We have*

$$\frac{\hat{C_N}(r)}{C_N(r)} \leq 1 + \frac{h_{max}}{20 \times s} \tag{4}$$

for any s satisfying $\alpha_j \times r_j \geq r \geq s \times r_j$, and $s \times r_j \geq \beta_{j-1} r_{j-1}$, $s \leq 1$. Similarly, we have

$$\frac{\hat{C_N}(r)}{C_N(r)} \leq 1 + \frac{h_{max}}{v_j} \tag{5}$$

for $\alpha_{j+1} \times r_{j+1} \geq r \geq \beta_j \times r_j$.

We now introduce the tax scheme and upper bound the corresponding ePoA for an aggregated network. The tax scheme chooses different values of α_i, β_i, with different regions of v_i.

▶ **Lemma 22.** *For any traffic network G_c with clustered latencies and its corresponding aggregated network G_a, there exists a tax scheme G_a such that ePoA of $G_a \leq 1.22$ when the tax is applied to G_a.*

Proof. The tax scheme tax(1.198, A, B), where α_j , β_j are decided according to the value of v in Table 1 satisfies the requirement.

The proof is omitted due to space constraints. ◀

With Lemma 20 and 22 we prove the main theorem in this section by simply multiplying 1.22 and 1.05.

▶ **Theorem 23.** *The ePoA is at most 1.281 for a traffic network with clustered latencies.*

Table 1 Corresponding α, β with different values of v, note that tax is not imposed in case 0.

cases	v_j	α_j	β_j
0	$[0, 2.95]$	any	any
1	$[2.95, 3]$	0.985	1.51
2	$[3.0, 3.2]$	0.981	1.51
3	$[3.2, 3.5]$	0.964	1.51
4	$[3.5, 4.0]$	0.9428	1.5108
5	$[4.0, 4.8]$	0.9175	1.524
6	$[4.8, 7.0]$	0.89	1.55
7	$[7.0, 11.0]$	0.87	1.79
8	$[11.0, \infty]$	0.83	1.90

Proof. For any traffic network G_c with clustered latencies and its corresponding aggregated network G_a, with Lemma 20 and 22 we know that ePoA of $G_a \leq 1.22$, and $\frac{C_{opt}(r,G_a)}{C_{opt}(r,G_c)} \leq 1.05$. We view the transformation Tr on G_c as tax T_1, and the tax imposed on G_a as tax T_2. The final tax scheme imposed on G_c is $T_1 + T_2$. While the tax scheme imposed on G_a is T_2. Now we prove the theorem

$$\begin{aligned} ePoA &= \frac{\hat{C}_N(r, G_c)}{C_{opt}(r, G_c)} \\ &= \frac{\hat{C}_N(r, G_a)}{C_{opt}(r, G_a)} \cdot \frac{C_{opt}(r, G_a)}{C_{opt}(r, G_c)} \\ &\leq 1.22 \cdot 1.05 = 1.281 \end{aligned}$$ ◀

A point to be noted is that the lower bound of PoA is proved in [8] to be 1.191 for two edge network, therefore that proving ePoA ≤ 1.22 is clearly close to optimal since additional tax is further accounted while in [8] the tax could be retrieved and that $\frac{r_{j+1}}{r_j}$ is ∞, where in Lemma 22 the restriction is much stricter, while only increasing the ePoA by less than 3 percent.

7 Open Problems

The goal of this work is to design a taxing scheme with unit step function which is able to be applied to general networks. In the case of parallel links in our study, we have demonstrated different possible approaches to bound the ePoA. We have proved a tight upperbound of the two link case in Section 4, given an upperbound of ePoA less than $\frac{4}{3}$ depending on the discrepancy between links in Section 5, and give an upperbound when the links are clustered while the discrepancy between links in each cluster are not limited in Section 6. However, it remains an open question whether there is a upperbound less that $\frac{4}{3}$ independent of both the discrepancy between the links and the number of links in the network. A combination of the previous methods could be a possible approach.

References

1 Yossi Azar, Lisa Fleischer, Kamal Jain, Vahab Mirrokni, and Zoya Svitkina. Optimal coordination mechanisms for unrelated machine scheduling. *Operations Research*, 2015.

2 Martin Beckmann, C. B. McGuire, and Christopher B. Winsten. Studies in the economics of transportation. Technical report, 1956.

3 David Bernstein and Tony E. Smith. Equilibria for networks with lower semicontinuous costs: With an application to congestion pricing. *Transportation Science*, 28(3):221–235, 1994.

4 Vincenzo Bonifaci, Mahyar Salek, and Guido Schäfer. *Efficiency of restricted tolls in non-atomic network routing games.* Springer, 2011.

5 Ioannis Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. *Algorithmica*, 66(3):512–540, 2013.

6 Ho-Lin Chen and Tim Roughgarden. Network design with weighted players. *Theory of Computing Systems*, 45(2):302–324, 2009.

7 George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. *Theoretical Computer Science*, 410(36):3327–3336, 2009.

8 Giorgos Christodoulou, Kurt Mehlhorn, and Evangelia Pyrga. Improving the price of anarchy for selfish routing via coordination mechanisms. *Algorithmica*, 69(3):619–640, 2014.

9 Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. How much can taxes help selfish routing? In *Proceedings of the 4th ACM conference on Electronic commerce*, pages 98–107. ACM, 2003.

10 Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. Pricing network edges for heterogeneous selfish users. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 521–530. ACM, 2003.

11 Lisa Fleischer. Linear tolls suffice: New bounds and algorithms for tolls in single source networks. *Theoretical Computer Science*, 348(2):217–225, 2005.

12 Lisa Fleischer, Kamal Jain, and Mohammad Mahdian. Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 277–285. IEEE, 2004.

13 Randolph Hall. *Handbook of transportation science*, volume 23. Springer Science & Business Media, 2012.

14 Nicole Immorlica, Li Erran Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. *Theoretical Computer Science*, 410(17):1589–1598, 2009.

15 George Karakostas and Stavros G. Kolliopoulos. Edge pricing of multicommodity networks for heterogeneous selfish users. In *FOCS*, volume 4, pages 268–276, 2004.

16 George Karakostas and Stavros G. Kolliopoulos. The efficiency of optimal taxes. In *Combinatorial and Algorithmic Aspects of Networking*, pages 3–12. Springer, 2005.

17 Konstantinos Kollias. Non-preemptive coordination mechanisms for identical machine scheduling games. In *Structural Information and Communication Complexity*, pages 197–208. Springer, 2008.

18 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *STACS 99*, pages 404–413. Springer, 1999.

19 Michael Patriksson. *The Traffic Assignment Problem: Models and Methods.* Courier Dover Publications, 2015.

20 Arthur Cecil Pigou. *The economics of welfare.* Palgrave Macmillan, 2013.

21 Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in internet-like environments. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 151–162. ACM, 2003.

22 Tim Roughgarden. The price of anarchy is independent of the network topology. *Journal of Computer and System Sciences*, 67(2):341–364, 2003.

23 Tim Roughgarden. *Selfish routing and the price of anarchy*, volume 174. MIT press Cambridge, 2005.

24 Tim Roughgarden. On the severity of braess's paradox: designing networks for selfish users is hard. *Journal of Computer and System Sciences*, 72(5):922–953, 2006.

25 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

A Linear-Time Algorithm for Integral Multiterminal Flows in Trees[*]

Mingyu Xiao[1] and Hiroshi Nagamochi[2]

1 School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
myxiao@gmail.com

2 Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Japan
nag@amp.i.kyoto-u.ac.jp

Abstract

In this paper, we study the problem of finding an integral multiflow which maximizes the sum of flow values between every two terminals in an undirected tree with a nonnegative integer edge capacity and a set of terminals. In general, it is known that the flow value of an integral multiflow is bounded by the cut value of a cut-system which consists of disjoint subsets each of which contains exactly one terminal or has an odd cut value, and there exists a pair of an integral multiflow and a cut-system whose flow value and cut value are equal; i.e., a pair of a maximum integral multiflow and a minimum cut. In this paper, we propose an $O(n)$-time algorithm that finds such a pair of an integral multiflow and a cut-system in a given tree instance with n vertices. This improves the best previous results by a factor of $\Omega(n)$. Regarding a given tree in an instance as a rooted tree, we define $O(n)$ rooted tree instances taking each vertex as a root, and establish a recursive formula on maximum integral multiflow values of these instances to design a dynamic programming that computes the maximum integral multiflow values of all $O(n)$ rooted instances in linear time. We can prove that the algorithm implicitly maintains a cut-system so that not only a maximum integral multiflow but also a minimum cut-system can be constructed in linear time for any rooted instance whenever it is necessary. The resulting algorithm is rather compact and succinct.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Multiterminal flow, Maximum flow, Minimum Cut, Trees, Linear-time algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.62

1 Introduction

The min-cut max-flow theorem by Ford and Fulkerson [5] is one of the most important theorems in graph theory. It catches a min-max relation between two fundamental graph problems. This theorem leads to many effective algorithms and much theory for flow problems as well as graph cut problems. Due to the great applications of it, researchers have interests to seek more similar min-max formulas in various kinds of flow and cut problems. In this paper, we consider the *maximum multiterminal flow problem*, a generalization of the basic maximum flow problem.

[*] A full version of the paper is available at https://arxiv.org/abs/1611.08803.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 62; pp. 62:1–62:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the maximum flow problem, we are given two terminals (source and sink) and asked to find a maximum flow between the two terminals. A natural generalization of the maximum flow problem is the famous *maximum multicommodity flow problem*, in which, a list of pairs of source and sink for the commodities is given and the objective is to maximize the sum of the simultaneous flows in all the source-sink pairs subject to the standard capacity and flow conservation requirements. The maximum multiterminal flow problem is one of the most important special cases of the maximum multicommodity flow problem. In it, a set T of more than one terminal is given and the list of source-sink pairs is given by all pairs of terminals in T. The extensions of the maximum flow problem have been extensively studied in the history. Readers are referred to a survey [2].

A dual problem of the maximum multiterminal flow problem is the *minimum multiterminal cut problem*, in which we are asked to find a minimum set of edges whose removal disconnects each pair of terminals in the graph. The minimum multiterminal cut problem is a generalization of the minimum cut problem. When there are only two terminals, the min-cut max-flow theorem shows that the value of the maximum flow equals to the value of the minimum cut in the graph. However, when there are more than two terminals, the equivalence may not hold. Consider a star with three leaves. Each leaf is a terminal and each of the three edges has capacity 1. The flow value of a maximum multiterminal flow is 1.5 (a flow of size 0.5 routed between every pair of the three terminal pairs), whereas the size of a minimum multiterminal cut is 2. In fact, Cunningham [4] has proved a min-max theory for the pair of problems: The size of a minimum multiterminal cut is at most $(2 - 2/|T|)$ times of the flow value of a maximum multiterminal flow. A similar min-max theory for the maximum multicommodity flow problem and its dual problem is presented in [6].

In the maximum multiterminal flow problem, each edge is assigned a nonnegative capacity and a flow routed between a terminal pair is allowed to take any feasible fraction, whereas in the *integral multiterminal flow problem*, a flow is allowed to take a nonnegative integer and we are asked to find a maximum flow under this restriction. Clearly, we can simply assume that all edge capacities of the integral multiterminal flow problem are nonnegative integers. The integral multiterminal flow problem is different from the maximum multiterminal flow problem. We can see in the above example, the flow value of a maximum integral multiterminal flow is 1. The special case of the integral multiterminal flow problem where all edges have unit capacities is also known as the *T-path problem*, in which we are asked to find the maximum number of edge-disjointed paths between different terminal pairs.

In this paper, we study the maximum multiterminal flow problem in trees and give linear-time algorithms for both fractional and integer versions, which improve the best previous algorithms by a factor of $\Omega(n)$ [3]. Note that the maximum (integral) multicommodity flow problem in trees is NP-hard and there is a $\frac{1}{2}$-approximation algorithm for it [7].

The rest of the paper is organized as follows. Section 2 introduces basic notations on flows and cuts, and reviews important min-max theorems for fractional and integer versions of maximum multiterminal flow problem. Section 3 discusses instances with rooted trees, and introduces notations necessary to build a dynamic programming method over the set of $O(n)$ instances of rooted subtrees of a given instance. Informally "a blocking flow" in a rooted tree instance is defined to be a flow in the tree currently pushing maximal flows among terminals except for the terminal designated as the root. Section 4 shows several properties of blocking flows, and presents a representation of flow values of blocking flows. Section 5 provides a main technical lemma that tells how to compute the representation of flow values of blocking flows and how to construct a maximum flow from the representations. Based on the lemma, Section 6 gives a description of a linear-time algorithm for computing the representations of flow values of blocking flows and constructing a maximum flow from the representations. Finally Section 7 makes some concluding remarks. The proofs of some lemmas are omitted due to the space limitation.

2 Preliminaries

This section introduces basic notations on flows and cuts, and reviews important min-max theorems for fractional and integer versions of maximum multiterminal flow problem. Let $\Re^+$ denote the set of nonnegative reals, and $\mathbb{Z}^+$ denote the set of nonnegative integers.

Graphs and Instances

We may denote by $V(G)$ and $E(G)$ the sets of vertices and edges of an undirected graph G, respectively. Let $G = (V, E)$ denote a simple undirected graph with a vertex set V and an edge set E, and let n and m denote the number of vertices and edges in a given graph. Let $X \subseteq V$ be a subset of vertices in G. Let $E(X)$ denote the set of edges with one end-vertex in X and the other in $V - X$, where $E(\{v\})$ for a vertex $v \in V$ is denoted by $E(v)$. Let $G - X$ denote the graph obtained from G by removing the vertices in X together with the edges in $\cup_{v \in X} E(v)$. For a vertex subset T, let $\mathcal{P}(T)$ be the set of all paths $P_{t,t'}$ with end-vertices $t, t' \in T$ with $t \neq t'$.

An instance I of a maximum flow problem consists of a graph G, a set T of vertices called terminals, and a capacity function $c : E \to \Re^+$.

Flows

For a function $h : E \to \Re^+$, $\sum_{e \in E(X)} h(e)$ for a subset $X \subseteq V$ is denoted by $h(X)$. A function $f : E \to \mathbb{Z}^+$ is called a *flow* in an instance (G, T, c) if there is a function $g : \mathcal{P}(T) \to \mathbb{Z}^+$ such that

$$f(e) = \sum \{g(P) \mid e \in E(P),\ P \in \mathcal{P}(T)\} \quad \text{for all edges } e \in E,$$

where $g(P)$ is the flow value sent along path P, and such a function g is called a *decomposition* of a flow f. A flow f is called *integer* if it admits a decomposition g such that $g(P) \in \mathbb{Z}^+$ for all paths $P \in \mathcal{P}(T)$ (note that f may not be integer even if $f(e) \in \mathbb{Z}^+$ for all edges $e \in E$).

A flow f is called *feasible* if $f(e) \le c(e)$ for all edges $e \in E$. The *flow value* $\alpha(f)$ is defined to be $\frac{1}{2}\sum_{t \in T} f(\{t\})$, and a feasible flow f that maximizes $\alpha(f)$ is called *maximum*.

Cut-Systems

A subset X of vertices is called a *terminal set* (or a *t-set*) if $X \cap T = \{t\}$ and X induces a connected subgraph from G. A *cut-system* of T is defined to be a collection $\mathcal{X}$ of disjoint $|T|$ terminal sets X_t, $t \in T$, where $\mathcal{X}$ is not required to be a partition of V. For a cut-system $\mathcal{X}$ of T, let $\gamma(\mathcal{X}) = \sum_{X \in \mathcal{X}} c(X)$. For any pair of a feasible flow f and a cut-system $\mathcal{X}$ of T in (G, T, c), it holds

$$\alpha(f) \le \frac{1}{2}\gamma(\mathcal{X}). \tag{1}$$

Cherkasskii [1] proved the next result.

▶ **Theorem 1.** *A feasible flow f in (G, T, c) is maximum if and only if there is a cut-system $\mathcal{X}$ such that $\alpha(f) = \frac{1}{2}\gamma(\mathcal{X})$.*

Ibaraki *et al.* [9] proposed an $O(nm \log n)$-time algorithm for computing a maximum flow f in a graph G with n vertices and m edges. Hagerup *et al.* [8] proved a characterization of the maximum multiterminal flow problem and gave an $O(\mathrm{ex}(|T|)n)$-time algorithm for the maximum multiterminal flow problem in bounded treewidth graphs, where $\mathrm{ex}(|T|)$ is an exponential function of the number $|T|$ of terminals. This algorithm runs in linear time only when $|T|$ is restricted to a constant.

An integer version of the multiterminal flow problem is defined as follows. Let $I = (G = (V, E), T, c)$ have integer capacities $c(e) \in \mathbb{Z}^+$, $e \in E$. Recall that an integral flow f is a flow which can be decomposed into integer individual flows g, i.e., $g : \mathcal{P}(T) \to \mathbb{Z}^+$. An instance (G, T, c) is called *inner-eulerian* if all edge capacities $c(e)$, $e \in E$ are integers and $c(E(v))$ is an even integer for each non-terminal vertex $v \in V - T$. It is known that any inner-eulerian instance admits a pair of a maximum integral flow f and a cut-system $\mathcal{X}$ with $\alpha(f) = \frac{1}{2}\gamma(\mathcal{X})$ [1]. In general, there is no pair of an integral flow f and a cut-system $\mathcal{X}$ with $\alpha(f) = \frac{1}{2}\gamma(\mathcal{X})$ even for trees. We review a min-max theorem on the integer version as follows.

Assume that $c(e) \in \mathbb{Z}^+$, $e \in E$. A component $W \subseteq V$ in the graph $G - \cup_{X \in \mathcal{X}} X$ is called an *odd set* in $\mathcal{X}$ if $c(W)$ is odd. Let $\kappa(\mathcal{X})$ denote the number of odd sets in $G - \cup_{X \in \mathcal{X}} X$. For each odd set W, at least one unit of capacity from $c(W)$ cannot be used by any feasible integral flow $f : E \to \mathbb{Z}^+$. Hence since each path in $\mathcal{P}(T)$ goes through edges in $E(X_t)$ of a t-set for exactly two terminals $t \in T$, we see that, for any decomposition g of f,

$$2\alpha(f) = \sum_{P \in \mathcal{P}(T)} g(P) \leq \sum_{X \in \mathcal{X}} c(X) - \kappa(\mathcal{X}) = \gamma(\mathcal{X}) - \kappa(\mathcal{X}). \tag{2}$$

Mader [10] proved the next result.

▶ **Theorem 2.** *A feasible integral flow f in (G, T, c) is maximum if and only if there is a cut-system $\mathcal{X}$ such that $\alpha(f) = \frac{1}{2}[\gamma(\mathcal{X}) - \kappa(\mathcal{X})]$.*

For trees with n vertices, an $O(n^2)$-time algorithm for computing a maximum integral flow f is proposed [3], while no strongly-polynomial time algorithm is known to general graphs (e.g., see [2]).

3 Tree Instances

In the rest of this paper, we assume that a given instance $I = (G, T, c)$ consists of a tree $G = (V, E)$, a terminal set T and an integer capacity $c(e) \in \mathbb{Z}^+$ for each $e \in E$. We simply call an integral flow a *flow*.

This section discusses instances with rooted trees, and introduces notations necessary to build a dynamic programming method over the set of $O(n)$ instances of rooted subtrees of a given instance.

If a vertex $v \in T$ is not a leaf of G, i.e., v is of degree $d \geq 2$, then we can split the instance at the cut-vertex v into d instances, and it suffices to find a maximum flow in each of these instances. Also we can split a vertex $v \in V - T$ of degree $d \geq 4$ into $d - 2$ vertices that induce a tree with edges of capacity sufficiently larger without losing the feasibility and optimality of the instance. In the rest of paper, we assume that T is the set of leaves of G, and the degree of each non-leaf is 3, and $c(e) \geq 1$ for all edges $e \in E$, as shown in Fig. 1.

For a leaf $v \in V$ in G, let e_v denote the edge incidenet to v. For two vertices $u, v \in V$, let $P_{u,v}$ denote the path connecting u and v in the tree G. For a subset $S \subseteq V$ of vertices, let $\mathcal{P}(S)$ denote the set of all paths $P_{s,s'}$ with $s, s' \in S$.

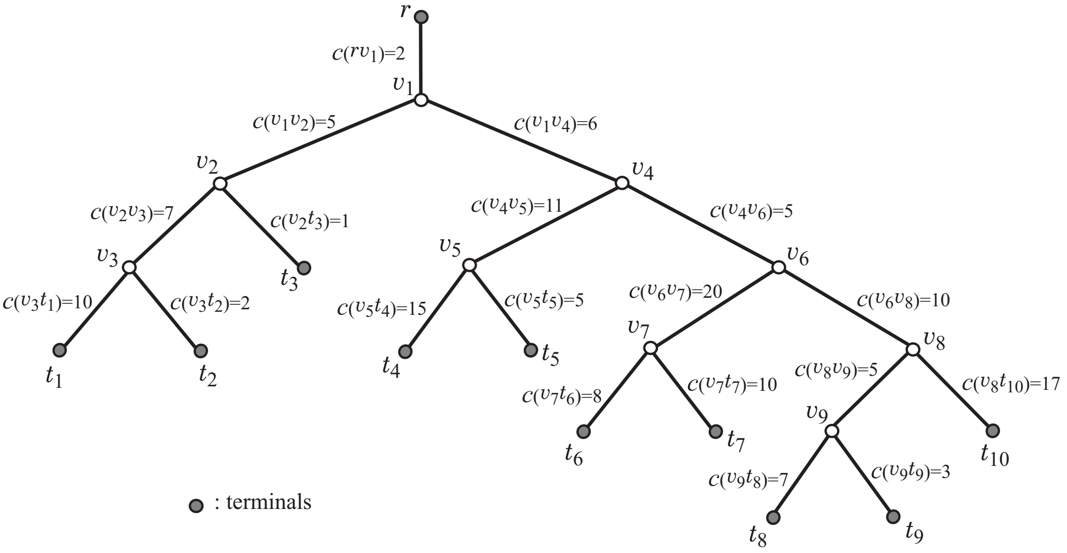


Figure 1 An example of a tree instance $I = (G, T, c)$ such that the degree of each internal vertex is 3 and all capacities are positive integers, where terminal r is chosen as the root.

In a tree instance (G, T, c), a *flow* admits a function $g : \binom{T}{2} \to \mathbb{Z}^+$ such that

$$f(e) = \sum\{g(t, t') \mid e \in E(P_{t,t'}),\ t, t' \in T\} \quad \text{for all edges } e \in E,$$

where $g(t, t')$ is the flow value sent along path $P_{t,t'}$. For a flow f, a path $P \in \mathcal{P}(T)$ is called a *positive-path* if f admits a decomposition g such that $g(t, t') > 0$.

For a path P in G, and an integer $\delta \geq -\min_{e' \in E} h(e')$ (possibly $\delta < 0$), the function $h' : E \to \mathbb{Z}^+$ obtained from h by setting $h'(e) = h(e) + \delta$ for all edges $e \in E(P)$ and $h'(e) = h(e)$ for all edges $e \in E - E(P)$ is denoted by $h + (P, \delta)$.

Rooted Tree

Choose a terminal $r \in T$, and regard G as a tree rooted at r, which defines a parent-child relationship among the vertices in G. In a rooted tree G, we write an edge $e = uv$ such that u is the parent of v by an ordered pair (u, v). For an edge $e = (u, v)$, any edge $e' = (v, w)$ is called a *child-edge* of e, and e is called the *parent-edge* of e'.

Let Y be a subset of vertices in $V - \{r\}$ such that Y induces a connected subgraph from G. Then there is exactly one edge $(u, v) \in E(Y)$ such that $v \in Y$ and u is the parent of v, and we call the edge uv the *parent-edge* of Y while any other edge in $E(Y)$ is called a *child-edge* of Y.

For an edge $e = (u, v) \in E$, let $V_e \subseteq V$ denote the set of vertex u and all the descendants of v including v itself, $G_e = (V_e, E_e)$ denote the graph induced from G by V_e, and let $T_e = (T \cap V_e) - \{u\}$, where we remark that $u \notin T_e$. Let $I(e)$ denote an instance $(G_e, T_e \cup \{u\}, c)$ induced from (G, T, c) by the vertex subset V_e, where we remark that u is included as a terminal in the instance $I(e)$.

Blocking Flows

Informally "a blocking flow" in a rooted tree instance is defined to be a flow in the tree currently pushing maximal flows among terminals except for the terminal designated as

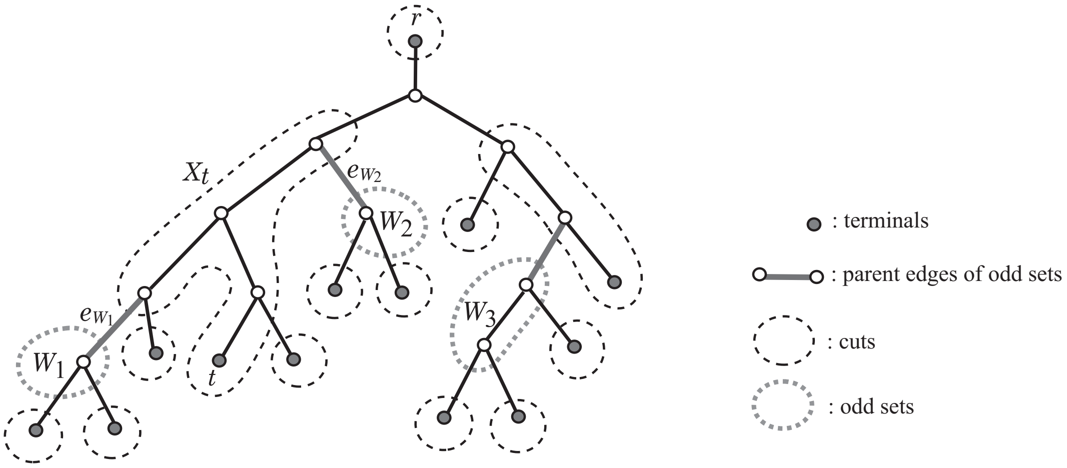


Figure 2 Illustration of a cut-system $\mathcal{X}$ and the family $\text{odd}(X_t) = \{W_1, W_2\}$ for a terminal set $X_t \in \mathcal{X}$.

the root. Let $\mathcal{X}$ be a cut-system of T_e in $I(e)$ for some edge $e = (u, v)$. An odd set W in $G_e - \cup_{X \in \mathcal{X}} X$ is called an *odd set* of a terminal set $X \in \mathcal{X}$ if the parent-edge of W is a child-edge of X, where $u \notin X$ implies $r, u \notin W$. For each terminal set $X \in \mathcal{X}$, let $\text{odd}(X)$ denote the family of odd sets of X, i.e., W of $\mathcal{X}$ whose parent-edge e_W is a child-edge of X. Fig. 2 illustrates a cut-system $\mathcal{X}$ and the family $\text{odd}(X_t) = \{W_1, W_2\}$.

For a function $h : E \to \Re_+$, let $E[h; k]$ denote the set of edges $e \in E$ such that $h(e) \geq k$.

Let f be a feasible flow of $I(e)$ for an edge $e = (u, v)$. We call a terminal set $X \in \mathcal{X}$ with $t \in X \cap T$ *blocked* (or *blocked by* f) if

$$f(e_t) = f(X) = c(X) - |\text{odd}(X)|,$$

and call $\mathcal{X}$ *blocked* (or *blocked by* f) if all terminal sets in it are blocked by f.

For each vertex $s \in V_e$, we define $V_f(s)$ to be the set of vertices $w \in V_e$ reachable from s by a path $P_{s,w'}$ from s to the common ancestor w' of s and w using edges in $E[c - f; 1]$ and by a path $P_{w',w}$ from w' to w using edges in $E[c - f; 2]$. In other words, we travel an edge e' upward if $c(e') - f(e') \geq 1$ and downward if $c(e') - f(e') \geq 2$ from s to w. By the definition of $V_f(s)$, we can see that $V_f(s)$ induces a connected subgraph, the parent-edge e' of $V_f(s)$ satisfies $f(e') = c(e')$, and any child-edge e' of $V_f(s)$ satisfies $f(e') \in \{c(e') - 1, c(e')\}$.

We call f *blocking* if $\{V_f(t) \mid t \in T_e\}$ is a cut-system of T_e blocked by f. Let $\Psi(e)$ denote the set of integers x such that $I(e)$ has a blocking flow $f(e) = x$.

Interval Computation

Our dynamic programming approach to compute the maximum flow value updates the set of flow values of blocking flows recursively. As it will be shown in Section 4, such a set of flow values always is given by an interval that consists of consecutive odd or even integers, and we here introduce a special operation on such types of intervals.

For two reals a, b with $a \leq b$, let $[a, b]$ denote the set of reals s with $a \leq s \leq b$.

For two integers $k, a \in \mathbb{Z}^+$, the set $\{a + 2i \mid i = 0, 1, \ldots, k\}$ of consecutive odd or even integers is denoted by $\langle a, b \rangle$, where $b = 2k + a$. For two sets $A, B \subseteq \mathbb{Z}^+$ of nonnegative integers, let $A \otimes B$ denote the set of nonnegative integers $\{a + b - 2i \mid i = 0, 1, \ldots, \min\{a, b\}\}$ over all $a \in A$ and $b \in B$. In particular, for sets $A_1 = \langle a_1, b_1 \rangle$ and $A_2 = \langle a_2, b_2 \rangle$, we observe that

$$A_1 \otimes A_2 = \begin{cases} \langle 0, b_1 + b_2 \rangle & \text{if } A_1 \cap A_2 \neq \emptyset \\ \langle 1, b_1 + b_2 \rangle & \text{if } a_2 \leq b_1,\ a_1 \leq b_2 \text{ and } A_1 \cap A_2 = \emptyset \\ \langle a_1 - b_2, b_1 + b_2 \rangle & \text{if } b_2 < a_1 \\ \langle a_2 - b_1, b_1 + b_2 \rangle & \text{if } b_1 < a_2. \end{cases}$$

Given an integer $x \in A_1 \otimes A_2$, we can find in $O(1)$ time three integers $x_i \in \langle a_i, b_i \rangle$, $i = 1, 2$ and $y \in [0, \min\{x_1, x_2\}]$ such that $x = x_1 + x_2 - 2y$. To see this, assume that $b_1 \leq b_2$ without loss of generality, and let a_2' be the minimum element in $\langle a_2, b_2 \rangle$ with $b_1 \leq a_2'$, where $a_2' \in \{b_1, b_1 - 1, a_2\}$. Observe that $\{x \in A_1 \otimes A_2 \mid x \leq b_2 - b_1\} = \{b_1 + x_2 - 2b_1 \mid x_2 \in \langle a_2', b_2 \rangle\}$ and $\{x \in A_1 \otimes A_2 \mid x > b_2 - b_1\} = \{b_1 + b_2 - 2y \mid y = 0, 1, \ldots, b_1 - 1\}$. Hence if $x \leq b_2 - b_1$ then let $x_1 = y = b_1$ and $x_2 = x + b_1$; otherwise $x_1 = b_1$, $x_2 = b_2$ and $y = (x - b_1 - b_2)/2$.

4 Basic Properties on Blocking Flows

This section shows several properties of blocking flows, and presents a representation of flow values of blocking flows. We first observe two lemmas on some properties of blocking flows.

▶ **Lemma 3.** *Let f be a feasible flow in $I(e)$ for an edge $e \in E$.*

(i) *For a terminal $t \in T_e$, let X_t be a t-cut such that $f(X_t) = f(e_t)$ and $P_{s,s'}$ be a positive-path of f with $s, s' \in T_e \cup \{u\}$. If $t \in \{s, s'\}$ then $P_{s,s'}$ contains exactly one edge in $E(X_t)$, and otherwise $P_{s,s'}$ is disjoint with X_t.*

(ii) *Assume that $V_f(t) \cap V_f(t') = \emptyset$ for any two $t, t' \in T_e$. Then $V_f(u)$ is disjoint with $V_f(t)$ of any terminal $t \in T_e$, and the following holds:*

(1) *For each edge $e' \in E(V_f(t))$ with $t \in T_e \cup \{u\}$,*

$$f(e') = \begin{cases} c(e') - 1 & \text{if } e' \text{ is the parent-edge of an odd set } W \in \mathrm{odd}(V_f(t)) \\ c(e') & \text{otherwise.} \end{cases}$$

(2) *$f(V_f(t)) = c(V_f(t)) - |\mathrm{odd}(V_f(t))|$ for each $t \in T_e \cup \{u\}$.*

(iii) *Flow f is blocking if $V_f(t) \cap V_f(t') = \emptyset$ for any two $t, t' \in T_e$, and $f(e_t) = f(V_f(t))$ for each $t \in T_e$.*

(iv) *When f is blocking, any edge $e' \in E_e$ with $f(e') = c(e')$ satisfies $c(e') \in \Psi(e')$.*

(v) *When f is blocking, the parent-edge e_W of any odd set $W \in \mathrm{odd}(V_f(t))$ for a terminal $t \in T_e$ satisfies $c(e_W) - 1 \in \Psi(e')$.*

A proof of this lemma can be found in the full version of this paper.

The next lemma tells how to obtain a maximum flow and a minimum cut-system in an instance $I(e)$.

▶ **Lemma 4.** *For an edge $e = (u, v) \in E$, let f be a blocking flow in $I(e)$ such that $f(e)$ is the maximum in $\Psi(e)$. Then $\mathcal{X} = \{V_f(t) \mid t \in T_e \cup \{u\}\}$ is a cut-system in $I(e)$ satisfying $2\alpha(f) = f(e) + \sum_{t \in T_e} f(e_t) = \gamma(\mathcal{X}) - \kappa(\mathcal{X})$ (hence f is a maximum flow in $I(e)$ by (2)).*

Proof. Since f is a blocking flow in $I(e)$, the family $\{V_f(t) \mid t \in T_e\}$ is a cut-system of T_e blocked by f by definition, and we know that $f(e_t) = f(V_f(t)) = c(V_f(t)) - |\mathrm{odd}(V_f(t))|$ for all terminals $t \in T_e$. First we see that $V_f(u)$ is disjoint with $V_f(t)$ of any terminal $t \in T_e$, since the vertices in $V_f(u)$ are spanned with edges in $E[c - f; 2]$ and the parent-edge of $V_f(t)$ is saturated by f. By Lemma 3(ii), we have $f(V_f(u)) = c(V_f(u)) - |\mathrm{odd}(V_f(u))|$.

We now show that $f(e) = f(V_f(u))$. If $f(e) \in \{c(e), c(e) - 1\}$, then we have $V_f(u) = \{u\}$ and $f(e) = f(V_f(u))$. Consider the case where $c(e) - f(e) \geq 2$. We claim that any positive-path P_{t_1,t_2} for $t_1, t_2 \in T_e$ is disjoint with $V_f(u)$. Assume indirectly that a positive-path P_{t_1,t_2} contains a vertex in $V_f(u)$. Let w be the branch vertex of $P_{t_1,u}$ and $P_{t_2,u}$. The function $f' := f + (P_{t_1,t_2}, -1) + (P_{t_1,u}, 1) + (P_{t_2,u}, 1)$ is a feasible flow in $I(e)$, since $V_f(u)$ is spanned with edges in $E[c - f; 2]$. Since $f'(e') = f(e')$ for all edges $e' \in E - E(P_{u,w})$, the cut-system $\mathcal{X}$ is blocked also by the flow f', and thereby f' is a blocking flow in $I(e)$ with $f'(e) > f(e) = \max\{x \in \Psi(e)\}$, which contradicts the definition of $\Psi(e)$. Hence any positive-path P_{t_1,t_2} with $t_1, t_2 \in T_e$ is disjoint with $V_f(u)$. This proves that $f(e) = f(V_f(u))$ even if $c(e) - f(e) \geq 2$. It always holds that $f(e) = f(V_f(u)) = c(V_f(u)) - |\mathrm{odd}(V_f(u))|$. Therefore we have $2\alpha(f) = f(e) + \sum_{t \in T_e} f(e_t) = \sum_{t \in T_e}(c(V_f(t)) - |\mathrm{odd}(V_f(t))|) + c(V_f(u)) - |\mathrm{odd}(V_f(u))| = \gamma(\mathcal{X}) - \kappa(\mathcal{X})$, as required. ◀

We prove that all edges $e \in E$ satisfies the following conditions (a) and (b) by an induction of depth of edges.

(a) $\Psi(e)$ is given by $\langle a(e), b(e) \rangle$ with some integers $a(e)$ and $b(e)$ such that

(i) For each leaf-edge e, it holds $\Psi(e) = \langle a(e) = c(e), b(e) = c(e) \rangle$;

(ii) For each non-leaf-edge e with two child-edges e_1 and e_2, it holds

$$\Psi(e) = \langle a(e), b(e) \rangle = ((\Psi(e_1) \otimes \Psi(e_2)) \cap [0, c(e)]) \cup \{c(e)\}.$$

That is, for $\langle \tilde{a}(e), \tilde{b}(e) \rangle = \Psi(e_1) \otimes \Psi(e_2)$, where $\tilde{b}(e) = b(e_1) + b(e_2)$ and

$$\tilde{a}(e) = \begin{cases} 0 & \text{if ``}a(e_2) < b(e_1) \text{ or } a(e_1) < b(e_2)\text{'' and } a(e_1) + a(e_2) \text{ is even,} \\ 1 & \text{if ``}a(e_2) < b(e_1) \text{ or } a(e_1) < b(e_2)\text{'' and } a(e_1) + a(e_2) \text{ is odd,} \\ a(e_i) - b(e_j) & \text{if } b(e_j) + 2 \leq a(e_i) \text{ with } \{i, j\} = \{1, 2\}, \end{cases} \tag{3}$$

where edge e_1 (resp., e_2) is called *dominating* if $b(e_2) + 2 \leq a(e_1)$ (resp., $b(e_1) + 2 \leq a(e_2)$), it holds that

$$\langle a(e), b(e) \rangle = \begin{cases} \langle \tilde{a}(e), \tilde{b}(e) \rangle & \text{if } \tilde{b}(e) \leq c(e), \\ \langle \tilde{a}(e), c(e) \rangle & \text{if } \tilde{a}(e) \leq c(e) < \tilde{b}(e) \text{ and } \tilde{a}(e) + c(e) \text{ is even,} \\ \langle \tilde{a}(e), c(e) - 1 \rangle & \text{if } \tilde{a}(e) \leq c(e) < \tilde{b}(e) \text{ and } \tilde{a}(e) + c(e) \text{ is odd,} \\ \langle c(e), c(e) \rangle & \text{if } c(e) < \tilde{a}(e). \end{cases} \tag{4}$$

(b) If $e = (u, v)$ has a dominating child-edge $e' = (v, w)$, then there is a terminal $t \in T_{e'}$ such that $g(u, t) \geq a(e)$ holds for any decomposition g of a blocking flow f to $I(e)$ and $P_{v,t}$ consists of dominating edges.

A path consisting of dominating edges is called a *dominating path.* Fig. 3 shows the pairs $\{\tilde{a}(e), \tilde{b}(e)\}$ and $\{a(e), b(e)\}$ for all edges $e \in E$ in the instance I in Fig. 1 computed according to (3) and (4).

Assuming that each edge with depth at least d satisfies conditions (a) and (b), we prove that any edge e with depth $d - 1$ satisfies the statements in the next lemma, which indicates not only conditions (a) and (b) for the edge e but also how to construct a blocking flow in $I(e)$ from blocking flows in $I(e_1)$ and $I(e_2)$ of the child-edges e_1 and e_2 of e.

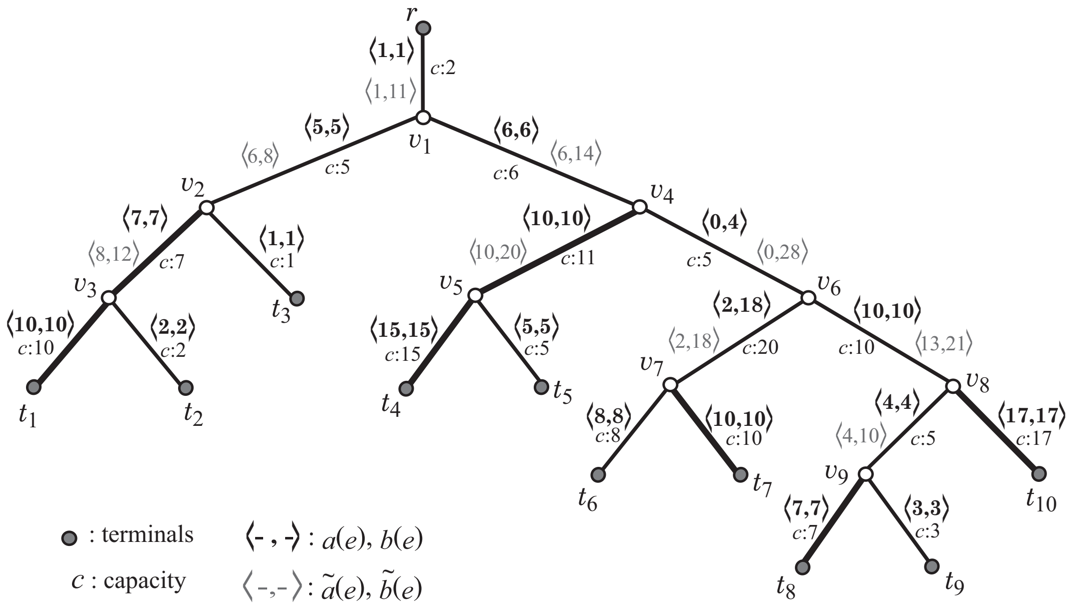


Figure 3 A pair of integers $\tilde{a}(e)$ and $\tilde{b}(e)$ in (3) and $\Psi(e) = \langle a(e), b(e)\rangle$ in (4) for each edge $e \in E$ in the instance I in Fig. 1, where each pair of $a(e)$ and $b(e)$ is depicted in bold while that of $\tilde{a}(e)$ and $\tilde{b}(e)$ in gray. The dominating edges are depicted in thick lines.

5 Main Lemma

This section provides a main technical lemma that tells how to compute the representation of flow values of blocking flows given by conditions (a) and (b), and how to construct a maximum flow from the representations.

▶ **Lemma 5.** *Let $e = (u, v)$ be a non-leaf-edge with depth $d - 1$ (≥ 1). Assume that all edges with depth at least d satisfy conditions* (a) *and* (b)*. For the two children w_1 and w_2 of v, let $\langle \tilde{a}, \tilde{b}\rangle = \Psi(vw_1) \otimes \Psi(vw_2) = \langle a(vw_1), b(vw_1)\rangle \otimes \langle a(vw_2), b(vw_2)\rangle$.*

(i) *For a blocking flow of $I(e)$, if $e \in E(V_f(t))$ for some terminal $t \in T_e$, then the path $P_{v,t}$ from v to t is a dominating path, the path $P_{u,t}$ from u' to t satisfies $g(u, t) \geq c(e)$ for any decomposition g of a blocking flow of $I(e)$, and it holds $c(e) < \tilde{a}$.*

(ii) *One of the child-edges of e is dominating if $c(e) < \tilde{a}$. Edge $e = (u, v)$ satisfies condition* (b)*; if vw_1 or vw_2, say vw_1 is dominating, then there is a terminal $t^* \in T_{vw_1}$ such that $g(u, t^*) \geq \min\{\tilde{a}, c(e)\}$ holds for any decomposition g of a blocking flow of $I(e)$ and P_{v,t^*} is a dominating path.*

(iii) *For any integers x_1, x_2 and x such that $x_i \in \Psi(vw_i)$, $i = 1, 2$ and $x = x_1 + x_2 - 2y$ for some integer $y \in [0, \min\{x_1, x_2\}]$, let f_i, $i = 1, 2$ be a blocking flow of $I(vw_i)$ with $f_i(vw_i) = x_i$. Then $x \geq \tilde{a}$ holds. When $\tilde{a} \leq c(e)$, any function $f = (x, f_1, f_2)$ with $x \leq c(e)$ is a blocking flow of $I(e)$.*

(iv) *If $I(e)$ admits a blocking flow f with $f(e) < c(e)$, then $f(e) \in \langle \tilde{a}, \tilde{b}\rangle$.*

(v) *Assume that $c(e) < \tilde{a}$ and vw_1 is dominating. Let P_{v,t^*} be the dominating path in* (iii) *and let $\delta_e = \tilde{a} - c(e)$. There is a blocking flow f of $I(e)$ with $f(e) = c(e)$, which can be constructed as*

$$f = (c(e), f_1 + (P_{v,t^*}, -\delta_e), f_2)$$

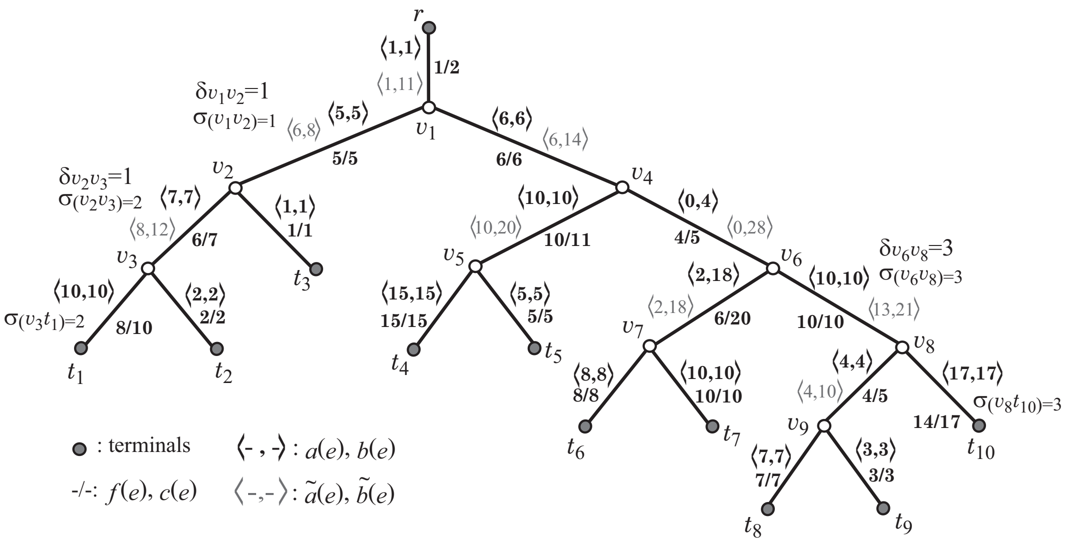


Figure 4 A blocking flow f with $f(e_r) = b(e_r)$ in the instance I in Fig. 1 such that $2\alpha(f) = \sum_{t\in T} f(e_t) = 1+8+2+1+15+5+8+10+7+3+14 = 74$, where the pair of flow value $f(e)$ and capacity $c(e)$ for each edge is indicated by f/c beside the line segment for edge e. The non-zero values for δ_e and $\sigma(e)$ are indicated beside the corresponding edge e.

by choosing a blocking flow f_1 of $I(vw_1)$ with $f_1(vw_1) = a(vw_1)$ and a blocking flow f_2 of $I(vw_2)$ with $f_2(vw_2) = b(vw_2)$.

(vi) *Edge $e = (u, v)$ satisfies condition* (a)*; i.e., $\Psi(e) = (\langle\tilde{a}, \tilde{b}\rangle \cap [0, c(e)]) \cup \{c(e)\}$.*

A proof of this lemma can be found in the full version of this paper.

6 Algorithm Description

Based on Lemma 5, this section gives a description of a linear-time algorithm for computing the representations of flow values of blocking flows and constructing a maximum flow from the representations.

By Lemma 5(ii) and (iv), we see by induction that every edge in E satisfies conditions (a) and (b). By Lemma 5(iii) and (v), we know how to construct a blocking flow in $I(e)$ for some edge e from blocking flows in $I(e_1)$ and $I(e_2)$ of the child-edges e_1 and e_2 of e. By Lemma 4, it suffices to construct a blocking flow in $I = I(e_r)$ with $f(e_r) = b(e_r)$. For this, we first compute the integers $\tilde{a}(e)$, $\tilde{b}(e)$, $a(e)$ and $b(e)$ for each edge $e \in E$ according to (3) and (4) selecting edges in E in a non-increasing order of depth, and identify all the dominating edges in E. Next we apply Lemma 5(iii) and (v) repeatedly from edge e_r to descendants of the edge in a top-down manner to construct a blocking flow in $I = I(e_r)$ with $f(e_r) = b(e_r)$. To implement the algorithm to run in linear time, we avoid reducing flow values repeatedly along part of a dominating path. We let $\sigma(e)$ to store the total amount of decrements over each dominating edge e, i.e., $\sigma(e)$ is the summation of $\delta_{e'}$ in Lemma 5(v) over all dominating edges e' that are ancestors of e. An entire algorithm is given by the following compact and succinct description.

The algorithm runs in linear time, because it executes an $O(1)$-time procedure to each edge in E in constant time. Fig. 4 illustrates a result obtained from the instance I in Fig. 1 by applying the algorithm.

Algorithm 1 BlockFlow

```
Input: An instance $I = (G = (V, E), T, c)$ rooted at a terminal $r \in T$.
Output: A maximum flow $f$ in $I$.
  Compute the integers $\tilde{a}(e)$, $\tilde{b}(e)$, $a(e)$ and $b(e)$ for each edge $e \in E$ according to (3) and
  (4) selecting edges in $E$ in a non-increasing order of depth;
  $x(e_r) := b(e_r)$; $\sigma(e_r) := 0$;
  for each edge $e \in E$ selected in a non-decreasing order of depth do
    $f(e) := x(e) - \sigma(e)$;
    if $e$ is not a leaf edge then
      /* Denote by $e_1$ and $e_2$ the child-edges of $e$ */
      if $\tilde{a}(e) \leq c(e)$ then
        Choose integers $x_1 \in \langle a(e_1), b(e_1)\rangle$ and $x_2 \in \langle a(e_2), b(e_2)\rangle$ such that
        $x(e) = x_1 + x_2 - 2y$ for some integer and $y \in [0, \min\{x_1, x_2\}]$;
        $x(e_1) = x_1$;  $x(e_2) = x_2$;
        if $e_i$ is dominating for $i = 1$ or 2 then
          $\sigma(e_i) := \sigma(e)$ and $\sigma(e_j) := 0$ for $j \in \{1, 2\} - \{i\}$
        else
          $\sigma(e_1) := \sigma(e_2) := 0$
        end if
      else
        /* $c(e_0) < \tilde{a}(e_0)$, where $e_0$ is dominating, and exactly one of $e_1$ and $e_2$ is
        dominating; assume that $e_1$ is dominating without loss of generality. */
        $x(e_1) = a(e_1)$;  $x(e_2) = b(e_2)$;  $\delta_{e_1} := a(e_1) - c(e)$;
        $\sigma(e_1) := \sigma(e) + \delta_{e_1}$;  $\sigma(e_2) := 0$
      end if
    end if
  end for
```

After a maximum flow f is constructed, a minimum cut-system $\mathcal{X}$ to a given instance can be constructed in linear time by Lemma 4. Fig. 5 illustrates the cut-system $\mathcal{X} = \{V_f(t) \mid t \in T\}$ for the blocking flow f in Fig. 4, which indicates that the flow f is maximum because $2\alpha(f) = \sum_{t \in T} f(e_t) = 74 = \gamma(\mathcal{X}) - \kappa(\mathcal{X})$ holds.

From the above argument, the next theorem is established.

▶ **Theorem 6.** *Given a tree instance (G, T, c), a feasible integral multiflow f and a cut-system $\mathcal{X}$ with $\alpha(f) = (\gamma(\mathcal{X}) - \kappa(\mathcal{X}))/2$ can be found in $O(n)$ time and space, where f is a maximum integral multiflow.*

7 Concluding Remarks

In this paper, we revealed a recursive formula among flow values of blocking flows in rooted instances and designed a linear-time dynamic programming algorithm for computing a maximum integral flow in a tree instance. The optimality of flows is ensured by the property of the formula, by which we can always construct the corresponding dual object, i.e., a minimum cut-system that satisfies (2) by equality.

It would be interesting to characterize similar recursive properties and design fast algorithms for the maximum integral multiterminal flows in more general classes of graphs.

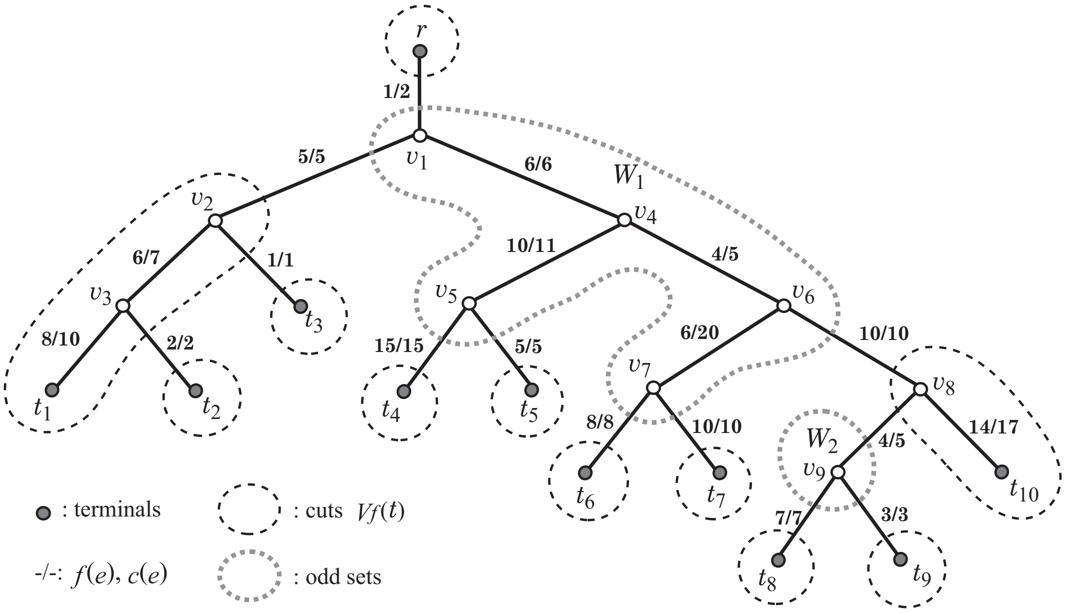


Figure 5 The cut-system $\mathcal{X} = \{V_f(t) \mid t \in T\}$ for the blocking flow f in Fig. 4, where the set $V - \cup_{X \in \mathcal{X}} X$ induces from G two odd sets $W_1 \in \text{odd}(V_f(r))$ and $W_2 \in \text{odd}(V_f(t_{10}))$, and it holds that $\gamma(\mathcal{X}) - \kappa(\mathcal{X}) = \sum_{t \in T} c(V_f(t)) - 2 = 2 + (2+1+5) + 2 + 1 + 15 + 5 + 8 + 10 + 7 + 3 + (5+10) - 2 = 74$.

References

1 B. V. Cherkasskii. Reshenie odnoi zadachi o mnogoproduktovykh potokakh v seti [russian; a solution of a problem of multicommodity flows in a network]. *Èkonomika i Matematicheskie Metody*, 13(1):143–151, 1977.

2 Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Minimal multicut and maximal integer multiflow: a survey. *European Journal of Operational Research*, 162(1):55–69, 2005.

3 Mariechristine Costa and Alain Billionnet. Multiway cut and integer flow problems in trees. *Electronic Notes in Discrete Mathematics*, 17(20):105–109, 2004.

4 William H. Cunningham. The optimal multiterminal cut problem. *DIMACS series in discrete mathematics and theoretical computer science*, 5:105–120, 1991.

5 J. R. Ford and D. R. Fulkerson. *Flows in networks.* Princeton university press, 1962.

6 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

7 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

8 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

9 Toshihide Ibaraki, Alexander V. Karzanov, and Hiroshi Nagamochi. A fast algorithm for finding a maximum free multiflow in an inner eulerian network and some generalizations. *Combinatorica*, 18(1):61–83, 1998.

10 Wolfgang Mader. Über die Maximalzahl kantendisjunkter A-Wege. *Archiv der Mathematik*, 30(1):325–336, 1978.

Shortest Disjoint $\mathcal{S}$-Paths Via Weighted Linear Matroid Parity

Yutaro Yamaguchi

Osaka University, Osaka 565-0871, Japan
yutaro_yamaguchi@ist.osaka-u.ac.jp

Abstract

Mader's disjoint $\mathcal{S}$-paths problem unifies two generalizations of bipartite matching: (a) non-bipartite matching and (b) disjoint s–t paths. Lovász (1980, 1981) first proposed an efficient algorithm for this problem via a reduction to matroid matching, which also unifies two generalizations of bipartite matching: (a) non-bipartite matching and (c) matroid intersection. While the weighted versions of the problems (a)–(c) in which we aim to minimize the total weight of a designated-size feasible solution are known to be solvable in polynomial time, the tractability of such a weighted version of Mader's problem has been open for a long while.

In this paper, we present the first solution to this problem with the aid of a linear representation for Lovász' reduction (which leads to a reduction to linear matroid parity) due to Schrijver (2003) and polynomial-time algorithms for a weighted version of linear matroid parity announced by Iwata (2013) and by Pap (2013). Specifically, we give a reduction of the weighted version of Mader's problem to weighted linear matroid parity, which leads to an $\mathrm{O}(n^5)$-time algorithm for the former problem, where n denotes the number of vertices in the input graph. Our reduction technique is also applicable to a further generalized framework, packing non-zero A-paths in group-labeled graphs, introduced by Chudnovsky, Geelen, Gerards, Goddyn, Lohman, and Seymour (2006). The extension leads to the tractability of a broader class of weighted problems not restricted to Mader's setting.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Approximation, G.2.2 Graph Theory

Keywords and phrases Mader's $\mathcal{S}$-paths, packing non-zero A-paths in group-labeled graphs, linear matroid parity, weighted problems, tractability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.63

1 Introduction

1.1 Mader's $\mathcal{S}$-paths

Let $G = (V, E)$ be an undirected graph. For a prescribed vertex set $A \subseteq V$ with its partition $\mathcal{S}$ (i.e., $\mathcal{S}$ is a family of disjoint nonempty subsets of A whose union is A), an *A-path* is a path between distinct vertices in A that does not intersect A in between, and an *$\mathcal{S}$-path* is an A-path whose end vertices belong to distinct elements of $\mathcal{S}$. Each vertex in A is called a *terminal*.

Mader's disjoint $\mathcal{S}$-paths problem is to find a maximum number of vertex-disjoint $\mathcal{S}$-paths in a given undirected graph with a terminal set partitioned as $\mathcal{S}$. This problem can formulate (a) the *non-bipartite matching problem* (finding a maximum matching in a given undirected graph) and (b) the *disjoint s–t paths problem* (finding a maximum number of openly disjoint paths between two specified vertices s and t in a given undirected graph), which are both

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 63; pp. 63:1–63:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

fundamental problems generalizing the *bipartite matching problem* (finding a maximum matching in a given bipartite graph).

Mader's problem was first mentioned by Gallai [8], and Mader [19] gave a good characterization by a min-max duality theorem. Lovász [16, 17] proposed the first polynomial-time algorithm via a reduction to the matroid matching problem, and later Schrijver [23, Section 73.1a] pointed out that Lovász' reduction enjoys a linear representation, which leads to a reduction of Mader's problem to the linear matroid parity problem (see Section 1.2 for the definitions). Chudnovsky, Geelen, Gerards, Goddyn, Lohman, and Seymour [5] recently introduced a further generalized framework, called packing non-zero A-paths in group-labeled graphs (see Section 3.1 for the definition). They provided a min-max duality theorem extending Mader's theorem, and Chudnovsky, Cunningham, and Geelen [4] developed a direct combinatorial polynomial-time algorithm for that generalized problem.

In this paper, we focus on the following weighted version of Mader's problem, whose tractability has been open for a long while: for a given nonnegative length of each edge, to minimize the total length of a designated number of disjoint $\mathcal{S}$-paths. For a family $\mathcal{P}$ of disjoint paths, we denote by $E(\mathcal{P})$ the set of edges traversed by some path in $\mathcal{P}$.

Shortest Disjoint $\mathcal{S}$-paths Problem

Input: An undirected graph $G = (V, E)$, a terminal set $A \subseteq V$ with its partition $\mathcal{S}$, a nonnegative edge length $\ell \in \mathbb{R}^E_{\geq 0}$, and a positive integer $k \in \mathbb{Z}_{>0}$.

Goal: Find a family $\mathcal{P}$ of k vertex-disjoint $\mathcal{S}$-paths in G with $\ell(\mathcal{P}) := \sum_{e \in E(\mathcal{P})} \ell_e$ minimum.

Related work

Karzanov [13] presented a polynomial-time algorithm for a similar weighted problem in the edge-disjoint A-paths setting (which is a special case of Mader's setting), whose full proof was left to an unpublished paper [12]. Karzanov's problem can be solved by finding shortest k vertex-disjoint $\mathcal{S}$-paths for all possible k, where the number of iterations is at most $|A|/2$ and can be reduced to $\mathrm{O}(\log |A|)$ by binary search. Hirai and Pap [10] discussed the tractability of a generalization of Karzanov's setting, in which each pair of two terminals has weight. Pap [21] dealt with another weighted version of Mader's problem, in which weight is defined only on terminal pairs (no edge length or cost).

Björklund and Husfeldt [1] recently devised a randomized algorithm for the *shortest* 2*-disjoint paths problem* (minimizing the total length of two vertex-disjoint paths between two pairs (s_1, t_1) and (s_2, t_2) of specified vertices in a given undirected graph), which is the first polynomial-time one. Inspired by their work, Kobayashi and Toyooka [14] developed one for finding a shortest non-zero s–t path in a group-labeled graph, and so did Hirai and Namba [9] for finding shortest disjoint $(A+B)$-paths in an undirected graph that cover all the vertices in $A \cup B$ when $|A| + |B|$ is fixed, where an $(A+B)$*-path* is an A-path or a B-path for disjoint vertex sets A and B. The weighted version of packing non-zero A-paths includes finding a shortest non-zero s–t path (see Section 3.5).

1.2 Matroid Matching

The matroid matching problem introduced by Lawler [15] also unifies two fundamental generalizations of bipartite matching: (a) the non-bipartite matching problem and (c) the *matroid intersection problem* (finding a maximum-cardinality common independent set in given two matroids on the same ground set). This problem cannot be solved in polynomial time in general, but is known to be tractable as well as to admit a good characterization when the matroid in question is linearly represented (or in a more general situation) due to Lovász [16, 17] and Dress and Lovász [6].

We here describe the problem formulation. A 2-*polymatroid* is a pair (S, f) of a finite set S (called the *ground set*) and an integer-valued set function $f\colon 2^S \to \mathbb{Z}$ such that

- $0 \leq f(X) \leq 2|X|$ for each $X \subseteq S$,
- $f(X) \leq f(Y)$ for each $X \subseteq Y \subseteq S$, and
- $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ for each $X, Y \subseteq S$.

A subset $X \subseteq S$ is called a *matching* in a 2-polymatroid (S, f) if $f(X) = 2|X|$, and a *parity base* if $f(X) = 2|X| = f(S)$.

The *matroid matching problem* is to find a maximum matching in a given 2-polymatroid. In this paper, we utilize the following weighted version of this problem: for a given weight on the ground set, to minimize the total weight of a parity base.

Weighted Matroid Matching Problem

Input: A 2-polymatroid (S, f) and a weight $w \in \mathbb{R}^S$.

Goal: Find a parity base $B \subseteq S$ in (S, f) with $w(B) := \sum_{e \in B} w_e$ minimum.

A 2-polymatroid (S, f) is said to be *linearly represented* over a field $\mathbb{F}$ if we are given a matrix $Z = (Z_e)_{e \in S} \in \mathbb{F}^{r \times 2S}$ obtained by concatenating $r \times 2$ matrices $Z_e \in \mathbb{F}^{r \times 2}$ $(e \in S)$ such that $f(X) = \operatorname{rank} Z(X)$ for every $X \subseteq S$, where r is a positive integer and $Z(X) = (Z_e)_{e \in X}$ denotes the submatrix of Z obtained by extracting the corresponding columns. A subset $X \subseteq S$ is called a *matching* or *parity base* for Z if it is a matching or parity base, respectively, in the corresponding 2-polymatroid.

When the input 2-polymatroid is linearly represented, the matroid matching problem is called the *linear matroid parity problem*, for which various efficient algorithms have been developed originated by Lovász [17], e.g., by Gabow and Stallmann [7], Orlin [20], and Cheung, Lau, and Leung [3]. The same weighted version is formulated as follows.

Weighted Linear Matroid Parity Problem

Input: A finite set S, a matrix $Z \in \mathbb{F}^{r \times 2S}$ over a field $\mathbb{F}$, and a weight $w \in \mathbb{R}^S$.

Goal: Find a parity base $B \subseteq S$ for Z with $w(B)$ minimum.

Camerini, Galbiati, and Maffioli [2] first showed that the weighted linear matroid parity problem can be solved in pseudopolynomial time, and later Cheung et al. [3] devised a faster pseudopolynomial-time algorithm. As announced by Iwata [11], this problem is solved in $\mathrm{O}(r^3|S|)$ time, which is strongly polynomial. Independently, Pap [22] also announced a strongly polynomial-time algorithm for an equivalent weighted problem, for which no estimate of the running time was given.

1.3 Results

We present a reduction of the shortest disjoint $\mathcal{S}$-paths problem to the weighted linear matroid parity problem.

▶ **Theorem 1.** *The shortest disjoint $\mathcal{S}$-paths problem reduces to the weighted linear matroid parity problem.*

Our reduction leads to the first polynomial-time algorithm for the shortest disjoint $\mathcal{S}$-paths problem with the aid of weighted linear matroid parity algorithms due to Iwata [11] and Pap [22]. This is also the first successful application of weighted linear matroid parity.

As seen in Section 2, when $|V| = n$, our reduction results in an $\mathrm{O}(n) \times \mathrm{O}(n^2)$ matrix and requires $\mathrm{O}(n^3)$ time in addition to solving weighted linear matroid parity, and hence we obtain the following running time bound by using Iwata's algorithm.

▶ **Corollary 2.** *One can solve the shortest disjoint $\mathcal{S}$-paths problem in $\mathrm{O}(|V|^5)$ time.*

Moreover, our reduction technique can be extended to the same weighted version of packing non-zero A-paths. With the aid of reductions of packing non-zero A-paths to matroid matching and to linear matroid parity due to Tanigawa and the author [24] and the author [25], respectively, we obtain reductions between the weighted versions. While we can see the tractability via a reasonable reduction to weighted linear matroid parity only when the group in question satisfies some representability condition (see Theorem 11), it holds for a variety of groups including one for formulating Mader's setting. We refer the readers to Section 3 for the details.

The rest of this paper is organized as follows. Section 2 is devoted to presenting our reduction of the shortest disjoint $\mathcal{S}$-paths problem to the weighted linear matroid parity problem. In Section 3, we show that our reduction technique can be utilized to reduce the same weighted version of packing non-zero A-paths to weighted matroid matching and to weighted linear matroid parity. Finally, in Section 4, we conclude this paper with some open problems related to our work.

2 Reduction

In this section, we give a reduction of the shortest disjoint $\mathcal{S}$-paths problem to the weighted linear matroid parity problem. We first review Schrijver's linear representation for Lovász' reduction in Section 2.1, and then present our reduction. For the sake of convenience, we restate the two problems.

Shortest Disjoint $\mathcal{S}$-paths Problem

Input: An undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, a terminal set $A \subseteq V$ with its partition $\mathcal{S}$, a nonnegative edge length $\ell \in \mathbb{R}^E_{\geq 0}$, and a positive integer $k \in \mathbb{Z}_{>0}$.

Goal: Find a family $\mathcal{P}$ of k vertex-disjoint $\mathcal{S}$-paths in G with $\ell(\mathcal{P})$ minimum.

Weighted Linear Matroid Parity Problem

Input: A finite set S, a matrix $Z \in \mathbb{F}^{r \times 2S}$ over a field $\mathbb{F}$, and a weight $w \in \mathbb{R}^S$.

Goal: Find a parity base $B \subseteq S$ for Z with $w(B)$ minimum.

Our reduction procedure is summarized as follows. We first construct an auxiliary graph G' from a given undirected graph G (see Section 2.2), which is the most important contribution of this paper. This step requires $\mathrm{O}(n^2)$ time. Next, following Schrijver's linear representation, we make a matrix Z over the field $\mathbb{Q}$ of rationals associated with the auxiliary graph G', and define a weight w from the edge length ℓ in a natural way (see Section 2.3). This step takes $\mathrm{O}(n^3)$ time. Finally, we show that the following two facts (see Claim 5), which complete the reduction within $\mathrm{O}(n^3)$ time in total:

- for any family $\mathcal{P}$ of k vertex-disjoint $\mathcal{S}$-paths in G, there exists a parity base B for Z with $w(B) = \ell(\mathcal{P})$;
- for any parity base B for Z, there exists a family $\mathcal{P}$ of k vertex-disjoint $\mathcal{S}$-paths in G with $\ell(\mathcal{P}) \leq w(B)$, which can be found easily, in $\mathrm{O}(n)$ time.

2.1 Associated Matrix for Mader's $\mathcal{S}$-paths

In this section, we review Schrijver's linear representation [23, Section 73.1a] for Lovász' reduction of Mader's problem to matroid matching. For a given undirected graph $G = (V, E)$ with a terminal set $A \subseteq V$ partitioned as $\mathcal{S} = \{A_1, A_2, \ldots, A_t\}$, we construct an associated matrix $Z \in \mathbb{Q}^{2V \times 2E}$, where a 2×2 submatrix corresponds to each vertex $v \in V$ and each edge $e \in E$. We assume that every connected component of G has at least one $\mathcal{S}$-path.

Associate each edge $e = uw \in E$ with a 2-dimensional linear subspace of $(\mathbb{Q}^2)^V$,

$$L_e := \{\, x \in (\mathbb{Q}^2)^V \mid x(u) + x(w) = \mathbf{0},\ x(v) = \mathbf{0} \ \ (v \in V \setminus \{u, w\}) \,\}. \tag{1}$$

For each terminal $a \in A_i$ $(i = 1, 2, \ldots, t)$, define a 1-dimensional linear subspace

$$Q_a := \{\, x \in (\mathbb{Q}^2)^V \mid x(a) \in \left\langle \binom{1}{i} \right\rangle,\ x(v) = \mathbf{0} \ \ (v \in V \setminus \{a\}) \,\}, \tag{2}$$

where $\langle y \rangle := \{\, py \mid p \in \mathbb{F} \,\}$ for a vector $y \in \mathbb{F}^r$ over a field $\mathbb{F}$.

Let $Q := \sum_{a \in A} Q_a$ and $\mathcal{E} := \{\, L_e/Q \mid e \in E \,\}$. Let us construct a matrix $Z \in \mathbb{Q}^{2V \times 2E}$ associated with $\mathcal{E}$ so that $\operatorname{rank} Z(F) = \dim(L_F/Q)$ for every $F \subseteq E$, where $L_F := \sum_{e \in F} L_e$. This can be done by arranging an appropriate basis of $L_e/Q \in \mathcal{E}$ (which is regarded as taken from the original space $(\mathbb{Q}^2)^V$) for each edge $e \in E$ (see also Remark at the end of this section). Then, the matchings for Z are characterized as follows, from which we can derive a useful characterization of the parity bases for Z (Lemma 4).

▶ **Lemma 3** (Schrijver [23, (73.18)]). *An edge set $F \subseteq E$ is a matching for Z if and only if F forms a forest such that every connected component has at most two terminals in A, which belong to distinct elements of $\mathcal{S}$ if there are two.*

▶ **Lemma 4.** *An edge set $F \subseteq E$ is a parity base for Z if and only if F forms a spanning forest of G such that every connected component has exactly two terminals in A, which belong to distinct elements of $\mathcal{S}$.*

Proof. We first see that $\operatorname{rank} Z = 2|V| - |A|$. By the above construction, it suffices to show the equality for each connected component, and assume that G itself is connected. Fix an $\mathcal{S}$-path in G (recall the existence assumption), and let $a, b \in A$ be its end vertices. Then, by Lemma 3, $\dim(L_E/(Q_a + Q_b)) \geq 2(|V| - 1)$ (consider a spanning tree of G with a restricted terminal set $\{a, b\}$ instead of A). Since $L_E/(Q_a + Q_b)$ is a linear subspace of the $(2|V| - 2)$-dimensional quotient space $(\mathbb{Q}^2)^V/(Q_a + Q_b)$, these two spaces are identified. Hence, $\operatorname{rank} Z = \dim(L_E/Q) = \dim((\mathbb{Q}^2)^V/Q) = \dim(\mathbb{Q}^2)^V - \dim Q = 2|V| - |A|$.

["If" part] Suppose that $F \subseteq E$ forms such a spanning forest of G. Then it immediately follows from Lemma 3 that F is a matching for Z. Since every connected component has exactly two terminals in A, the spanning forest formed by F consists of exactly $|A|/2$ connected components, which implies $|F| = |V| - \frac{1}{2}|A| = \frac{1}{2}\operatorname{rank} Z$.

["Only if" part] Suppose that $F \subseteq E$ is a parity base for Z. We then have $|F| = \frac{1}{2}\operatorname{rank} Z = |V| - \frac{1}{2}|A|$. Since F forms a forest by Lemma 3, the subgraph (V, F) is also a forest, which consists of $|A|/2$ connected components including isolated vertices if exist. Then, by Lemma 3 and the pigeonhole principle, every connected component of (V, F) has exactly two terminals in A (which belong to distinct elements of $\mathcal{S}$), and hence F must be spanning. ◀

▶ Remark. The above construction does not define a unique associated matrix $Z \in \mathbb{Q}^{2V \times 2E}$, and one is obtained as follows. We first compute the Kronecker product $B_G \otimes I_2 \in \mathbb{Q}^{2V \times 2E}$ of the incidence matrix $B_G \in \{-1, 0, 1\}^{V \times E} \subseteq \mathbb{Q}^{V \times E}$ of G (where each edge in G is assumed to be arbitrarily oriented) and the 2×2 identity matrix $I_2 \in \mathbb{Q}^{2\times 2}$. Note that $B_G \otimes I_2$ is a matrix obtained by arranging a basis of L_e for each edge $e \in E$. We then obtain Z by adding to each column of $B_G \otimes I_2$ a multiple of a vector $x \in (\mathbb{Q}^2)^V$ with $\langle x \rangle = Q_a$ for each terminal $a \in A$ (e.g., x is defined by $x(a) := \binom{1}{i}$ and $x(v) := \mathbf{0}$ ($v \in V \setminus \{a\}$) when $a \in A_i$) so that the first row of the corresponding submatrix $Z_a \in \mathbb{Q}^{2\times 2E}$ has only zero. This procedure takes $\mathrm{O}(|V| \cdot |E|)$ time in total.

2.2 Construction of Auxiliary Graph

As the first step of our reduction, we construct an auxiliary undirected graph $G' = (V', E')$ with a terminal set $A' \subseteq V'$ partitioned as $\mathcal{S}'$ from a given undirected graph $G = (V, E)$ with a terminal set $A \subseteq V$ partitioned as $\mathcal{S}$. We assume that there exists a feasible solution, i.e., G has k vertex-disjoint $\mathcal{S}$-paths, and then we have $|A| \geq 2k$.

The construction is summarized as follows (see also Fig. 1). Add $(|A| - 2k)$ extra terminals so that each extra terminal is adjacent to all the original terminals in A, and let A_0 be the set of those extra terminals. Besides, add two other extra terminals b_1, b_2 so that b_1 and b_2 are adjacent and b_1 is adjacent to all the non-terminal vertices in $V \setminus A$. Finally, define $\mathcal{S}' := \mathcal{S} \cup \{A_0, \{b_1\}, \{b_2\}\}$.

Formally, for the vertex set, let a'_i ($i = 1, 2, \ldots, |A| - 2k$) and b_j ($j = 1, 2$) be distinct new vertices not in V, and define

$$\begin{aligned}
A_0 &:= \{\, a'_i \mid i = 1, 2, \ldots, |A| - 2k \,\},\\
V' &:= V \cup A_0 \cup \{b_1, b_2\},\\
A' &:= A \cup A_0 \cup \{b_1, b_2\},\\
\mathcal{S}' &:= \mathcal{S} \cup \{A_0, \{b_1\}, \{b_2\}\}.
\end{aligned}$$

For the edge set, define

$$\begin{aligned}
E_1 &:= \{\, e_{ia} = a'_i a \mid a'_i \in A_0,\ a \in A \,\},\\
E_2 &:= \{\, e_v = b_1 v \mid v \in V \setminus A \,\},\\
E' &:= E \cup E_1 \cup E_2 \cup \{e' = b_1 b_2\}.
\end{aligned}$$

Note that, since $|A_0| \leq |A| = \mathrm{O}(n)$ and we may assume that G has no parallel edges, we have $|V'| = \mathrm{O}(n)$ and $|E'| = \mathrm{O}(m + n^2) = \mathrm{O}(n^2)$.

2.3 Completion of Reduction

For the auxiliary graph $G' = (V', E')$ with a terminal set $A' \subseteq V'$ partitioned as $\mathcal{S}'$ obtained in Section 2.2, we construct an associated matrix $Z \in \mathbb{Q}^{2V' \times 2E'}$ defined in Section 2.1. Note that the construction requires $\mathrm{O}(n^3)$ time, because $|V'| = \mathrm{O}(n)$ and $|E'| = \mathrm{O}(n^2)$. Define a weight $w \in \mathbb{R}^{E'}$ as follows: for each $e \in E'$,

$$w_e := \begin{cases} \ell_e & (e \in E),\\ 0 & (e \in E' \setminus E). \end{cases} \tag{3}$$

Note that $w(F) = w(F \cap E) = \sum_{e \in F \cap E} \ell_e$ for every $F \subseteq E'$, and $w(F_1) \leq w(F_2)$ for every $F_1 \subseteq F_2 \subseteq E'$ (recall that ℓ is nonnegative).

Our reduction is completed by the following claim, which implies that one can efficiently transform any minimum-weight parity base for Z into an optimal solution to the shortest disjoint $\mathcal{S}$-paths problem.

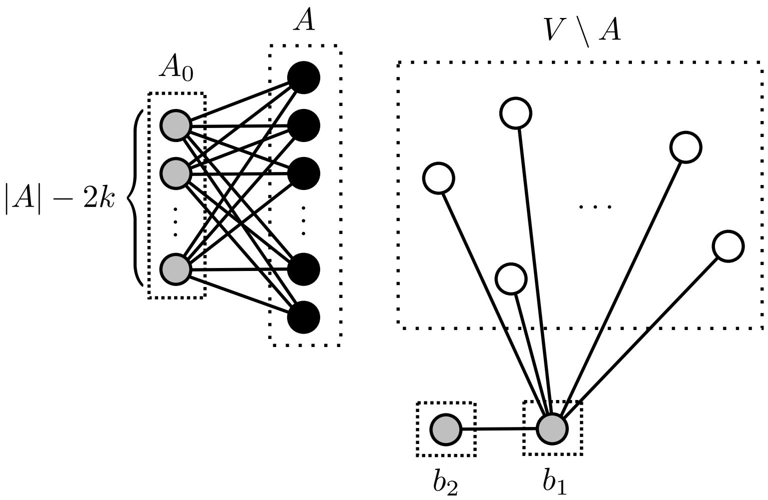


Figure 1 How to construct the auxiliary graph (the original edges are omitted).

▶ **Claim 5.** *The following relations hold between feasible solutions of the two problems.*

(i) *For any family $\mathcal{P}$ of k vertex-disjoint $\mathcal{S}$-paths in G, there exists a parity base $B_{\mathcal{P}} \subseteq E'$ for Z with $B_{\mathcal{P}} \cap E = E(\mathcal{P})$ (hence $w(B_{\mathcal{P}}) = \ell(\mathcal{P})$).*

(ii) *For any parity base $B \subseteq E'$ for Z, there exists a family $\mathcal{P}_B$ of k vertex-disjoint $\mathcal{S}$-paths in G with $E(\mathcal{P}_B) \subseteq B \cap E$ (hence $\ell(\mathcal{P}_B) \leq w(B)$), which can be found in $\mathrm{O}(n)$ time.*

Proof. (i) Let $\mathcal{P}$ be a family of k vertex-disjoint $\mathcal{S}$-paths in G, which are also $\mathcal{S}'$-paths in G'. Then, exactly $(|A| - 2k)$ terminals in A are exposed by $\mathcal{P}$. By the definition of $A_0 \subseteq A' \setminus A$ and $E_1 \subseteq E' \setminus E$, there exist a perfect matching in G' between those exposed terminals and A_0, which form $(|A| - 2k)$ vertex-disjoint $\mathcal{S}'$-paths in G'. Besides, G' has one more $\mathcal{S}'$-path consisting of the single edge $e' = b_1 b_2 \in E' \setminus E$. We can obtain a family $\mathcal{P}'$ of $(|A| - k + 1)$ vertex-disjoint $\mathcal{S}'$-paths in G' by adding these paths to $\mathcal{P}$.

Let $U \subseteq V \setminus A$ be the set of non-terminal vertices that are not intersected by any path in $\mathcal{P}'$. By the definition of $E_2 \subseteq E' \setminus E$, there exists an edge $e_v = b_1 v \in E_2$ for every $v \in U$. Let $B_{\mathcal{P}} := E(\mathcal{P}') \cup \{\, e_v \mid v \in U \,\}$. Then, obviously $B_{\mathcal{P}} \cap E = E(\mathcal{P})$, and $B_{\mathcal{P}}$ is indeed a parity base for Z by Lemma 4.

(ii) Let $B \subseteq E'$ be a parity base for Z. Then, by Lemma 4, B forms a spanning forest of G such that every connected component has exactly two terminals in A', which belong to distinct elements of $\mathcal{S}'$.

By the construction of G', the terminal b_2 can be connected only to the terminal b_1, and hence $e' \in B$. In addition, every terminal in A_0 can be connected only to the original terminals in A, and hence every connected component containing a terminal in A_0 has an original terminal in A. This implies that $2k$ original terminals in A are distributed to k connected components in the spanning forest so that every connected component has two terminals belonging to distinct elements of $\mathcal{S}$. Since every additional edge in $E' \setminus E$ incident to at least one extra terminal in $A' \setminus A$, the restriction $B \cap E$ forms a forest in G that contains k vertex-disjoint $\mathcal{S}$-paths. Those $\mathcal{S}$-paths can be extracted in $\mathrm{O}(n)$ time by the breadth first search in the forest from each terminal in A. ◀

3 Extension to Non-zero A-paths

In this section, we discuss a possible extension of our reduction technique to a further generalized framework, packing non-zero A-paths in group-labeled graphs. We first describe necessary definitions and the background in Section 3.1, and next show how to adjust our reduction to this framework in Section 3.2. In Sections 3.3 and 3.4, we discuss reductions to weighted matroid matching in general and to weighted linear matroid parity under some representability condition for the group in question, respectively. Finally, in Section 3.5, we briefly describe several special cases solvable via weighted linear matroid parity.

3.1 Packing Non-zero A-paths

Let Γ be a group. A Γ-*labeled graph* is a directed graph $G = (V, E)$ with each edge labeled by an element of Γ, i.e., with a mapping $\psi_G \colon E \to \Gamma$ called a *label function*. The *label* of a walk $W = (v_0, e_1, v_1, e_2, v_2, \dots, e_l, v_l)$ in the underlying graph of G (i.e., $v_i \in V$ for each $i = 0, 1, \dots, l$ and either $e_i = v_{i-1}v_i \in E$ or $e_i = v_iv_{i-1} \in E$ for each $i = 1, 2, \dots, l$) is defined as $\psi_G(W) := \psi_G(e_l) \cdots \psi_G(e_2) \cdot \psi_G(e_1)$ if $e_i = v_{i-1}v_i$ for every $i = 1, 2, \dots, l$, and otherwise by replacing the corresponding label $\psi_G(e_i)$ with its inverse $\psi_G(e_i)^{-1}$ for each i with $e_i = v_iv_{i-1}$. A walk is called *balanced* (or a *zero* walk) if its label is the identity element 1_Γ of Γ, and *unbalanced* (or a *non-zero* walk) otherwise.

The problem of *packing non-zero A-paths*, introduced by Chudnovsky et al. [5], is to find a maximum number of vertex-disjoint non-zero A-paths in a given Γ-labeled graph. It generalizes Mader's disjoint $\mathcal{S}$-paths problem as well as several interesting problems such as ones in topological graph theory (see [5, Section 2] for more details). For this problem, a min-max duality theorem and a polynomial-time algorithm were given by Chudnovsky et al. [5] and Chudnovsky et al. [4], respectively. Extending Lovász' reduction [16] of Mader's problem to tractable matroid matching, Tanigawa and the author [24] gave a reduction of this problem to tractable matroid matching. Furthermore, the author [25] clarified when Schrijver's reduction [23, Section 73.1a] (cf. Section 2.1) of Mader's problem to linear matroid parity can be extended to this problem.

The weighted version of packing non-zero A-paths is similarly formulated as follows.

Shortest Disjoint Non-zero A-paths Problem

Input: A Γ-labeled graph $G = (V, E)$, a terminal set $A \subseteq V$, a nonnegative edge length $\ell \in \mathbb{R}^E_{\geq 0}$, and a positive integer $k \in \mathbb{Z}_{>0}$.

Goal: Find a family $\mathcal{P}$ of k vertex-disjoint non-zero A-paths in G with $\ell(\mathcal{P})$ minimum.

3.2 Adjustment for Non-zero A-paths

Similarly to Section 2.2, we construct an auxiliary Γ-labeled graph $G' = (V', E')$ with a terminal set $A' \subseteq V'$ from a given Γ-labeled graph $G = (V, E)$ with a terminal set $A \subseteq V$. The underlying graph and terminal set are defined in the same way, and only the difference is that we have to define a label for each additional edge in $E' \setminus E$ instead of defining a partition $\mathcal{S}'$ of the terminal set A'.

It in fact works well to define the labels of additional edges in $E' \setminus E$ so that each $\mathcal{S}'$-path consisting of a single additional edge in the auxiliary graph constructed in Section 2.2 is replaced by a non-zero A'-path. Formally, we extend $\psi_G \colon E \to \Gamma$ to $\psi_{G'} \colon E' \to \Gamma$ as follows:

for each edge $e \in E'$,

$$\psi_{G'}(e) := \begin{cases} \psi_G(e) & (e \in E), \\ \alpha & (e \in E' \setminus E), \end{cases}$$

where $\alpha \in \Gamma \setminus \{1_\Gamma\}$ is an arbitrary nonidentity element.

3.3 Reduction to Weighted Matroid Matching

Tanigawa and the author [24] gave a reduction of packing non-zero A-paths to tractable matroid matching by constructing a 2-polymatroid (E, f) associated with a given Γ-labeled graph $G = (V, E)$ with a terminal set $A \subseteq V$, which extends Lovász' idea [16] for Mader's problem. The associated 2-polymatroid (E, f) is defined as follows (cf. [24, Lemma 3.1]): for each $F \subseteq E$,

$$f(F) := \sum_{F' \in \mathrm{comp}(F)} \bigl(2|V(F')| - 2 + \rho(F') - |V(F') \cap A|\bigr),$$

where $\mathrm{comp}(F)$ denotes the partition of F according to the connected components, and $\rho\colon 2^E \to \mathbb{Z}_{\geq 0}$ is defined as

$$\rho(F') := \begin{cases} 2 & (|V(F') \cap A| \geq 1 \text{ and } F' \text{ has a non-zero } A\text{-path or a non-zero cycle}), \\ 1 & \begin{pmatrix} |V(F') \cap A| \geq 1 \text{ and } F' \text{ has no non-zero } A\text{-path and no non-zero cycle, or} \\ |V(F') \cap A| = 0 \text{ and } F' \text{ has a non-zero cycle,} \end{pmatrix}, \\ 0 & (\text{otherwise}). \end{cases}$$

The matchings and parity bases in this associated 2-polymatroid (E, f) are characterized analogously to Lemmas 3 and 4, respectively, as follows.

▶ **Lemma 6** (Tanigawa–Yamaguchi [24, Lemma 3.2]). *An edge set $F \subseteq E$ is a matching in (E, f) if and only if F forms a forest such that every connected component has at most one A-path in G, which is non-zero if exists.*

▶ **Lemma 7.** *An edge set $F \subseteq E$ is a parity base in (E, f) if and only if F forms a spanning forest of G such that every connected component has exactly one A-path in G, which is non-zero.*

Thus, by the same argument as Section 2.3, we can complete an analogous reduction of the shortest disjoint non-zero A-paths problem to weighted matroid matching, in which we construct the associated 2-polymatroid for the auxiliary Γ-labeled graph $G' = (V', E')$ with the terminal set $A' \subseteq V'$ obtained in Section 3.2, and define a weight $w \in \mathbb{R}^{E'}$ as (3).

▶ **Theorem 8.** *The shortest disjoint non-zero A-paths problem reduces to the weighted matroid matching problem.*

3.4 Reduction to Weighted Linear Matroid Parity

The author [25] showed that, under some representability condition for the group Γ, Schrijver's linear representation can be extended to the non-zero A-paths setting. We define $\mathrm{PGL}_2(\mathbb{F}) := \mathrm{GL}_2(\mathbb{F})/\{\, pI_2 \mid p \in \mathbb{F} \setminus \{0\}\,\}$, where $\mathrm{GL}_2(\mathbb{F})$ denotes the general linear group of degree 2 over a field $\mathbb{F}$ (i.e., the set of all nonsingular 2×2 matrices over $\mathbb{F}$ with the ordinary multiplication) and $I_2 \in \mathrm{GL}_2(\mathbb{F})$ the 2×2 identity matrix. Recall that we denote by $\langle y \rangle$ the 1-dimensional linear subspace spanned by a vector y.

▶ **Theorem 9** (Yamaguchi [25, Theorem 1]). *Let Γ be a group and $\mathbb{F}$ a field. If there exists a homomorphism $\rho\colon \Gamma \to \mathrm{PGL}_2(\mathbb{F})$ such that $\rho(\alpha)\binom{1}{0} \notin \left\langle\binom{1}{0}\right\rangle$ for every $\alpha \in \Gamma \setminus \{1_\Gamma\}$, then packing non-zero A-paths in Γ-labeled graphs reduces to linear matroid parity over $\mathbb{F}$.*

For a given Γ-labeled graph $G = (V, E)$ with a terminal set $A \subseteq V$, an associated matrix $Z \in \mathbb{F}^{2V \times 2E}$ is constructed in a similar way to Section 2.1. In the construction, we just modify the definitions (1) and (2) of the linear subspaces L_e and Q_a, respectively, as follows (cf. [25, Section 2.2]):

$$L_e := \{\, x \in (\mathbb{F}^2)^V \mid \rho(\psi_G(e))x(u) + x(w) = \mathbf{0},\ x(v) = \mathbf{0} \ \ (v \in V \setminus \{u, w\}) \,\} \quad (e = uw \in E),$$
$$Q_a := \{\, x \in (\mathbb{F}^2)^V \mid x(a) \in \left\langle\tbinom{1}{0}\right\rangle,\ x(v) = \mathbf{0} \ \ (v \in V \setminus \{a\}) \,\} \quad (a \in A).$$

While Lemmas 3 and 4 are not extended straightforward, the parity bases for this associated matrix Z are characterized as follows by a similar argument based on a characterization [25, Lemmas 9 and 10] of the matchings for Z.

▶ **Lemma 10.** *An edge set $F \subseteq E$ is a parity base for Z*

if *F forms a spanning forest of G such that every connected component has exactly one A-path in G, which is non-zero*;

only if *F forms a spanning subgraph of G such that every connected component has either no cycle and exactly one A-path in G, which is non-zero, or no terminal in A and exactly one cycle.*

We here show the correctness of an analogous reduction of the shortest disjoint non-zero A-paths problem to the weighted linear matroid parity problem under the representability condition of Γ in Theorem 9. We construct the associated matrix $Z \in \mathbb{F}^{2V' \times 2E'}$ for the auxiliary Γ-labeled graph $G' = (V', E')$ with the terminal set $A' \subseteq V'$ obtained in Section 3.2, and define a weight $w \in \mathbb{R}^{E'}$ as (3).

The key observation is that any cycle in G' intersecting no terminal in A' is contained in G. Hence, for any parity base $B \subseteq E'$ for Z that has some cycles (at most one in each connected component by the "only if" part of Lemma 10), there exists a parity base $B' \subseteq E'$ for Z with $w(B') \leq w(B)$ that has no cycle (obtained by replacing edges in the cycles with edges in $E_2 \subseteq E' \setminus E$). Thus we can show an analogous statement to Claim 5 (in particular the part (ii)) in Section 2.3, which suffices to complete the reduction. Note that, even if a minimum-weight parity base for Z with some cycles is obtained by solving weighted linear matroid parity, we can ignore the cycles, whose length must be totally zero.

▶ **Theorem 11.** *Let Γ be a group and $\mathbb{F}$ a field. If there exists a homomorphism $\rho\colon \Gamma \to \mathrm{PGL}_2(\mathbb{F})$ such that $\rho(\alpha)\binom{1}{0} \notin \left\langle\binom{1}{0}\right\rangle$ for every $\alpha \in \Gamma \setminus \{1_\Gamma\}$, then the shortest disjoint non-zero A-paths problem on Γ-labeled graphs reduces to the weighted linear matroid parity problem over $\mathbb{F}$, which can be solved in $\mathrm{O}(|V|^5)$ time.*

3.5 Applications

As shown in [25, Section 3], a variety of groups enjoy such projective representations ρ in Theorems 9 and 11, called *coherent representations*. We here describe several examples with its applications (see [25, Section 3] for more details).

Mader's $\mathcal{S}$-paths. Mader's $\mathcal{S}$-paths are formulated as non-zero A-paths in $\mathbb{Z}$-labeled graphs as follows, where $\mathbb{Z}$ denotes the additive group on the integers. For an undirected graph $G = (V, E)$ with a terminal set $A \subseteq V$ partitioned as $\mathcal{S} := \{A_1, A_2, \ldots, A_t\}$, let $A_0 := V \setminus A$.

Orient each edge in E arbitrarily, and define a label function ψ on the set of oriented edges as follows: for each edge uw with $u \in A_i$ and $w \in A_j$, let $\psi(uw) := i - j \in \mathbb{Z}$. Then, a non-zero A-path in the resulting $\mathbb{Z}$-labeled graph is indeed an $\mathcal{S}$-path in G, and vice versa.

The group $\mathbb{Z}$ admits a coherent representation over $\mathbb{Q}$ in general, and hence our reduction to weighted linear matroid parity works well for any $\mathbb{Z}$-labeled graph.

Odd A-paths. A parity constraint is formulated by using the cyclic group $\mathbb{Z}_2 = \mathbb{Z}/2\mathbb{Z} = \{0, 1\}$, which enjoys a coherent representation over an arbitrary field. Let $G = (V, E)$ be a $\mathbb{Z}_2$-labeled graph with a terminal set $A \subseteq V$, and define $E_1 := \{\, e \in E \mid \psi_G(e) = 1 \,\}$, where the direction of each edge has no meaning since $-0 = 0$ and $-1 = 1$ in $\mathbb{Z}_2$. Then, each non-zero A-path in G is an A-path traversing an odd number of edges in E_1, and vice versa.

Extending the well-known fact that one can find a shortest odd s–t path via the minimum-weight perfect matching problem, one can see that shortest disjoint odd A-paths in an undirected graph can be obtained by finding a minimum-weight perfect matching in an appropriate auxiliary graph. Our reduction exaggerates the problem in a sense, but provides a unified view through the matroid matching framework.

Shortest non-zero s–t path. In general, not only $\mathbb{Z}$ and $\mathbb{Z}_2$, all cyclic groups admit coherent representations. Hence, by the fundamental theorem of finitely generated abelian groups, when $k = 1$ and Γ is a finitely generated abelian group given with its decomposition into q cyclic groups, one can find a shortest non-zero A-path in a Γ-labeled graph by solving the weighted linear matroid parity problem repeatedly q times. When $A = \{s, t\}$ for distinct vertices $s, t \in V$ in particular, one can find a shortest non-zero s–t path in $\mathrm{O}(q|V|^5)$ time. In contrast with an algebraic, randomized algorithm due to Kobayashi and Toyooka [14], which is restricted to the case when Γ is a "finite" abelian group, our result leads to a combinatorial, deterministic solution to a more general case.

Dihedral groups. Even when Γ is non-abelian, there exists a solvable case admitting a coherent representation. A simple example is the dihedral group D_n of degree $n \geq 3$, i.e., $D_n = \langle\, r, R \mid r^n = R^2 = \mathrm{id},\ rR = Rr^{n-1} \,\rangle$.

4 Concluding Remarks

In this paper, we have presented a reduction of a weighted version of Mader's disjoint $\mathcal{S}$-paths problem to weighted linear matroid parity, which leads to the first polynomial-time algorithm for the former problem with the aid of weighted linear matroid parity algorithms due to Iwata [11] and Pap [22]. It should be emphasized that this is not only a solution to a longstanding open problem but also essentially the first successful application of weighted linear matroid parity. We have also discussed a possible extension of our reduction to a generalized framework, packing non-zero A-paths in group-labeled graphs. The shortest disjoint non-zero A-paths problem always reduces to weighted matroid matching, and to weighted linear matroid parity under some representability condition for the group in question.

There remain two open problems related to this work. One is whether the shortest disjoint non-zero A-paths problem is tractable in general or not. While Chudnovsky et al. [4] developed a direct combinatorial algorithm for packing non-zero A-paths, it is quite nontrivial whether their algorithm can be extended to the weighted version. The other is on weighted matroid matching. In Lovász' matroid matching algorithm [17], a linear representation of the input 2-polymatroid is not necessarily required, and it is sufficient to be able to compute an appropriate projection whenever we encounter a nontrivial double circuit (see also [16, 18]). This fact enables us to solve packing non-zero A-paths efficiently via matroid matching in general. While strongly polynomial-time weighted linear matroid parity algorithms were developed by Iwata [11] and Pap [22], these algorithms rely on the linearity. If one can extend Lovász' matroid matching algorithm to the weighted case, then it is expected that a broader class of the weighted matroid matching problem turns out to be tractable, which also leads to a positive answer to the first open problem.

Acknowledgments. I am deeply grateful to Satoru Iwata for giving a large number of helpful comments and suggestions. I also appreciate insightful and instructive comments of the anonymous referees.

References

1 A. Björklund and T. Husfeldt. Shortest two disjoint paths in polynomial time. In *Proceedings of ICALP 2014*, pages 211–222, 2014.

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

3 H. Y. Cheung, L. C. Lau, and K. M. Leung. Algebraic algorithms for linear matroid parity problems. *ACM Transactions on Algorithms*, 10(3):No. 10, 2014.

4 M. Chudnovsky, W. H. Cunningham, and J. Geelen. An algorithm for packing non-zero A-paths in group-labelled graphs. *Combinatorica*, 28(2):145–161, 2008.

5 M. Chudnovsky, J. Geelen, B. Gerards, L. Goddyn, M. Lohman, and P. Seymour. Packing non-zero A-paths in group-labelled graphs. *Combinatorica*, 26(5):521–532, 2006.

6 A. Dress and L. Lovász. On some combinatorial properties of algebraic matroids. *Combinatorica*, 7(1):39–48, 1987.

7 H. N. Gabow and M. Stallmann. An augmenting path algorithm for linear matroid parity. *Combinatorica*, 6(2):123–150, 1986.

8 T. Gallai. Maximum-Minimum Sätze und verallgemeinerte Faktoren von Graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 12(1):131–173, 1961.

9 H. Hirai and H. Namba. Shortest $(A + B)$-path packing via hafnian. *arXiv preprints*, arXiv:1603.08073, 2016.

10 H. Hirai and G. Pap. Tree metrics and edge-disjoint S-paths. *Mathematical Programming, Ser. A*, 147(1):81–123, 2014.

11 S. Iwata. A weighted linear matroid parity algorithm. In *Proceedings of the 8th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications*, pages 251–259, 2013.

12 A. V. Karzanov. Edge-disjoint T-paths of minimum total cost. Technical Report STAN-CS-92-1465, Department of Computer Science, Stanford University, 1993.

13 A. V. Karzanov. Multiflows and disjoint paths of minimum total cost. *Mathematical Programming*, 78(2):219–242, 1997.

14 Y. Kobayashi and S. Toyooka. Finding a shortest non-zero path in group-labeled graphs via permanent computation. *Algorithmica*, to appear.

15 E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

16 L. Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Ser. B*, 28(2):208–236, 1980.

17 L. Lovász. The matroid matching problem. *Algebraic Methods in Graph Theory: Colloquia Mathematica Societatis János Bolyai*, 25(2):495–517, 1981.

18 L. Lovász and M. D. Plummer. *Matching Theory.* Akadémiai Kiadó, 1986.

19 W. Mader. Über die Maximalzahl kreuzungsfreier$H$-Wege. *Archiv der Mathematik*, 31(1):387–402, 1978.

20 J. B. Orlin. A fast, simpler algorithm for the matroid parity problem. In *Proceedings of IPCO 2008*, pages 240–258, 2008.

21 G. Pap. A polynomial time algorithm for weighted node-disjoint S-paths. In *Proceedings of the 7th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 322–331, 2011.

22 G. Pap. Weighted linear matroid matching. In *Proceedings of the 8th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications*, pages 411–413, 2013.

23 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, 2003.

24 S. Tanigawa and Y. Yamaguchi. Packing non-zero A-paths via matroid matching. *Discrete Applied Mathematics*, 214(11):169–178, 2016.

25 Y. Yamaguchi. Packing A-paths in group-labelled graphs via linear matroid parity. *SIAM Journal on Discrete Mathematics*, 30(1):474–492, 2016.

The (1|1)-Centroid Problem on the Plane Concerning Distance Constraints[*]

Hung-I Yu[1], Tien-Ching Lin[2], and Der-Tsai Lee[3]

1 Institute of Information Science, Academia Sinica, Taipei 115, Taiwan
herbert@iis.sinica.edu.tw

2 Institute of Information Science, Academia Sinica, Taipei 115, Taiwan
kero@iis.sinica.edu.tw

3 Institute of Information Science, Academia Sinica, Taipei 115, Taiwan
dtlee@iis.sinica.edu.tw

Abstract

In 1982, Drezner proposed the (1|1)-centroid problem on the plane, in which two players, called the leader and the follower, open facilities to provide service to customers in a competitive manner. The leader opens the first facility, and then the follower opens the second. Each customer will patronize the facility closest to him (ties broken in favor of the leader's one), thereby decides the market share of the two players. The goal is to find the best position for the leader's facility so that his market share is maximized. The best algorithm for this problem is an $O(n^2 \log n)$-time parametric search approach, which searches over the space of possible market share values.

In the same paper, Drezner also proposed a general version of (1|1)-centroid problem by introducing a minimal distance constraint R, such that the follower's facility is not allowed to be located within a distance R from the leader's. He proposed an $O(n^5 \log n)$-time algorithm for this general version by identifying $O(n^4)$ points as the candidates of the optimal solution and checking the market share for each of them. In this paper, we develop a new parametric search approach searching over the $O(n^4)$ candidate points, and present an $O(n^2 \log n)$-time algorithm for the general version, thereby closing the $O(n^3)$ gap between the two bounds.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases competitive facility, Euclidean plane, parametric search

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.64

1 Introduction

In 1929, economist Hotelling introduced the first competitive location problem in his seminal paper [13]. Since then, the subject of competitive facility location has been extensively studied by researchers in the fields of spatial economics, social and political sciences, and operations research, and spawned hundreds of contributions in the literature. The interested reader is referred to the following survey papers [1, 4, 8, 9, 10, 12, 17, 19].

Hakimi [11] and Drezner [6] independently proposed a series of competitive location problems in a leader-follower framework. The framework is briefly described as follows. There are n customers in the market, and each is endowed with a certain buying power. Two players, called the *leader* and the *follower*, sequentially open facilities to attract the

[*] Research supported by the Ministry of Science and Technology under Grants MOST 104-2221-E-001-031, 104-2221-E-001-032.

27th International Symposium on Algorithms and Computation (ISAAC 2016).
Editor: Seok-Hee Hong; Article No. 64; pp. 64:1–64:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

buying power of customers. At first, the leader opens his p facilities, and then the follower opens another r facilities. Each customer will patronize the closest facility with all buying power (ties broken in favor of the leader's ones), thereby deciding the market share of the two players. Since both players ask for market share maximization, two competitive facility location problems are defined. Given that the leader locates his p facilities at the set X_p of p points, the follower wants to locate his r facilities in order to attract the most buying power, called the $(r|X_p)$-*medianoid* problem. On the other hand, knowing that the follower will react with maximization strategy, the leader wants to locate his p facilities in order to retain the most buying power against the competition, called the $(r|p)$-*centroid* problem.

Drezner [6] first proposed to study the two competitive facility location problems on the Euclidean plane. Since then, several related results [5, 7, 12, 14] have been obtained for different values of r and p. Due to page limit, here we introduce only previous results about the case $r = p = 1$. For the $(1|X_1)$-medianoid problem, Drezner [6] showed that there exists an optimal solution arbitrarily close to X_1, and solved the problem in $O(n \log n)$ time by a sweeping technique. Later, Lee and Wu [14] obtained an $\Omega(n \log n)$ lower bound for the $(1|X_1)$-medianoid problem, and thus proved the optimality of Drezner's result. For the $(1|1)$-centroid problem, Drezner [6] developed a parametric search approach that searches over the space of $O(n^2)$ possible market share values, along with an $O(n^4)$-time test procedure constructing and solving a linear program of $O(n^2)$ constraints, and gave an $O(n^4 \log n)$-time algorithm. Then, by improving the test procedure via Megiddo's linear-time result [16] for solving linear programs, Hakimi [12] reduced the time complexity to $O(n^2 \log n)$.

In [6], Drezner also proposed a more general setting for the leader-follower framework by introducing a *minimal distance constraint* $R \geq 0$ into the $(1|X_1)$-medianoid problem and the $(1|1)$-centroid problem, such that the follower's facility is not allowed to be located within a distance R from the leader's. The augmented problems are respectively called the $(1|X_1)_R$-*medianoid* problem and $(1|1)_R$-*centroid* problem in this paper. Drezner showed that the $(1|X_1)_R$-medianoid problem can also be solved in $O(n \log n)$ time by using nearly the same proof and technique as for the $(1|X_1)$-medianoid problem. However, for the $(1|1)_R$-centroid problem, he argued that it is hard to generalize the parametric approach approach for the $(1|1)$-centroid problem to solve this general version, due to the change of problem properties. Then, he gave an $O(n^5 \log n)$-time algorithm by identifying $O(n^4)$ candidate points on the plane, which contain at least one optimal solution, and performing medianoid computation on each of them. So far, the $O(n^3)$ gap between the two centroid problems remains open.

In this paper, we propose an $O(n^2 \log n)$-time algorithm for the $(1|1)_R$-centroid problem on the Euclidean plane, thereby closing the gap that has existed for decades. Instead of searching over market share values, we develop a new approach based on a different parametric search technique by searching over the $O(n^4)$ candidate points mentioned in [6]. This is made possible by making a critical observation on the distribution of optimal solutions for the $(1|X_1)_R$-medianoid problem given X_1, which provides us a useful tool to prune candidate points with respect to X_1. We then extend the usage of this tool to design a key procedure to prune candidates with respect to a given vertical line. Due to page limits, most of the proofs are omitted.

The rest of this paper is organized as follows. Section 2 gives formal problem definitions and describes previous results in [6, 12]. In Section 3, we make the observation on the $(1|X_1)_R$-medianoid problem, and use it to find a "local" centroid on a given line. This result is then extended as a new pruning procedure with respect to any given line in Section 4, and utilized in our parametric search approach for the $(1|1)_R$-centroid problem. Finally, in Section 5, we give some concluding remarks.

2 Notations and Preliminary Results

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of n points on the Euclidean plane $\mathbb{R}^2$, as the representatives of the n customers. Each point $v_i \in V$ is assigned a positive weight $w(v_i)$, representing its buying power. To simplify the presentation, we assume that the points in V are in general position, that is, no three points are collinear and no two points share a common x or y-coordinate.

Let $d(u, w)$ denote the Euclidean distance between any two points $u, w \in \mathbb{R}^2$. For any set Z of points on the plane, we define $W(Z) = \sum\{w(v)|v \in V \bigcap Z\}$. Suppose that the leader has located his facility at $X_1 = \{x\}$, which is shortened as x for simplicity. Due to the minimal distance constraint R mentioned in [6], any point $y' \in \mathbb{R}^2$ with $d(y', x) < R$ is infeasible to be the follower's choice. If the follower locates his facility at some feasible point y, the set of customers patronizing y instead of x is defined as $V(y|x) = \{v \in V|d(v, y) < d(v, x)\}$, with their total buying power $W(y|x) = W(V(y|x))$. Then, the largest market share that the follower can capture is denoted by the function $W^*(x) = \max_{y\in\mathbb{R}^2, d(y,x)\geq R} W(y|x)$, which is called the *weight loss* of x. Given a point $x \in \mathbb{R}^2$, the $(1|x)_R$-medianoid problem is to find a $(1|x)_R$*-medianoid*, which is a feasible point $y^* \in \mathbb{R}^2$ such that $W(y^*|x) = W^*(x)$.

In contrast, the leader tries to minimize the weight loss of his own facility by finding a point $x^* \in \mathbb{R}^2$ such that $W^*(x^*) \leq W^*(x)$ for any point $x \in \mathbb{R}^2$. The $(1|1)_R$-centroid problem is to find a $(1|1)_R$*-centroid*, which is a point x^* minimizing its own weight loss. Note that, when $R = 0$, the two problems degenerate to the $(1|x)$-medianoid problem and $(1|1)$-centroid problem.

2.1 Previous approaches

In this subsection, we briefly review previous results for the $(1|x)_R$-medianoid, $(1|1)$-centroid, and $(1|1)_R$-centroid problems in [6, 12], so as to derive properties essential to our approach.

Let L be an arbitrary line, which partitions the Euclidean plane into two half-planes. For any point $y \notin L$, we define $H(L, y)$ as the closed half-plane including L and y, and $H^-(L, y)$ as the open half-plane $H(L, y)\backslash L$. For any two distinct points $x, y \in \mathbb{R}^2$, let $B(y|x)$ denote the perpendicular bisector of $\overline{xy}$, the line segment connecting x and y.

Given an arbitrary point $x \in \mathbb{R}^2$, we first describe the algorithm for finding a $(1|x)_R$-medianoid in [6]. Let y be a feasible point other than x, and y' be some point on the open line segment $\overline{xy}\backslash\{x, y\}$. We can see that $H^-(B(y|x), y) \subset H^-(B(y'|x), y')$, which implies the fact that $W(y'|x) = W(H^-(B(y'|x), y')) \geq W(H^-(B(y|x), y)) = W(y|x)$, It shows that moving y toward x does not diminish its weight capture, thereby follows the lemma.

▶ **Lemma 1** ([6]). *There exists a* $(1|x)_R$*-medianoid in* $\{y \mid y \in \mathbb{R}^2, d(x, y) = R\}$.

For any point $z \in \mathbb{R}^2$, let $C_R(z)$ and $C_\gamma(z)$ be the circles centered at z with radii R and $\gamma = R/2$, respectively. By Lemma 1, finding a $(1|x)_R$-medianoid can be reduced to searching a point y on $C_R(x)$ maximizing $W(y|x)$. Since the perpendicular bisector $B(y|x)$ of each point y on $C_R(x)$ is a tangent line to the circle $C_\gamma(x)$, the searching of y on $C_R(x)$ is equivalent to finding a tangent line to $C_\gamma(x)$ that partitions the most weight from x. The latter problem can be solved in $O(n \log n)$ time as follows. For each $v \in V$ outside $C_\gamma(x)$, we calculate its two tangent lines to $C_\gamma(x)$. Then, by sorting these tangent lines according to the polar angles of their corresponding tangent points with respect to x, we can use an angle sweeping technique to check how much weight they partition.

▶ **Theorem 2** ([6]). *Given a point* $x \in \mathbb{R}^2$*, the* $(1|x)_R$*-medianoid problem can be solved in* $O(n \log n)$ *time.*

Next, we describe the algorithm of the $(1|1)_R$-centroid problem in [6]. Let S be a subset of V. We define $\mathcal{C}(S)$ to be the set of all circles $C_\gamma(v)$, $v \in S$, and $CH(\mathcal{C}(S))$ to be the convex hull of these circles. For any positive number W_0, let $I(W_0)$ be the intersection of all convex hulls $CH(\mathcal{C}(S))$, where $S \subseteq V$ and $W(S) \geq W_0$. Drezner [6] argued that the set of all $(1|1)_R$-centroids is equivalent to the intersection $I(W_0)$ for the smallest possible W_0. We slightly clarify his argument below. Let $\mathcal{W} = \{W(y|x) \mid x, y \in \mathbb{R}^2, d(x,y) \geq R\}$. The following lemma can be obtained.

▶ **Lemma 3.** *Let W_0^* be the smallest number in $\mathcal{W}$ such that $I(W_0^*)$ is not null. A point x is a $(1|1)_R$-centroid if and only if $x \in I(W_0^*)$.*

Although it is hard to compute $I(W_0^*)$ itself, we can find its vertices as solutions to the $(1|1)_R$-centroid problem. Let $\mathcal{T}$ be the set of outer tangent lines of all pairs of circles in $\mathcal{C}(V)$. For any subset $S \subseteq V$, the boundary of $CH(\mathcal{C}(S))$ is formed by segments of lines in $\mathcal{T}$ and arcs of circles in $\mathcal{C}(V)$. Since $I(W_0)$ is an intersection of such convex hulls, its vertices must fall within the set of intersection points between lines in $\mathcal{T}$, between circles in $\mathcal{C}(V)$, and between one line in $\mathcal{T}$ and one circle in $\mathcal{C}(V)$. Let $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{C}(V)$, and $\mathcal{T} \times \mathcal{C}(V)$ denote the three sets of intersection points, respectively. We have the lemma below.

▶ **Lemma 4** ([6])**.** *There exists a $(1|1)_R$-centroid in $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{C}(V)$, and $\mathcal{T} \times \mathcal{C}(V)$.*

Obviously, there are at most $O(n^4)$ intersection points, which can be viewed as the *candidates* of being $(1|1)_R$-centroids. Drezner thus gave an algorithm by evaluating the weight loss of each candidate by Theorem 2.

▶ **Theorem 5** ([6])**.** *The $(1|1)_R$-centroid problem can be solved in $O(n^5 \log n)$ time.*

We remark that, when $R = 0$, $CH(\mathcal{C}(S))$ for any $S \subseteq V$ degenerates to a convex polygon, so does $I(W_0)$ for any given W_0, if not null. Drezner [6] proved that in this case $I(W_0)$ is equivalent to the intersection of all half-planes H with $W(H) \geq W_0$. Thus, whether $I(W_0)$ is null can be determined by constructing and solving a linear program of $O(n^2)$ constraints, which takes $O(n^2)$ time by Megiddo's result [16]. Since $|\mathcal{W}| = O(n^2)$, according to Lemma 3 the $(1|1)$-centroid problem can be solved in $O(n^2 \log n)$ time [12], by applying parametric search over $\mathcal{W}$ for W_0^*. Unfortunately, it is hard to generalize this idea to the case $R > 0$.

3 Local $(1|1)_R$-Centroid within a Line

In this section, we analyze the properties of $(1|x)_R$-medianoids of a given point x in Subsection 3.1, and derive a procedure that prunes candidate points with respect to x. Applying this procedure, we study a restricted version of the $(1|1)_R$-centroid problem in Subsection 3.2, in which the leader's choice is limited to a given line L, and obtain an $O(n \log^2 n)$-time algorithm. The algorithm is then extended as the basis of the vertical-line test procedure for the parametric search approach in Section 4.

3.1 Pruning with Respect to a Point

Given a point $x \in \mathbb{R}^2$ and an angle θ between 0 and 2π, let $y(\theta|x)$ be the point on $C_R(x)$ with polar angle θ with respect to x.[1] We define $MA(x) = \{\theta \,|\, W(y(\theta|x)|x) = W^*(x), 0 \leq \theta < 2\pi\}$, that is, the set of angles θ maximizing $W(y(\theta|x)|x)$. It can be observed that, for any

[1] We assume that a polar angle is measured counterclockwise from the positive x-axis.

$\theta \in MA(x)$ and sufficiently small ϵ, both $\theta + \epsilon$ and $\theta - \epsilon$ belong to $MA(x)$, because each $v \in V(y(\theta|x)|x)$ does not intersect $B(y(\theta|x)|x)$ by definition. This implies that angles in $MA(x)$ form open angle interval(s) of non-zero length.

To simplify the terms, let $W(\theta|x) = W(y(\theta|x)|x)$ and $B(\theta|x) = B(y(\theta|x)|x)$ in the remaining parts. Also, let $F(\theta|x)$ be the line passing through x and parallel to $B(\theta|x)$. The following lemma provides the basis for pruning candidates.

▶ **Lemma 6.** *Let $x \in \mathbb{R}^2$ be an arbitrary point, and θ be an angle in $MA(x)$. For any point $x' \notin H^-(F(\theta|x), y(\theta|x))$, $W^*(x') \geq W^*(x)$.*

This lemma tells us that, given a point x and an angle $\theta \in MA(x)$, all points not in $H^-(F(\theta|x), y(\theta|x))$ can be ignored while finding $(1|1)_R$-centroids, as their weight losses are no less than that of x. Besides, the distribution of angles in $MA(x)$ is also meaningful. Let $CA(x)$ be the minimum angle interval covering all angles in $MA(x)$, and $\delta(CA(x))$ be its angle span in radians. Since $MA(x)$ consists of open angle interval(s) of non-zero length, $CA(x)$ is also an open interval and $\delta(CA(x)) > 0$. Moreover, we can derive the following.

▶ **Lemma 7.** *If $\delta(CA(x)) > \pi$, x is a $(1|1)_R$-centroid.*

We call a point x satisfying Lemma 7 a *strong $(1|1)_R$-centroid*, since its discovery gives an immediate solution to the $(1|1)_R$-centroid problem. Note that there are problem instances in which no strong $(1|1)_R$-centroids exist.

Suppose that $\delta(CA(x)) \leq \pi$ for some point $x \in \mathbb{R}^2$. Let $Wedge(x)$ denote the *wedge* of x, defined as the intersection of the two half-planes $H(F(\theta_b|x), y(\theta_b|x))$ and $H(F(\theta_e|x), y(\theta_e|x))$, where θ_b and θ_e are the beginning and ending angles of $CA(x)$, respectively. $Wedge(x)$ consists of the two half-lines extending from x, defined by $F(\theta_e|x)$ and $F(\theta_b|x)$, and the infinite region lying between them. The counterclockwise (CCW) angle between the two half-lines is denoted by $\delta(Wedge(x))$. Since $0 < \delta(CA(x)) \leq \pi$, we have that $Wedge(x) \neq \emptyset$ and $0 \leq \delta(Wedge(x)) < \pi$.

It should be emphasized that $Wedge(x)$ is a computational byproduct of $CA(x)$ when x is not a strong $(1|1)_R$-centroid. In other words, not every point has its wedge. Therefore, we make the following assumption (or restriction) in order to avoid the misuse of $Wedge(x)$.

▶ **Assumption 8.** *Whenever $Wedge(x)$ is mentioned, the point x has been found not to be a strong $(1|1)_R$-centroid, either by computation or by properties. Equivalently, $\delta(CA(x)) \leq \pi$.*

The following lemma makes $Wedge(x)$ our main tool for prune-and-search. (Note that its proof is not trivial, since by definition θ_b and θ_e do not belong to $CA(x)$ and $MA(x)$.)

▶ **Lemma 9.** *Let $x \in \mathbb{R}^2$ be an arbitrary point. For any point $x' \notin Wedge(x)$, $W^*(x') \geq W^*(x)$.*

The computation of $Wedge(x)$ is simple. We first compute $W^*(x)$ in $O(n \log n)$ time by Theorem 2. Then, by reusing the sweeping technique, we can obtain $MA(x)$ and $CA(x)$ in $O(n)$ time and, if x is not a strong $(1|1)_R$-centroid, $Wedge(x)$ in $O(1)$ time.

▶ **Lemma 10.** *Given a point $x \in \mathbb{R}^2$, $MA(x)$, $CA(x)$, and $Wedge(x)$ can be computed in $O(n \log n)$ time.*

3.2 Searching on a Line

Although wedges can be used to prune candidate points, the performance is not stable, since wedges of different points have distinct angle intervals and spans. However, they work fine with lines by Assumption 8. Here we show how to use the wedges to compute a *local optimal* point on a given line, i.e. a point x with $W^*(x) \leq W^*(x')$ for any point x' on the line.

Let L be an arbitrary line, which is assumed to be non-horizontal for ease of discussion. For any point x on L, we can compute $Wedge(x)$ and make use of it for pruning purposes by defining its *direction* with respect to L. Since $\delta(Wedge(x)) < \pi$ by definition, there are only three categories of directions according to the intersection of $Wedge(x)$ and L:
Upward – the intersection is the half-line of L above and including x;
Downward – the intersection is the half-line of L below and including x;
Sideward – the intersection is x itself.
If $Wedge(x)$ is sideward, x is a local optimal point on L, since by Lemma 9 $W^*(x) \leq W^*(x') \; \forall \; x' \in L$. Otherwise, either $Wedge(x)$ is upward or downward, the points on the opposite half of L can be pruned by Lemma 9. It shows that computing wedges acts as a predictable tool for pruning points on L.

Next, we list sets of *breakpoints* on L in which a local optimal point exists. Recall that $\mathcal{T}$ is the set of outer tangent lines of all pairs of circles in $\mathcal{C}(V)$. We define the $\mathcal{T}$*-breakpoints* as the set $L \times \mathcal{T}$ of intersection points between L and lines in $\mathcal{T}$, and the $\mathcal{C}$*-breakpoints* as the set $L \times \mathcal{C}(V)$ of intersection points between L and circles in $\mathcal{C}(V)$. (Note that outer tangent lines parallel to L can be ignored while defining breakpoints.) We have the following lemma for breakpoints.

▶ **Lemma 11.** *There exists a local optimal point x^*_L which is also a breakpoint.*

Since $|L \times \mathcal{T}| = O(n^2)$ and $|L \times \mathcal{C}(V)| = O(n)$, we can sort all breakpoints on L in $O(n^2 \log n)$ time according to the decreasing order of their y-coordinates, and, by Lemma 11, perform binary search via wedges to find a local optimal point x^*_L among them in $O(n \log n \times \log n) = O(n \log^2 n)$ time. Thus, the restricted problem is trivially solved in $O(n^2 \log n)$ time. In the following, we however give a more complicated algorithm to deal with the case that the line L is given as a query. The algorithm consists of an $O(n^2 \log n)$-time preprocessing and an $O(n \log^2 n)$-time procedure to find x^*_L on L.

The preprocessing itself is very simple. For each point $v \in V$, we compute a sequence $P(v)$, consisting of points in $V \backslash \{v\}$ sorted in increasing order of their polar angles with respect to v. The computation for all $v \in V$ takes $O(n^2 \log n)$ time in total. We will show that, for any given line L, $O(n)$ sorted sequences of breakpoints can be obtained from these pre-computed sequences in $O(n \log n)$ time, and can be used to replace the role of the sorted sequence of all breakpoints while performing binary search on L.

For any two points $v \in V$ and $z \in \mathbb{R}^2$, let $T^r(z|v)$ be the outer tangent line of $C_\gamma(v)$ and $C_\gamma(z)$ to the right of the line from v to z. Similarly, let $T^l(z|v)$ be the outer tangent line to the left. Moreover, let $t^r_L(z|v)$ and $t^l_L(z|v)$ be the points at which $T^r(z|v)$ and $T^l(z|v)$ intersect with L, respectively. We partition $\mathcal{T}$ into $O(n)$ sets $\mathcal{T}^r(v) = \{T^r(v_i|v) | v_i \in V \backslash \{v\}\}$ and $\mathcal{T}^l(v) = \{T^l(v_i|v) | v_i \in V \backslash \{v\}\}$ for $v \in V$, and for each set consider its corresponding $\mathcal{T}$-breakpoints independently.

We discuss the set of $\mathcal{T}$-breakpoints $L \times \mathcal{T}^r(v)$ first. Let v be an arbitrary point in V. By general position assumption, we can observe that, in some consecutive subsequences of $P(v)$, points v_i are listed in the same order as their corresponding breakpoints $t^r_L(v_i|v)$ in decreasing y-coordinates, whereas the order of points in other consecutive subsequences correspond to that of breakpoints in increasing y-coordinates. Thus, we can partition $P(v)$ into $O(1)$ consecutive subsequences to represent $L \times \mathcal{T}^r(v)$, as shown in the following.

▶ **Lemma 12.** *For each $v \in V$, we can construct $O(1)$ sequences of $\mathcal{T}$-breakpoints on L in $O(\log n)$ time, which satisfy the following statements:*

(a) *Each sequence is of length $O(n)$.*
(b) *Breakpoints in each sequence are sorted in decreasing y-coordinates.*
(c) *The union of breakpoints in all sequences form $L \times \mathcal{T}^r(v)$.*

By Lemma 12, the $O(1)$ sorted sequences can replace the role of $L \times \mathcal{T}^r(v)$. Symmetrically, we can also obtain a similar lemma constructing another $O(1)$ sorted sequences of breakpoints to replace $L \times \mathcal{T}^l(v)$. By applying such a construction to all $v \in V$, in $O(n \log n)$ time we can construct total $O(n)$ sorted sequences of length $O(n)$, whose union is equivalent to $L \times \mathcal{T}$. Moreover, since $|L \times \mathcal{C}(V)| = O(n)$, we can directly arrange them into a sorted sequence in $O(n \log n)$ time. Consequently, all breakpoints on L are partitioned into $N_0 = O(n)$ sequences, each of length $O(n)$ and sorted in decreasing y-coordinates.

The searching of x^*_L in the N_0 sorted sequences is done by parametric search technique for parallel binary searches, introduced in [2]. For each sorted sequence, we obtain its middle element, and associate it with a weight. Then, we compute the *weighted median* x of the N_0 middle elements [18]. Finally, we apply Lemma 10 on x, and prune breakpoints not in $Wedge(x)$ for every sequence. By proper weighting scheme (details omitted), the total number of breakpoints in all sequences will be reduced by a constant factor. By repeating the above process, we can find x^*_L in at most $O(\log n)$ iterations.

The running time is analyzed as follows. As discussed above, constructing the N_0 sorted sequences takes $O(n \log n)$ time. The pruning process requires at most $O(\log n)$ iterations. At each iteration, we compute the weighted median x in $O(N_0) = O(n)$ time by [18], and $Wedge(x)$ in $O(n \log n)$ time by Lemma 10. Finally, pruning every sequences takes $O(n)$ time. Thus, the total running time is $O(n \log n) + O(\log n) \times O(n \log n) = O(n \log^2 n)$ time.

▶ **Lemma 13.** *With an $O(n^2 \log n)$-time preprocessing, given an arbitrary line L, a local optimal point x^*_L on L can be computed in $O(n \log^2 n)$ time.*

4 $(1|1)_R$-Centroid on the Plane

In this section, we study the $(1|1)_R$-centroid problem and propose an improved algorithm of time complexity $O(n^2 \log n)$. This algorithm is as efficient as the best-so-far algorithm for the $(1|1)$-centroid problem in [12], but based on a completely different approach.

In Subsection 4.1, we extend the algorithm of Lemma 13 to develop a procedure allowing us to prune candidate points on the plane with respect to a given vertical line. Then, in Subsection 4.2, we show how to compute a $(1|1)_R$-centroid in $O(n^2 \log n)$ time based on this newly-developed pruning procedure.

4.1 Pruning with Respect to a Vertical Line

Let L be an arbitrary vertical line on the plane. We call the half-plane strictly to the left of L the *left plane* of L and the one strictly to its right the *right plane* of L. A sideward wedge of some point on L is said to be *rightward* (resp. *leftward*) if it intersects the right (resp. left) plane of L. We can observe that, if there is some point $x \in L$ such that $Wedge(x)$ is rightward, every point x' on the left plane of L can be pruned, since $W^*(x') \geq W^*(x)$ by Lemma 9. Similarly, if $Wedge(x)$ is leftward, points on the right plane of L can be pruned. Although the power of wedges is not fully exerted in this way, pruning via vertical lines and sideward wedges is superior than directly via wedges due to predictable pruning regions.

Therefore, in this subsection we describe how to design a procedure that enables us to prune either the left or the right plane of a given vertical line L. As mentioned above, the key point is the searching of sideward wedges on L. It is achieved by carrying out three conditional phases. In the first phase, we try to find some proper breakpoints with sideward wedges. If failed, we pick some representative point in the second phase and check its wedge to determine whether or not sideward wedges exist. Finally, in case of their nonexistence, we show that their functional alternative can be computed, called the *pseudo wedge*, that still allows us to prune the left or right plane of L. In the following, we develop a series of lemmas to demonstrate the details of the three phases.

▶ **Lemma 14.** *Let x be an arbitrary point on L. If $Wedge(x)$ is either upward or downward, for any point $x' \in L \backslash Wedge(x)$, $Wedge(x')$ has the same direction as $Wedge(x)$.*

Following from this lemma, if there exist two arbitrary points x_1 and x_2 on L with their wedges downward and upward, respectively, we can derive that x_1 must be strictly above x_2, and that points with sideward wedges or even strong $(1|1)_R$-centroids can lie only between x_1 and x_2. Thus, we can find sideward wedges between some specified downward and upward wedges. Let x_D be the lowermost breakpoint on L with its wedge downward, x_U the uppermost breakpoint on L with its wedge upward, and G_{DU} the open segment $\overline{x_D x_U} \backslash \{x_D, x_U\}$. (For ease of discussion, we assume that both x_D and x_U exist on L, and show how to resolve this assumption later by constructing a bounding box.) Again, x_D is strictly above x_U. Also, we have the following corollary by their definitions.

▶ **Corollary 15.** *If there exist breakpoints in the segment G_{DU}, for any such breakpoint x, either x is a strong $(1|1)_R$-centroid or $Wedge(x)$ is sideward.*

Given x_D and x_U, the first phase can thus be done by checking whether there exist breakpoints in G_{DU} and picking any of them if exist. Supposing that the picked one is not a strong $(1|1)_R$-centroid, a sideward wedge is found by Corollary 15 and can be used for pruning. Notice that, when there are two or more such breakpoints, one may question whether their wedges are of the same direction, as different directions result in inconsistent pruning results. The following lemma answers the question in the positive.

▶ **Lemma 16.** *Let x_1, x_2 be two distinct points on L, where x_1 is strictly above x_2 and none of them is a strong $(1|1)_R$-centroid. If $Wedge(x_1)$ and $Wedge(x_2)$ are both sideward, they are either both rightward or both leftward.*

The second phase deals with the case that no breakpoint exists between x_D and x_U by determining the wedge direction of a representative point of all inner points in G_{DU}. The following lemma enables us to pick an arbitrary point in G_{DU} as the representative.

▶ **Lemma 17.** *When there is no breakpoint between x_D and x_U, any two distinct points x_1, x_2 in G_{DU} have the same wedge direction, if they are not strong $(1|1)_R$-centroids.*

We choose the bisector point x_B of x_D and x_U as the representative. If x_B is not a strong $(1|1)_R$-centroid and $Wedge(x_B)$ is sideward, the second phase finishes with a sideward wedge found. Otherwise, if $Wedge(x_B)$ is downward or upward, we can derive the following and have to invoke the third phase.

▶ **Lemma 18.** *If there is no breakpoint between x_D and x_U and $Wedge(x_B)$ is not sideward, there exist neither strong $(1|1)_R$-centroids nor points with sideward wedges on L.*

When L satisfies Lemma 18, it consists of only points with downward or upward wedges, and is said to be *non-leaning.* Obviously, our pruning strategy via sideward wedges could not apply to such non-leaning lines. The third phase overcomes this obstacle by constructing a functional alternative of sideward wedges, called the pseudo wedge, on either x_D or x_U, so that pruning with respect to L is still achievable. We start with auxiliary lemmas.

▶ **Lemma 19.** *If L is non-leaning, $W^*(x_D) \neq W^*(x_U)$.*

Let $W_1 = \max\{W^*(x_D), W^*(x_U)\}$. We are going to define the pseudo wedge on either x_U or x_D, depending on which one has the smaller weight loss. We consider first the case that $W^*(x_D) > W^*(x_U)$, and obtain the following.

▶ **Lemma 20.** *If L is non-leaning and $W^*(x_D) > W^*(x_U)$, there exists one angle θ for x_U, where $\pi \leq \theta \leq 2\pi$, such that $W(H(B(\theta|x_U), y(\theta|x_U))) \geq W_1$.*

Let θ_U be an arbitrary angle satisfying the conditions of Lemma 20. We apply the line $F(\theta_U|x_U)$ for trimming the region of $Wedge(x_U)$, so that a sideward wedge can be obtained. Let $PW(x_U)$, called the *pseudo wedge* of x_U, denote the intersection of $Wedge(x_U)$ and $H(F(\theta_U|x_U), y(\theta_U|x_U))$. Deriving from the three facts that $Wedge(x_U)$ is upward, $\delta(Wedge(x_U)) < \pi$, and $\pi \leq \theta_U \leq 2\pi$, we can observe that either $PW(x_U)$ is x_U itself, or it intersects only one of the right and left planes of L. In the two circumstances, $PW(x_U)$ is said to be *closed* or *sideward*, respectively. The pseudo wedge has similar functionality as wedges, as shown in the following corollary.

▶ **Corollary 21.** *For any point $x' \notin PW(x_U)$, $W^*(x') \geq W^*(x_U)$.*

By this lemma, if $PW(x_U)$ is found to be sideward, points on the opposite half-plane with respect to L can be pruned. If $PW(x_U)$ is closed, x_U becomes another kind of strong $(1|1)_R$-centroids, in the meaning that it is also an immediate solution to the $(1|1)_R$-centroid problem. Without confusion, we call x_U a *conditional $(1|1)_R$-centroid* in the latter case.

On the other hand, considering the opposite case that $W^*(x_D) < W^*(x_U)$, we can also obtain an angle θ_D and a pseudo wedge $PW(x_D)$ for x_D by symmetric arguments. Then, either $PW(x_D)$ is sideward and the opposite side of L can be pruned, or x_D itself is a conditional $(1|1)_R$-centroid. Thus, the third phase overcomes the obstacle of the nonexistence of sideward wedges.

Recall that the three phases of searching sideward wedges is based on the existence of x_D and x_U on L, which was not guaranteed before. Here we show that, by constructing appropriate border lines, we can guarantee the existence of x_D and x_U while searching between these border lines. The *bounding box* is defined as the smallest axis-aligned rectangle that encloses all circles in $\mathcal{C}(V)$. Clearly, any point x outside the box satisfies that $W^*(x) = W(V)$ and must not be a $(1|1)_R$-centroid. Thus, given a vertical line not intersecting the box, the half-plane to be pruned is trivially decided. Moreover, let T_{top} and T_{btm} be two arbitrary horizontal lines strictly above and below the box, respectively. We can obtain the following.

▶ **Lemma 22.** *Let L be an arbitrary vertical line intersecting the bounding box, and x'_D and x'_U denote its intersection points with T_{top} and T_{btm}, respectively. $Wedge(x'_D)$ is downward and $Wedge(x'_U)$ is upward.*

According to this lemma, by inserting T_{top} and T_{btm} into $\mathcal{T}$, the existence of x_D and x_U is enforced for any vertical line intersecting the bounding box. Besides, the insertion does not affect the correctness of all lemmas developed so far.

Summarizing the above discussion, the whole picture of our desired pruning procedure can be described as follows. In the beginning, we perform a preprocessing to obtain the bounding box and then add T_{top} and T_{btm} into $\mathcal{T}$. Now, given a vertical line L, whether to prune its left or right plane can be determined by the following steps.

1. If L does not intersect the bounding box, prune the half-plane not containing the box.
2. Compute x_D and x_U on L.

3. Find a sideward wedge or pseudo wedge via three forementioned phases. (Terminate whenever a strong or conditional $(1|1)_R$-centroid is found.)
   a. If breakpoints exist between x_D and x_U, pick any of them and check it.
   b. If no such breakpoint, decide whether L is non-leaning by checking x_B.
   c. If L is non-leaning, compute $PW(x_U)$ or $PW(x_D)$ depending on which of x_U and x_D has smaller weight loss.
4. Prune the right or left plane of L according to the direction of the sideward wedge or pseudo wedge.

The correctness of this procedure follows from the developed lemmas. Any vertical line not intersecting the bounding box is trivially dealt with in Step 1, due to the property of the box. When L intersects the box, by Lemma 22, x_D and x_U can certainly be found in Step 2. The three sub-steps of Step 3 correspond to the three searching phases. When L is not non-leaning, a sideward wedge is found, either at some breakpoint between x_D and x_U in Step 3(a) by Corollary 15, or at x_B in Step 3(b) by Lemma 17. Otherwise, according to Lemma 20 or its symmetric version, a pseudo wedge can be built in Step 3(c) for x_U or x_D, respectively. In Step 4, whether to prune the left or right plane of L can be determined via the just-found sideward wedge or pseudo wedge, by respectively Lemma 9 or Corollary 21.

The time complexity of this procedure is analyzed as follows. Computing the bounding box takes $O(n)$ time. In Step 2, x_D and x_U can be found by using the binary-search discussed in 3.2. Although the algorithm is not designed for this purpose, a slightly modification to its objective satisfies our need, and Step 2 can be done in $O(n \log^2 n)$ time by Lemma 13.

In Step 3(a), all breakpoints between x_D and x_U can be obtained in $O(n \log n)$ time as follows. As done in Lemma 13, we list all breakpoints on L as $O(n)$ sorted sequences, and prune breakpoints not in G_{DU} from each sequence by binary search. In Step 3(a) or 3(b), checking a picked point is done in $O(n \log n)$ time by invoking Lemma 10. The pseudo wedge $PW(x_U)$ or $PW(x_D)$ in Step 3(c) can be computed in $O(n \log n)$ time by using a sweeping technique to find the angle θ_U satisfying Lemma 20, or symmetrically θ_D, in $O(n \log n)$ time. Summarizing the above, these steps require $O(n \log^2 n)$ time in total. Since the invocation of Lemma 13 needs an additional $O(n^2 \log n)$-time preprocessing, we have the following result.

▶ **Lemma 23.** *With an $O(n^2 \log n)$-time preprocessing, whether to prune the right or left plane of a given vertical line L can be determined in $O(n \log^2 n)$ time.*

4.2 Searching on the Euclidean Plane

In this subsection, we come back to the $(1|1)_R$-centroid problem. Recall that, by Lemma 4, at least one $(1|1)_R$-centroid can be found in the three sets of intersection points $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{T}$, and $\mathcal{C}(V) \times \mathcal{C}(V)$, which consist of total $O(n^4)$ points. Let $\mathcal{L}$ denote the set of all vertical lines passing through these $O(n^4)$ intersection points. By definition, there exists a vertical line $L^* \in \mathcal{L}$ such that its local optimal point is a $(1|1)_R$-centroid. Conceptually, with the help of Lemma 23, L^* can be derived by applying prune-and-search approach to $\mathcal{L}$. However, it costs too much to explicitly generate and maintain the $O(n^4)$ lines. In the following, we show how to implicitly maintain these lines, by dealing with each of the above three sets separately, so that prune-and-search approaches can be applied.

Let $\mathcal{L}_\mathcal{T}$, $\mathcal{L}_\mathcal{M}$, and $\mathcal{L}_\mathcal{C}$ be the sets of all vertical lines passing through the intersection points in $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{T}$, and $\mathcal{C}(V) \times \mathcal{C}(V)$, respectively. A *local optimal line* of $\mathcal{L}_\mathcal{T}$ is a vertical line $L_t^* \in \mathcal{L}_\mathcal{T}$, such that its local optimal point has weight loss no larger than those of other lines in $\mathcal{L}_\mathcal{T}$. The local optimal lines L_m^* and L_c^* can be similarly defined for $\mathcal{L}_\mathcal{M}$ and $\mathcal{L}_\mathcal{C}$, respectively. We will adopt different prune-and-search techniques to find the local optimal lines of the three sets, so that a $(1|1)_R$-centroid can be found on one of them. Since some of the algorithms are fairly complicated, due to page limit, we provide only the overview of our approaches in the following.

To deal with up to $O(n^4)$ vertical lines in $\mathcal{L}_\mathcal{T}$, we apply the ingenious idea of parametric search via parallel sorting algorithms, proposed by Megiddo [15]. In this approach, the process of pruning vertical lines in $\mathcal{L}_\mathcal{T}$ to find L_t^* is reduced to the problem of sorting the $O(n^2)$ lines of $\mathcal{T}$ according to their intersection points on the undetermined vertical line L_t^*, in which each comparison between two lines of $\mathcal{T}$ can be resolved by deciding whether L_t^* is to the right or left of their intersection point. Obviously, the decision can be done by invoking Lemma 23 on the vertical line passing through the point.

Given a batch of k such comparisons, Megiddo showed how to resolve them in $O(k+\tau \log k)$ time, where τ is the time required to resolve one comparison. Then, he found that executing parallel sorting algorithms in a sequential way serves as good batching schemes. For example, the parallel merge sort algorithm [3] sorts N_1 items in $O(\log N_1)$ parallel steps on $O(N_1)$ processors. Executing this algorithm sequentially forms a sorting framework that takes $O(\log N_1)$ iterations, in each of which a batch of $k = O(N_1)$ comparisons has to be resolved. Thus, by letting $N_1 = |\mathcal{T}|$ and $\tau = O(n \log^2 n)$, our sorting problem can be solved in $O((k + \tau \log k) \times \log N_1) = O(n^2 \log n)$ time.

▶ **Lemma 24.** *A local optimal line L_t^* of $\mathcal{L}_\mathcal{T}$ can be found in $O(n^2 \log n)$ time.*

By similar observation as made in Lemma 12, for any two points $u, v \in V$, $C_\gamma(u) \times \mathcal{T}^r(v)$ and $C_\gamma(u) \times \mathcal{T}^l(v)$ can be represented by $O(1)$ consecutive subsequences of $P(v)$ in $O(\log n)$ time. Thus, for $\mathcal{C}(V) \times \mathcal{T}$ we can construct in $O(n^2 \log n)$ time $O(n^2)$ sequences of $O(n)$ breakpoints, each sorted in increasing x-coordinates. Correspondingly, $\mathcal{L}_\mathcal{M}$ can be represented by $O(n^2)$ sorted sequences of vertical lines. Then, finding L_m^* can be done by applying prune-and-search to the $O(n^2)$ sequences of vertical lines via parallel binary searches, like in Lemma 13, which takes $O(\log n)$ iterations and $O(n^2 + n \log^2 n)$ time per iteration.

▶ **Lemma 25.** *A local optimal line L_m^* of $\mathcal{L}_\mathcal{M}$ can be found in $O(n^2 \log n)$ time.*

Since $|\mathcal{C}(V) \times \mathcal{C}(V)| = O(n^2)$, a sorted sequence of $\mathcal{L}_\mathcal{C}$ can be obtained in $O(n^2 \log n)$ time. Then, L_c^* can be easily found by binary search with Lemma 23 in $O(n \log^3 n)$ time.

▶ **Lemma 26.** *A local optimal line L_c^* of $\mathcal{L}_\mathcal{C}$ can be found in $O(n^2 \log n)$ time.*

By definition, L^* can be found among L_t^*, L_m^*, and L_c^*, which can be computed in $O(n^2 \log n)$ time by Lemmas 24, 25, and 26, respectively. Then, a $(1|1)_R$-centroid can be computed as the local optimal point of L^* in $O(n \log^2 n)$ time by Lemma 13. Combining with the $O(n^2 \log n)$-time preprocessing for computing the angular sorted sequence $P(v)$s and the bounding box enclosing $\mathcal{C}(V)$, we have the following theorem.

▶ **Theorem 27.** *The $(1|1)_R$-centroid problem can be solved in $O(n^2 \log n)$ time.*

5 Concluding Remarks

In this paper, we revisited the (1|1)-centroid problem on the Euclidean plane under the consideration of minimal distance constraint between facilities, and proposed an $O(n^2 \log n)$-time algorithm, which closes the bound gap between this problem and its unconstrained version. Starting from a critical observation on the medianoid solutions, we developed a pruning tool with indefinite region remained after pruning, and made use of it via multi-level structured parametric search approach, which is different to the previous approach in [6, 12].

Considering distance constraint between facilities in various competitive facility location models is both of theoretical interest and of practical importance. However, similar constraints are rarely seen in the literature. It would be good starting points by introducing the constraint to the facilities between players in the $(r|X_p)$-medianoid and $(r|p)$-centroid problems, maybe even to the facilities between the same player.

References

1 Aritra Banik, Jean-Lou De Carufel, Anil Maheshwari, and Michiel Smid. Discrete voronoi games and ε-nets, in two and three dimensions. *Computational Geometry*, 55:41–58, 2016.

2 Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, January 1987.

3 Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

4 Abdullah Dasci. Conditional location problems on networks and in the plane. In Horst A. Eiselt and Vladimir Marianov, editors, *Foundations of Location Analysis*, pages 179–206. Springer US, Boston, MA, 2011.

5 Ivan Davydov, Yury Kochetov, and Alexandr Plyasunov. On the complexity of the $(r|p)$-centroid problem in the plane. *TOP*, 22(2):614–623, 2014.

6 Zvi Drezner. Competitive location strategies for two facilities. *Regional Science and Urban Economics*, 12(4):485–493, 1982.

7 Zvi Drezner and E. Zemel. Competitive location in the plane. *Annals of Operations Research*, 40(1):173–193, 1992.

8 Horst A. Eiselt and Gilbert Laporte. Sequential location problems. *European Journal of Operational Research*, 96(2):217–231, 1997.

9 Horst A. Eiselt, Gilbert Laporte, and Jacques-Francois Thisse. Competitive location models: a framework and bibliography. *Transportation Science*, 27(1):44–54, 1993.

10 Horst A. Eiselt, Vladimir Marianov, and Tammy Drezner. Competitive location models. In Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama, editors, *Location Science*, pages 365–398. Springer International Publishing, Cham, 2015.

11 S. Louis Hakimi. On locating new facilities in a competitive environment. *European Journal of Operational Research*, 12(1):29–35, 1983.

12 S. Louis Hakimi. Locations with spatial interactions: competitive locations and games. In Pitu B. Mirchandani and Richard L. Francis, editors, *Discrete location theory*, pages 439–478. Wiley, 1990.

13 Harold Hotelling. Stability in competition. *The Economic Journal*, 39(153):41–57, 1929.

14 D. T. Lee and Y. F. Wu. Geometric complexity of some location problems. *Algorithmica*, 1(1):193–211, 1986.

15 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, October 1983.

16 Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

17 Frank Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

18 Angelika Reiser. A linear selection algorithm for sets of elements with weights. *Information Processing Letters*, 7(3):159–162, 1978.

19 D. R. Santos-Peñate, R. Suárez-Vega, and P. Dorta-González. The leader–follower location model. *Networks and Spatial Economics*, 7(1):45–61, 2007.